18-661 Introduction to Machine Learning

Overfitting and the Bias-Variance Trade-off

Spring 2020

ECE – Carnegie Mellon University

Announcements

- HW 2 will be released today, due February 10.
- Recitation on Friday will cover linear regression, gradient descent, and the bias-variance tradeoff (today's lecture). These problems will be helpful for HW 2.
- There has been some student demand for a Python and Jupyter tutorial.
  - SV will include this material at the end of Friday's recitation.
  - Pittsburgh will hold the tutorial tomorrow and broadcast to Rwanda around 1pm ET (rooms TBD).
- HW 1 solutions will be posted on Canvas later this week.

Outline

1. Review of Ridge Regression
2. Review of Non-linear Basis Functions
3. Overfitting and Regularization
4. Hyperparameter Tuning and Cross-Validation
5. Bias-Variance Trade-off

Review of Ridge Regression

What can go wrong with the LMS solution?

$$\mathbf{w}^{LMS} = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y}$$

Why might $\boldsymbol{X}^\top \boldsymbol{X}$ be non-invertible?

- **Answer 1:** N $<$ D. Not enough data to estimate all parameters.
- **Answer 2:** Columns of $\boldsymbol{X}$ are not linearly independent, e.g., some features are linear functions of other features. In this case, solution is not unique. Examples:
  - A feature is a re-scaled version of another, for example, having two features correspond to length in meters and feet respectively
  - Same feature is repeated twice – could happen when there are many features
  - A feature has the same value for all data points
  - Sum of two features is equal to a third feature

Example: Matrix $X^\top X$ is not invertible

sqft (1000's)	bathrooms	sale price (100k)
1	2	2
2	2	3.5
1.5	2	3
2.5	2	4.5

Design matrix and target vector:

$$\mathbf{X} = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 2 \\ 1 & 1.5 & 2 \\ 1 & 2.5 & 2 \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} 2 \\ 3.5 \\ 3 \\ 4.5 \end{bmatrix}$$

The 'bathrooms' feature is redundant, so we don't need w_2

$$\begin{aligned} y &= w_0 + w_1 x_1 + w_2 x_2 \\ &= w_0 + w_1 x_1 + w_2 \times 2, \quad \text{since } x_2 \text{ is always 2!} \\ &= w_{0,eff} + w_1 x_1, \quad \text{where } w_{0,eff} = (w_0 + 2w_2) \end{aligned}$$

Ridge regression

Intuition: what does a non-invertible $\boldsymbol{X}^\top \boldsymbol{X}$ mean?
Consider the SVD of this matrix:

$$\boldsymbol{X}^\top \boldsymbol{X} = \boldsymbol{V} \begin{bmatrix} \lambda_1 & 0 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & \cdots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 \\ 0 & \cdots & \cdots & \lambda_r & 0 \\ 0 & \cdots & \cdots & 0 & 0 \end{bmatrix} \boldsymbol{V}^\top$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \lambda_r > 0$, $r < \mathrm{D}$, and $\boldsymbol{V}$ is a unitary matrix (its transpose is its inverse). We will need to divide by zero to compute $(\boldsymbol{X}^\top \boldsymbol{X})^{-1}$...
Fix the problem: ensure all singular values are non-zero:

$$\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I} = \boldsymbol{V} \mathrm{diag}(\lambda_1 + \lambda, \lambda_2 + \lambda, \cdots, \lambda) \boldsymbol{V}^\top$$

where $\lambda > 0$ and $\boldsymbol{I}$ is the identity matrix.

Regularized least squares (ridge regression)

Solution

$$\boldsymbol{w} = \left(\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{X}^\top \boldsymbol{y}$$

This is equivalent to adding an extra term to $RSS(\boldsymbol{w})$

$$\overbrace{\frac{1}{2}\left\{\boldsymbol{w}^\top \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{w} - 2\left(\boldsymbol{X}^\top \boldsymbol{y}\right)^\top \boldsymbol{w}\right\}}^{RSS(\boldsymbol{w})} + \underbrace{\frac{1}{2}\lambda \|w\|_2^2}_{\text{regularization}}$$

Benefits

- Numerically more stable, invertible matrix
- Force **w** to be small
- Prevent overfitting — more on this later

