Chapter 3

Models Characterization and Tuning

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.

3.1 Generalization

Without loss of generality, and for simplicity, uni-variate definition is used. As we already stated, our data is a set of (x_k, y_k) pairs. Let's assume that the dataset $(\mathbf{x}, \mathbf{y})$ are M samples drawn from some (unknown) joint probability distribution, $(x, y) \sim \mathcal{D}$. In practice, the value of M is (very) limited.
When we use some data to "train" some model, $f(\mathbf{X}; \mathbf{w})$, the question is, how can we assess the performance on other points from $\mathcal{D}$.
Generalization: The difference between performance metrics over data that is used to train model (train performance) and performance metric over all (theoretically) possible points from $\mathcal{D}$ (generalization error),

$$p_{gen} = E_{\mathcal{D}}[J(y, \hat{y})]. \tag{3.1}$$

The problem is that the distribution $\mathcal{D}$ is unknown in most of the practical applications. In the following, method to approximate the generalization probability will be discussed.
The LS example performance is presented in Fig. 3.1.
Notes:

- Better generalization $\Rightarrow$ smaller difference between model performance and generalization performance.
- Can be evaluated theoretically only for very simple models and datasets.
- Require some practical assessment tools, as follows.

3.2 Polynomial model

Goal:
- Extension of a linear model "engine" to polynomial models. The polynomial model is very flexible, i.e. due to the Taylor expansion theorem.
- Illustration of generalization principle.

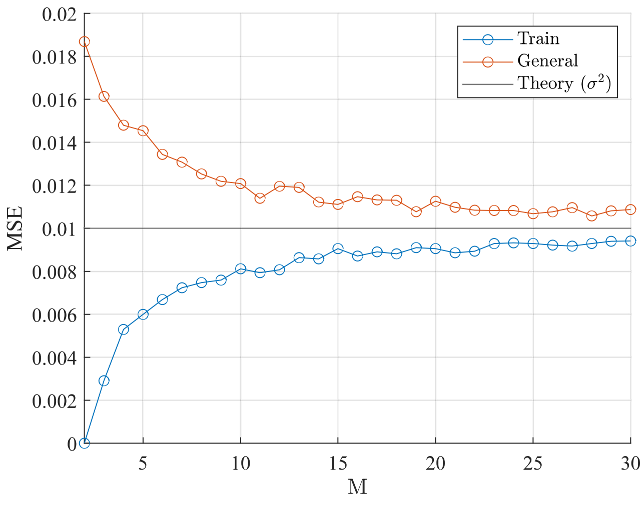


Figure 3.1: The difference between the results in Fig. 2.2 and the "realistic" performance.

The N-degree uni-variate polynomial model is

$$\begin{aligned}\hat{y} = f(x; \mathbf{w}) &= w_0 + w_1 x + w_2 x^2 + \cdots + w_N x^N \\ &= \sum_{j=0}^{N} w_j x^j\end{aligned} \tag{3.2}$$

The corresponding prediction is

$$\hat{y_k} = \sum_{j=0}^{N} w_j x_k^j \tag{3.3}$$

This problem is linear by change of variables, $z_{kj} = x_k^j$,

$$\hat{y_k} = \sum_{j=0}^{N} w_j z_{kj} \tag{3.4}$$

Using matrix notation (also termed Vandermonde matrix),

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^N \\ 1 & x_2 & x_2^2 & \cdots & x_2^N \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_M & x_M^2 & \cdots & x_M^N \end{bmatrix} \tag{3.5}$$

the weights values are straightforward.

3.3 Overfitting and underfitting

Goal: Two common and fundamental problems in machine learning.

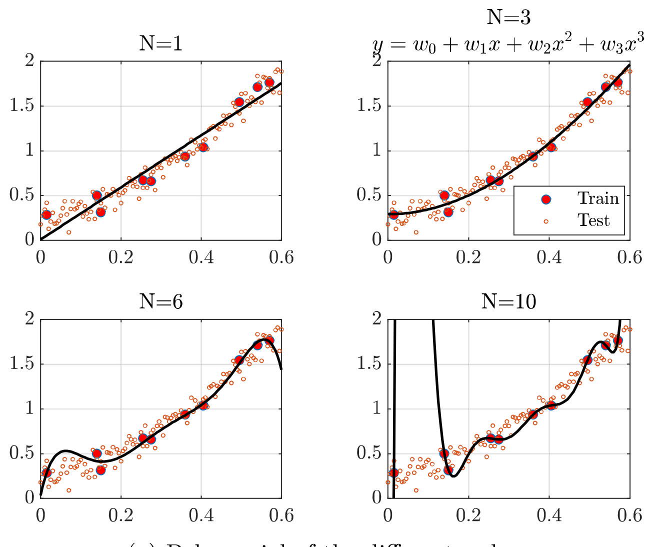
Overfitting when model is too complex, i.e. have too many parameters.

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

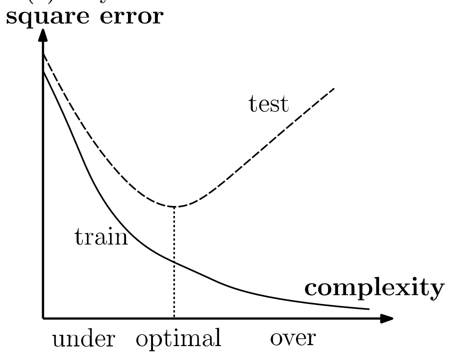
Underfitting happens when a model is too simple.

- 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.

Overfitting and underfitting are complimentary and balancing between them is key to building robust machine learning models that perform well on new, unseen data, i.e. generalize well.



(a) Polynomial of the different order.



(b)

Figure 3.2: (a) Overfitting and underfitting polynomial example. (b) 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.

Hyper-parameter optimization The order N is the hyper-parameter of the polynomial model. Selecting the most appropriate hyper-parameters value is called hyper-parameter optimization.

3.4 Cross-validation

Goal: Trial and error approach to quantify generalization performance and overfitting-underfitting balance.

The cross-validation is also termed performance assessment.

- First step of any following technique is resample the dataset into the **random order**.

Big dataset (tens of thousands): train/validation/test

Split into three *distinctive* datasets:

- *Training* (50-80%): used for learning of model parameters, e.g. weights $\mathbf{w}$.
- *Validation* (10-25%): used for assessment of model hyper-parameters influence.
- *Test* (10-25%): performance assessment that is supposed to be an estimation for the generalization error.

Medium datasets (hundreds to thousandths): k-fold

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.

Very small datasets (tens to hundreds): one-hold-out

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

3.4.1 Summary

Model performance insights from the differences between train and test datasets and undefitting-overfitting trade-off are presented in Fig. 3.2(a).

3.5 Noisy deterministic function interpretation and bias-variance trade-off

The general model beneath the dataset model is

$$y = h(\mathbf{x}) + \epsilon, \tag{3.6}$$

where $f(\mathbf{x})$ is some unknown function and ϵ is some random noise with unknown distribution and with $E[\epsilon] = 0, \mathrm{Var}[\epsilon] = \sigma^2$.
Prediction is by $\hat{y}_i = f(\mathbf{x}_i; \boldsymbol{\theta})$ and the resulting mse loss of the model is

$$\begin{aligned}
\mathcal{L} &= E\left[(\hat{y} - y)^2\right] = \frac{1}{M}\sum_{i=1}^{M}(\hat{y}_i - y_i)^2 \\
&= E\left[\hat{y}^2\right] - 2E[y]\,E[\hat{y}] + E\left[y^2\right] \\
E\left[\hat{y}^2\right] &= \mathrm{Var}[\hat{y}] + E^2[\hat{y}] \\
E[y] &= E\left[h(\mathbf{x})\right] = h(\mathbf{x}) \\
E\left[y^2\right] &= E\left[(h(\mathbf{x}) + \epsilon)^2\right] = \frac{1}{M}\sum_{i=1}^{M}\left(h(\mathbf{x}_i) + \epsilon_i\right)^2 \\
&= E\left[h^2(\mathbf{x})\right] + 2E\left[h(\mathbf{x})\right]\cancelto{0}{E[\epsilon]} + E\left[\epsilon^2\right] \\
&= E\left[h^2(\mathbf{x})\right] + \sigma^2 \\
&= h^2(\mathbf{x}) + \sigma^2 \\
\mathcal{L} &= \mathrm{Var}[\hat{y}] + E^2[\hat{y}] - 2h(\mathbf{x})E[\hat{y}] + h^2(\mathbf{x}) + \sigma^2 \\
&= \underbrace{\left(E[\hat{y}] - h(\mathbf{x})\right)^2}_{bias^2} + \underbrace{\mathrm{Var}[\hat{y}]}_{variance} + \underbrace{\sigma^2}_{noise}
\end{aligned} \tag{3.7}$$

The visualizaion of this principle for polynomial regression is presented in Fig. 3.3. Underfitting is low variance and high bias, and overfitting is high variance and low bias.