Applying this to our example

sqft (1000's)	bathrooms	sale price (100k)
1	2	2
2	2	3.5
1.5	2	3
2.5	2	4.5

The 'bathrooms' feature is redundant, so we don't need w_2

$$
\begin{aligned}
y &= w_0 + w_1 x_1 + w_2 x_2 \\
&= w_0 + w_1 x_1 + w_2 \times 2, && \text{since } x_2 \text{ is always 2!} \\
&= w_{0,eff} + w_1 x_1, && \text{where } w_{0,eff} = (w_0 + 2w_2) \\
&= 0.45 + 1.6 x_1 && \text{Should get this}
\end{aligned}
$$

Applying this to our example

The 'bathrooms' feature is redundant, so we don't need w_2

$$
\begin{aligned}
y &= w_0 + w_1 x_1 + w_2 x_2 \\
&= w_0 + w_1 x_1 + w_2 \times 2, \quad \text{since } x_2 \text{ is always 2!} \\
&= w_{0,eff} + w_1 x_1, \quad \text{where } w_{0,eff} = (w_0 + 2w_2) \\
&= 0.45 + 1.6 x_1 \quad \text{Should get this}
\end{aligned}
$$

Compute the solution for $\lambda = 0.5$

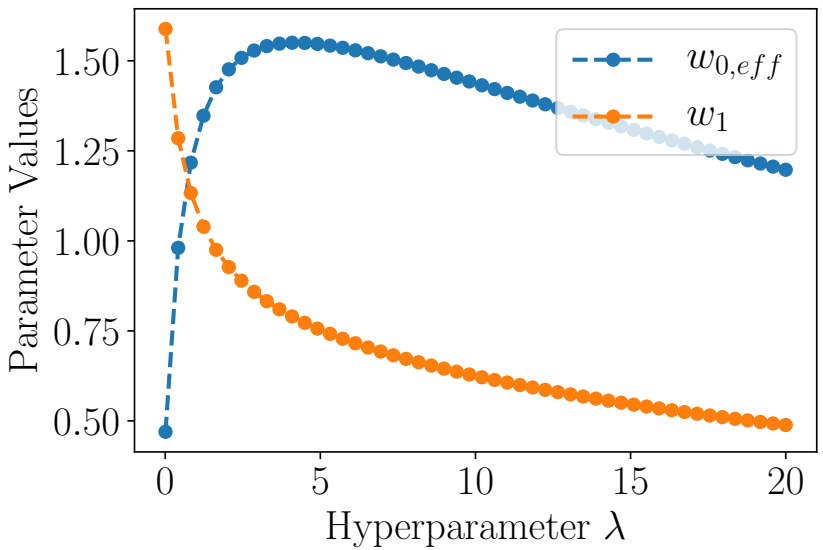
$$
\begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \left(\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I} \right)^{-1} \boldsymbol{X}^\top \boldsymbol{y}
$$

$$
\begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0.208 \\ 1.247 \\ 0.4166 \end{bmatrix}
$$

How does λ affect the solution?

$$\begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \left(\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I} \right)^{-1} \boldsymbol{X}^\top \boldsymbol{y}$$

Let us plot $w_o' = w_0 + 2w_2$ and w_1 for different $\lambda \in [0.01, 20]$

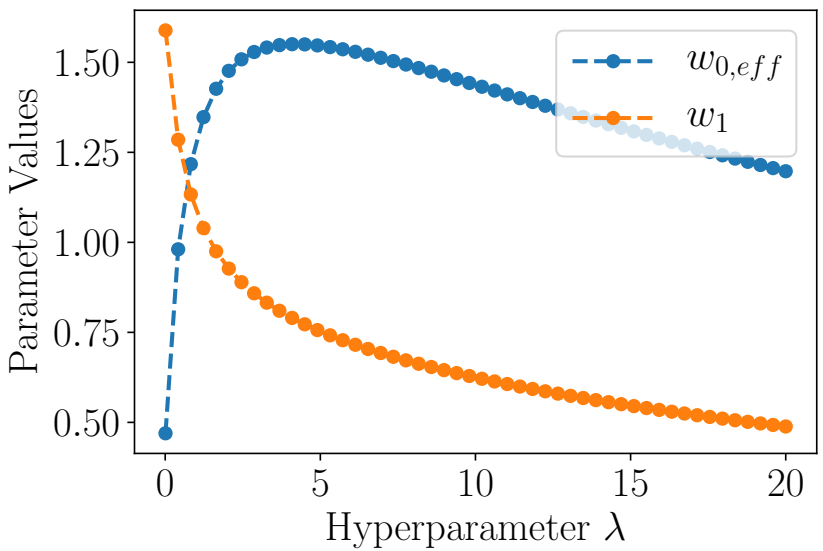


Setting small λ gives almost the least-squares solution ($w_o' = 0.45$ and $w_1 = 1.6$), but it can cause numerical instability in the inversion.

How to choose λ?

λ is referred as *hyperparameter*

- Associated with the estimation method, not the dataset
- In contrast $\boldsymbol{w}$ is the parameter vector
- Use validation set or cross-validation to find good choice of λ



Review of Non-linear Basis Functions

Is a linear modeling assumption always a good idea?

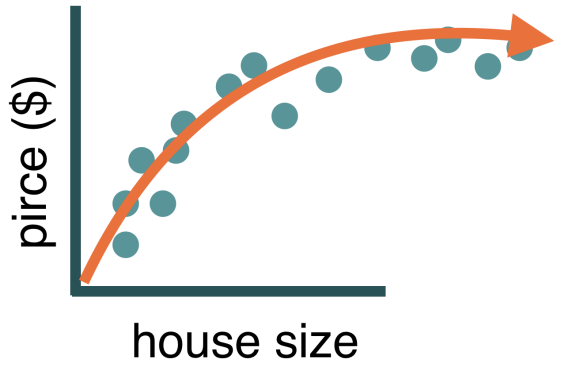


Figure 1: Sale price can saturate as sq.footage increases

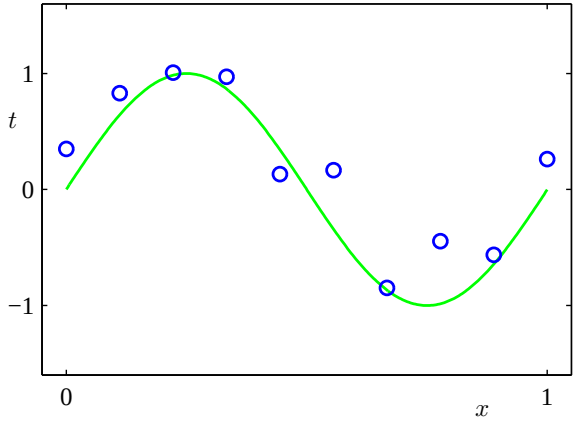


Figure 2: Temperature has cyclic variations over each year

General nonlinear basis functions

We can use a nonlinear mapping:

$$\boldsymbol{\phi}(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^D \rightarrow \boldsymbol{z} \in \mathbb{R}^M$$

- M is dimensionality of new features $\boldsymbol{z}$ (or $\boldsymbol{\phi}(\boldsymbol{x})$)
- M could be greater than, less than, or equal to D

We can apply existing learning methods on the transformed data:

- linear methods: prediction is based on $\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x})$
- other methods: nearest neighbors, decision trees, etc

Regression with nonlinear basis

Residual sum of squares

$$\sum_n [\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}_n) - y_n]^2$$

where $\boldsymbol{w} \in \mathbb{R}^M$, the same dimensionality as the transformed features $\boldsymbol{\phi}(\boldsymbol{x})$.

The LMS solution can be formulated with the new design matrix

$$\boldsymbol{\Phi} = \begin{pmatrix} \boldsymbol{\phi}(\boldsymbol{x}_1)^\top \\ \boldsymbol{\phi}(\boldsymbol{x}_2)^\top \\ \vdots \\ \boldsymbol{\phi}(\boldsymbol{x}_N)^\top \end{pmatrix} \in \mathbb{R}^{N \times M}, \quad \boldsymbol{w}^{\text{LMS}} = \left(\boldsymbol{\Phi}^\top \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^\top \boldsymbol{y}$$

Example: Lots of flexibility in designing features!

x_1, Area (1k sqft)	$\sqrt{x_1}$	Price (100k)
1	1	1
2.25	1.5	2
4	2	2.2
6.25	2.5	2.5

Price $= \sqrt{x_1}$ is more accurate than Price $= x_1$.

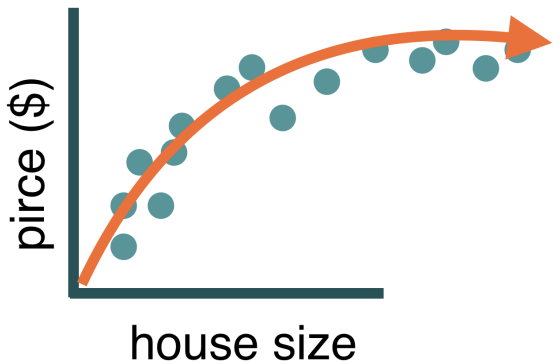


Figure 3: Add $\sqrt{x_1}$ as a feature to allow us to fit square-root, instead of linear, functions of the house area x_1.