3.6 Takeaways

3.1 The goal is to implement polynomial regression and find the optimal N for the given problem.

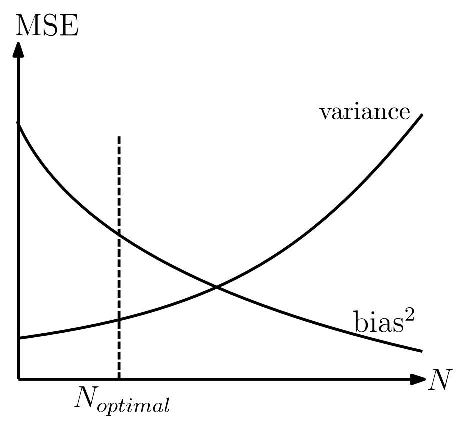


Figure 3.3: Bias-variance trade-off for polynomial regression.

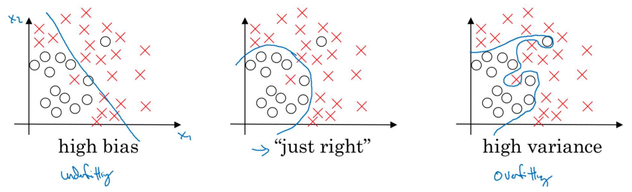


Figure 3.4: Overfitting and underfitting at classification bounds.

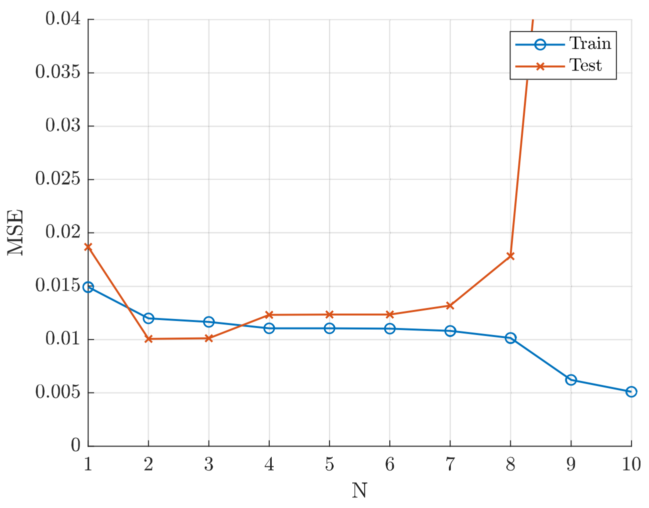


Figure 3.5: Polynomial regression: note the difference between train and test performance.

Chapter 4

Overfitting Management

Goal: Typically, the most applied models (beyond the basic linear one) in their general form are overfit data. The goal is to provide the list of the methods that are used to manage the overfit.

4.1 Dataset size

Goal: Dataset level in Fig. 1.1.

Adding data usually improves the performance for an overly complex system as presented in Fig. 4.1.

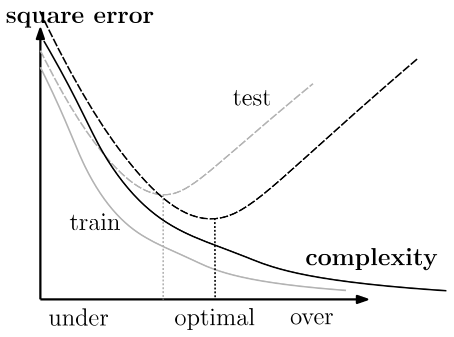


Figure 4.1: Bias-variance trade-off for polynomial regression.

4.2 Regularization

Goal:
- Tweak the bias-variance trade-off by penalizing weights size.
- Loss function level in Fig. 1.1.
- Introduces new hyper-parameter that have to be tuned.

Regularization: Penalty to the loss function

$$\mathcal{L}_{\text{new}} = \mathcal{L} + \lambda g(\mathbf{w}) \tag{4.1}$$

λ is termed *regularization parameter*.
L_1**-regularization**: Special case of $g(\cdot)$, where

$$g(\mathbf{w}) = \frac{\lambda}{2M}\|\mathbf{w}\|_1 = \frac{\lambda}{2M}\sum_{i=1}^{N}|w_i| \tag{4.2}$$

L_2**-regularization**: Special case of $g(\mathbf{w}) = \frac{1}{2M}\|\mathbf{w}\|^2$,

$$\mathcal{L}_{\text{new}} = \mathcal{L} + \lambda\mathbf{w}^T\mathbf{w}\frac{\lambda}{2M}\sum_{i=1}^{N} w_i^2 \tag{4.3}$$

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

4.2.1 Ridge Regression

Ridge regression: L_2-regularized linear regression.

$$\begin{aligned}\mathcal{L}(\mathbf{w}) &= \frac{1}{2M}\|\mathbf{y}-\mathbf{X}\mathbf{w}\|^2 + \frac{\lambda}{2M}\underbrace{\|\mathbf{w}\|^2}_{\sum_{i=1}^{N} w_i^2} \\ &= \frac{1}{2M}(\mathbf{y}-\mathbf{X}\mathbf{w})^T(\mathbf{y}-\mathbf{X}\mathbf{w}) + \frac{\lambda}{2M}\mathbf{w}^T\mathbf{w}\end{aligned} \tag{4.4}$$

Derivative

$$\nabla\mathcal{L}(\mathbf{w}) = \frac{1}{M}\left(-\mathbf{X}^T(\mathbf{y}-\mathbf{X}\mathbf{w}) + \lambda\mathbf{w}\right) = 0 \tag{4.5}$$

Solution

$$-\mathbf{X}^T(\mathbf{y}-\mathbf{X}\mathbf{w}) + \lambda\mathbf{w} = -\mathbf{X}^T\mathbf{y} + \mathbf{X}^T\mathbf{X}\mathbf{w} + \lambda\mathbf{w} = 0$$
$$\mathbf{X}^T\mathbf{y} = \mathbf{X}^T\mathbf{X}\mathbf{w} + \lambda\mathbf{w}$$

$$\mathbf{w} = \left(\lambda\mathbf{I} + \mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}, \tag{4.6}$$

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

$$\tilde{y} = \tilde{\mathbf{X}}\mathbf{w}$$
$$\begin{bmatrix}\mathbf{y}\\0\\\vdots\\0\end{bmatrix} = \begin{bmatrix}- & \mathbf{X} & -\\ \sqrt{\lambda} & & 0\\ & \ddots & \\ 0 & & \sqrt{\lambda}\end{bmatrix}\begin{bmatrix}w_1\\\vdots\\w_N\end{bmatrix} + w_0 \tag{4.7}$$

4.2.2 General aspects

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.

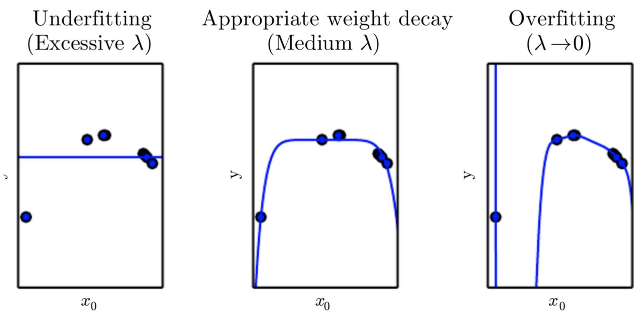


Figure 4.2: Illustration of λ influence on model fitting.

Polynomial interpretation One-dimensional dataset (x_i, y_i) with nonlinear mapping $\mathbf{x}_i = [x_i, x_i^2, \ldots, x_i^N]$. The resulting coefficients w_i will be significantly higher for higher i (Fig. 4.3). Reducing them results in smooth $\hat{y}$ prediction.

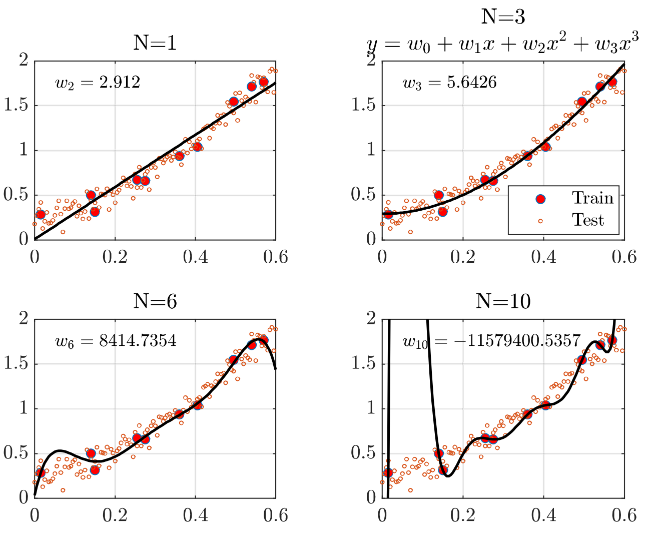


Figure 4.3: Illustration of weight grows with the polynomial model complexity.