Example: Lots of flexibility in designing features!

x_1, front (100ft)	x_2 depth (100ft)	$10x_1x_2$, Lot (1k sqft)	Price (100k)
0.5	0.5	2.5	2
0.5	1	5	3.5
0.8	1.5	12	3
1.0	1.5	15	4.5

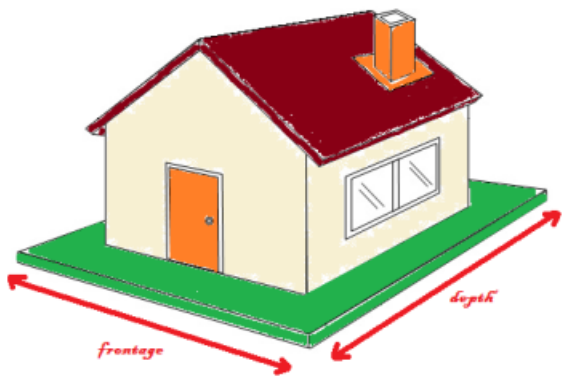


Figure 4: Instead of having frontage and depth as two separate features, it may be better to consider the lot-area, which is equal to frontage$\times$depth

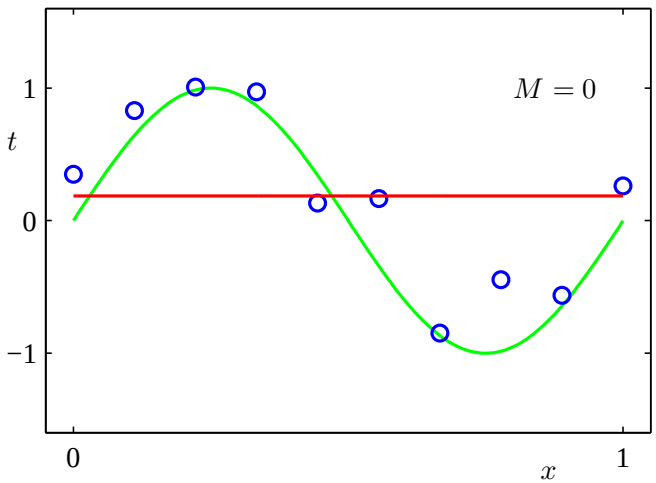
Overfitting and Regularization

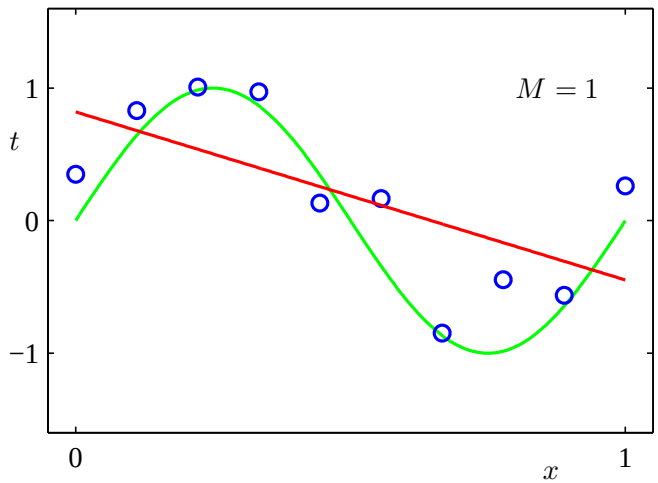
Non-linear basis functions: Polynomial regression

Polynomial basis functions

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^M \end{bmatrix} \Rightarrow f(x) = w_0 + \sum_{m=1}^{M} w_m x^m$$

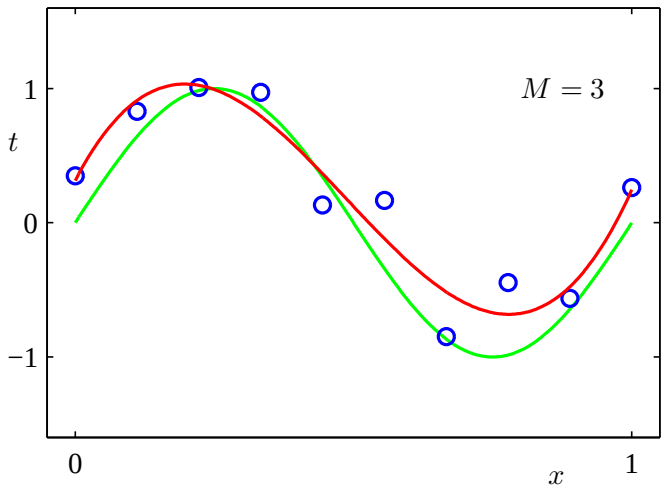
Fitting samples from a sine function:



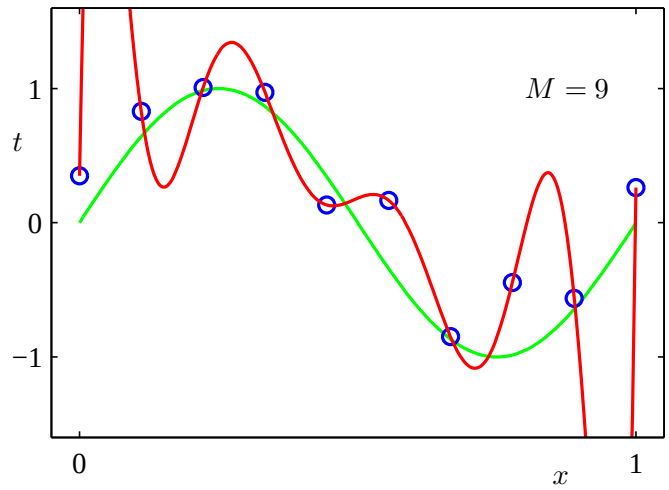


Adding high-order terms

M=3



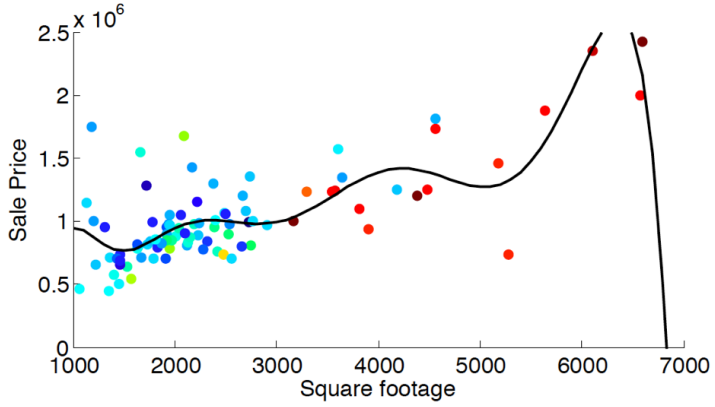
M=9: overfitting



More complex features lead to better results on the training data, but potentially worse results on new data, e.g., test data!

Overfitting can be quite disastrous

Fitting the housing price data with large M:



Predicted price goes to zero (and is ultimately negative) if you buy a big enough house!

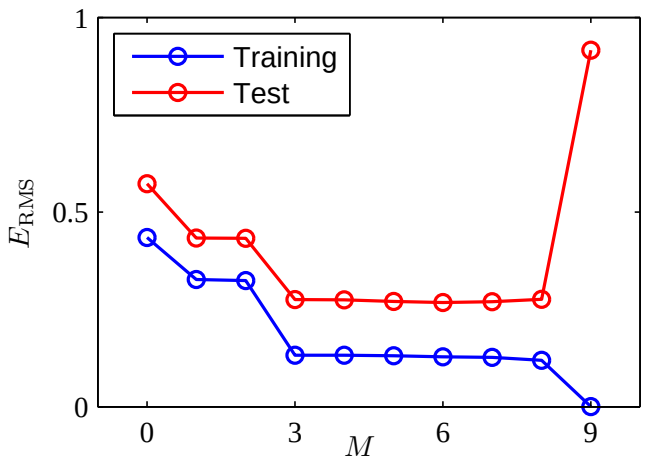
This is called poor generalization/overfitting.

Detecting overfitting

Plot model complexity versus objective function:

- X axis: model complexity, e.g., M
- Y axis: error, e.g., RSS, RMS (square root of RSS), 0-1 loss

Compute the objective on a training and test dataset.