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

4.3 Normalization and Standardization

Goal: Data pre-processing level in Fig. 1.1. Used for multi-variate data.

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

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

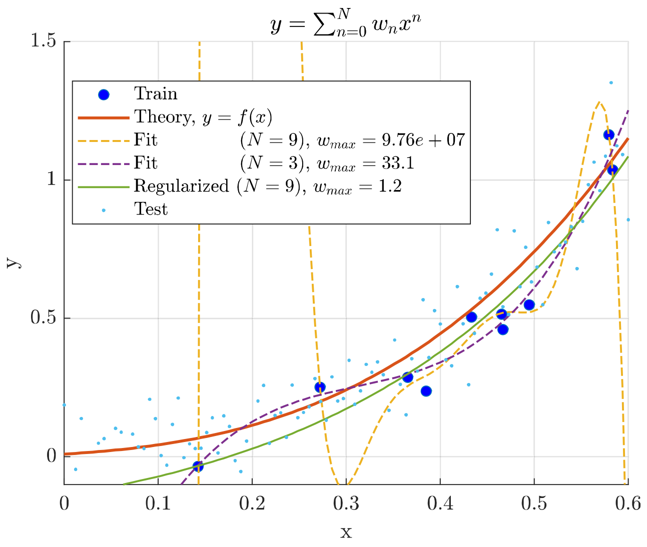


Figure 4.4: Illustration of regularization influence on the polynomial model.

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

$$z_{std} = \frac{\mathbf{x} - \bar{\mathbf{x}}}{\sigma_{\mathbf{x}}}, \tag{4.8}$$

where

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

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

Implementation steps:

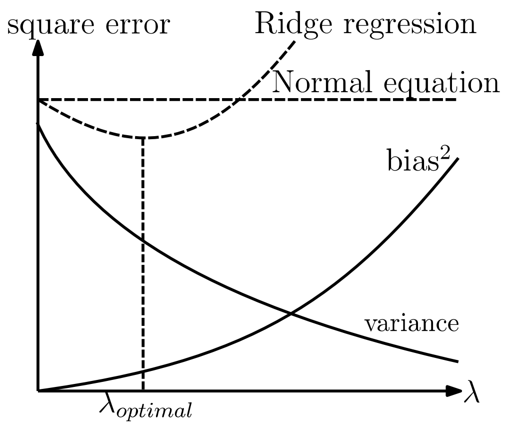
1. On **train** dataset, evaluate $\bar{\mathbf{x}}$ and $\sigma_{\mathbf{x}}$.
2. Apply normalization on **train** dataset, using $\bar{\mathbf{x}}$ and $\sigma_{\mathbf{x}}$.
3. Apply normalization on **test** dataset, using **same** $\bar{\mathbf{x}}$ and $\sigma_{\mathbf{x}}$ (no recalculation).

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

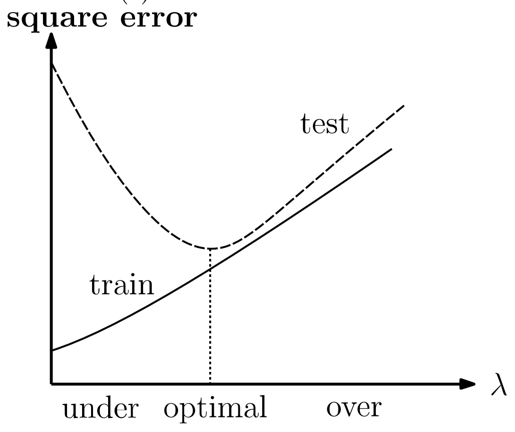
$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{4.9}$$

Implementation steps for normalization are similar to standardization.

Beware, normalization and standardization are used interchangeably.



(a) Performance trade-off



(b) Train/test

Figure 4.5: Ridge regression: visualization of bias-variance trade-off.