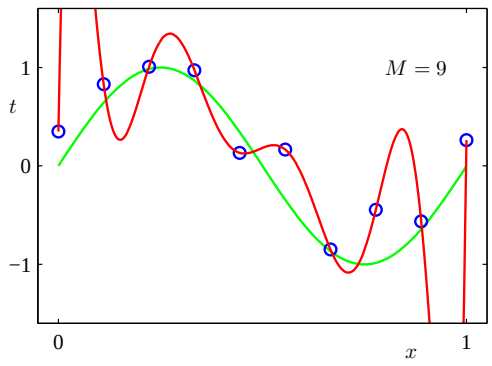


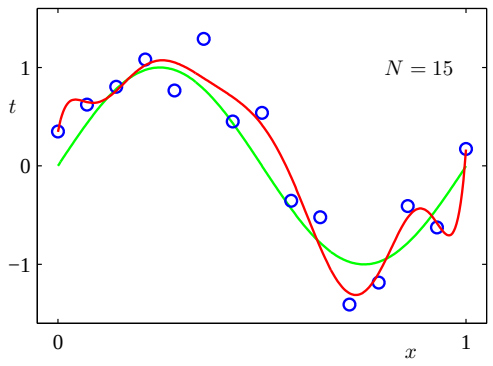
As a model increases in complexity:

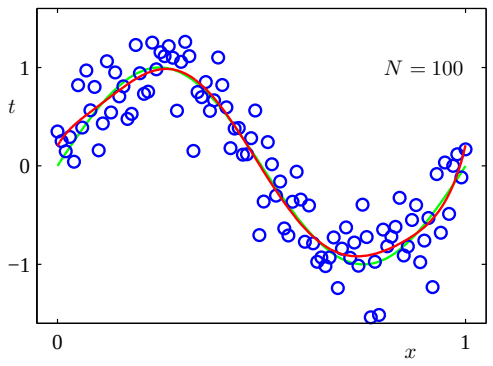
- Training error keeps improving
- Test error may first improve but eventually will deteriorate

Dealing with overfitting: Option 1

Try to use more training data



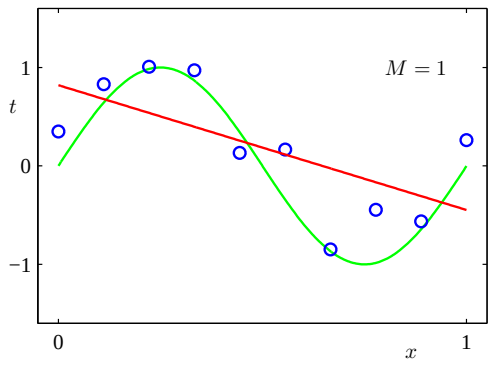


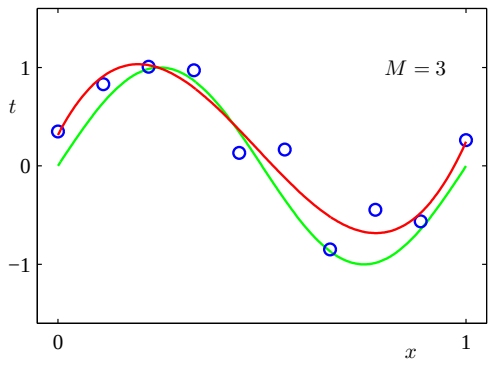


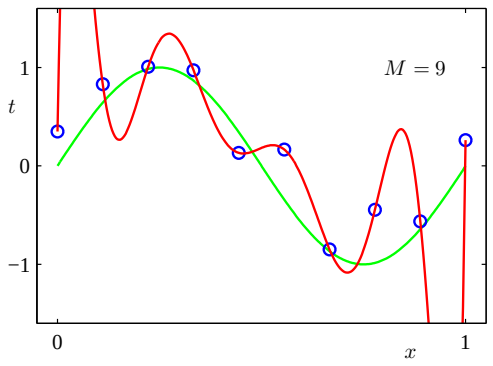
But getting a lot of data can be expensive and time-consuming

Dealing with overfitting: Option 2

Reduce the Number of Features







May not know which and how many features to remove

Dealing with overfitting: Option 3

Regularization Methods: Give preference to 'simpler' models

- How do we define a simple linear regression model — $\boldsymbol{w}^\top \boldsymbol{x}$?
- Intuitively, the weights corresponding to higher order terms should not be "too large"

	$M=0$	$M=1$	$M=3$	$M=9$
w_0	0.19	0.82	0.31	0.35
w_1		-1.27	7.99	232.37
w_2			-25.43	-5321.83
w_3			17.37	48568.31
w_4				-231639.30
w_5				640042.26
w_6				-1061800.52
w_7				1042400.18
w_8				-557682.99
w_9				125201.43

Regularization methods

Add a term to the objective function.

Choose the parameters to not just minimize risk, but avoid being large.

$$\frac{1}{2}\left\{\boldsymbol{w}^\top \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{w} - 2\left(\boldsymbol{X}^\top \boldsymbol{y}\right)^\top \boldsymbol{w}\right\} + \frac{1}{2}\lambda\|w\|_2^2$$

Ridge regression is just regularized linear regression.

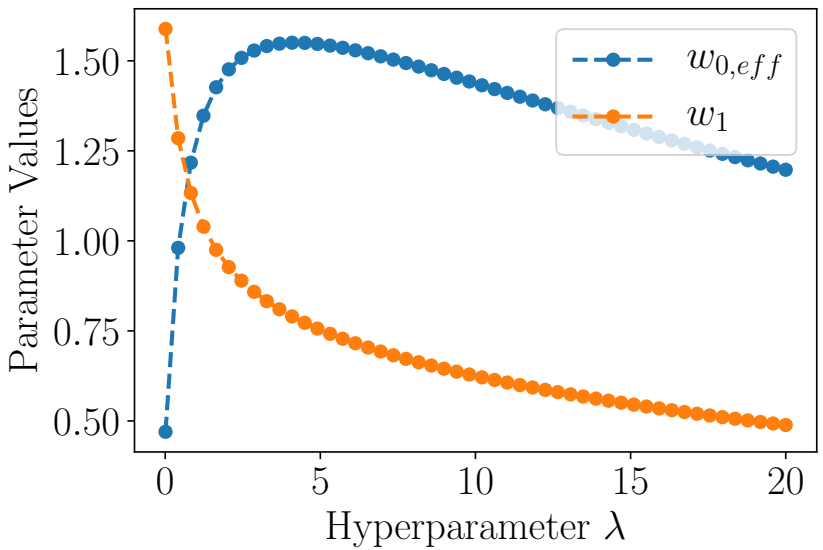
Advantages

- Forces the magnitude of $\boldsymbol{w}$ to be small
- Tries to find a simple model with few parameters
- Generalizes well to new data points

Ridge regression as regularization

$$\begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \left(\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I} \right)^{-1} \boldsymbol{X}^\top \boldsymbol{y}$$

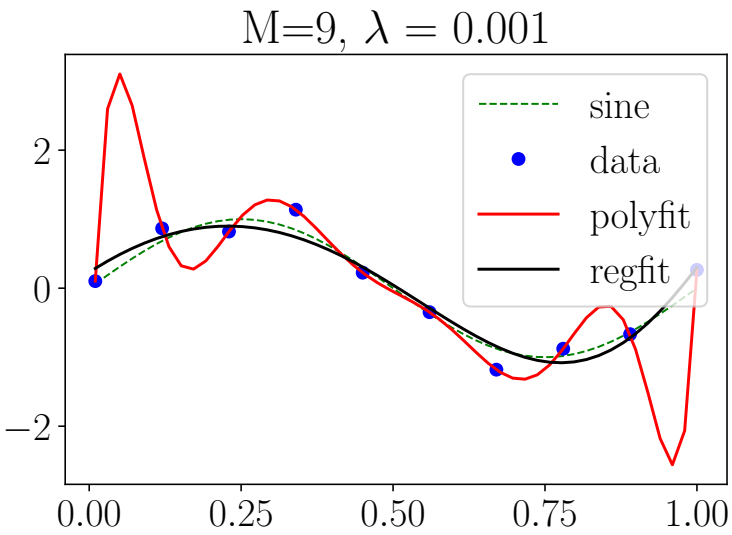
Let us plot $w_o' = w_0 + 2w_2$ and w_1 for different $\lambda \in [0.01, 20]$

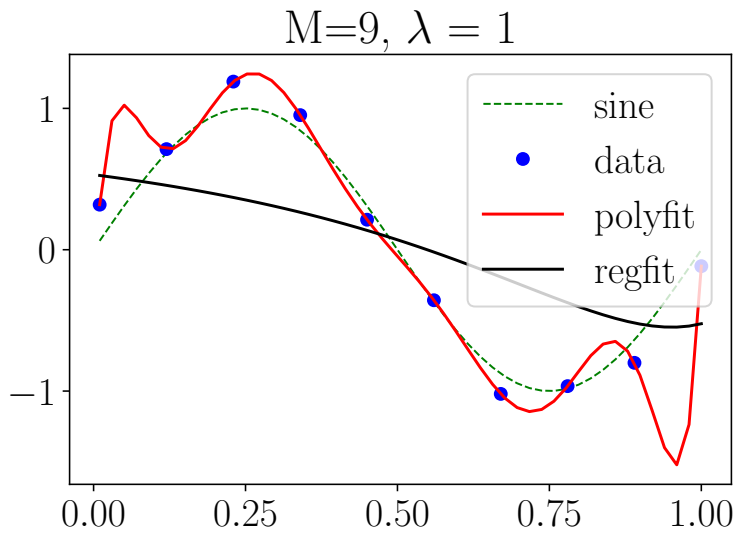


Setting small λ gives almost the least-squares solution, but it can cause numerical instability in the inversion

Example: Effect of regularization

- Regularization makes the higher order w_i's smaller
- Regularized polynomial fit will generalize much better
- As λ increases, the model becomes simpler





Probabilistic interpretation of regularization

Regularized Regression model: $Y = \boldsymbol{w}^\top \boldsymbol{X} + \eta$

- $Y \sim N(\boldsymbol{w}^\top \boldsymbol{X}, \sigma_0^2)$ is a Gaussian random variable (as before)
- $w_d \sim N(0, \sigma^2)$ are i.i.d. Gaussian random variables (unlike before)
- We first choose the weight for each feature d, w_d, from $N(0, \sigma^2)$. Then for each input vector $\boldsymbol{x}_n$, draw y_n from the distribution $N(\boldsymbol{w}^\top \boldsymbol{x}_n, \sigma_0^2)$.

How do we estimate $\boldsymbol{w}$ for this model?

Maximum a posterior (MAP) estimate:

$$\begin{aligned} \boldsymbol{w}^{\text{MAP}} &= \arg\max_{\boldsymbol{w}} p(\boldsymbol{w}|\mathcal{D}) = \arg\max_{\boldsymbol{w}} \frac{p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w})}{p(\mathcal{D})} \\ &= \arg\max_{\boldsymbol{w}} p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w}) \end{aligned}$$

Estimating w

Let $\mathbf{x}_1, \ldots, \mathbf{x}_N$ be i.i.d. with $y|\boldsymbol{w}, \boldsymbol{x} \sim N(\boldsymbol{w}^\top \boldsymbol{x}, \sigma_0^2)$; $w_d \sim N(0, \sigma^2)$.

Given σ_0, σ, we choose $\boldsymbol{w}$ so as to maximize:

$$p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w}) = \prod_n p(y_n|\boldsymbol{x}_n, \boldsymbol{w}) \prod_d p(w_d)$$

Now we know $p(w_d) \propto \exp\left(\frac{-w_d^2}{2\sigma^2}\right)$ and $p(y_n|\boldsymbol{x}_n, \boldsymbol{w}) \propto \exp\left(\frac{-(\boldsymbol{w}^T \boldsymbol{x}_n - y_n)^2}{2\sigma_0^2}\right)$:

$$\begin{aligned}\log p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w}) &= \sum_n \log p(y_n|\boldsymbol{x}_n, \boldsymbol{w}) + \sum_d \log p(w_d) \\ &= -\frac{\sum_n (\boldsymbol{w}^\top \boldsymbol{x}_n - y_n)^2}{2\sigma_0^2} - \sum_d \frac{1}{2\sigma^2} w_d^2 + \text{const}\end{aligned}$$

MAP estimate: $\boldsymbol{w}^{\text{MAP}} = \arg\max_{\boldsymbol{w}} \log p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w})$

$$\boldsymbol{w}^{\text{MAP}} = \text{argmin}_{\boldsymbol{w}} \frac{\sum_n (\boldsymbol{w}^\top \boldsymbol{x}_n - y_n)^2}{2\sigma_0^2} + \frac{1}{2\sigma^2} \|\boldsymbol{w}\|_2^2$$

Maximum a posterior (MAP) estimate

$$\mathcal{E}(\boldsymbol{w}) = \sum_n (\boldsymbol{w}^\top \boldsymbol{x}_n - y_n)^2 + \lambda \|\boldsymbol{w}\|_2^2$$

where $\lambda > 0$ is used to denote σ_0^2/σ^2. This extra term $\|\boldsymbol{w}\|_2^2$ is called regularization/regularizer and controls the magnitude of $\boldsymbol{w}$.

- If $\lambda \to +\infty$, then $\sigma_0^2 \gg \sigma^2$: the variance of noise is far greater than what our prior model can allow for $\boldsymbol{w}$. In this case, our prior model on $\boldsymbol{w}$ will give a simpler model. Numerically,

$$\boldsymbol{w}^{\mathrm{MAP}} \to \mathbf{0}$$

- If $\lambda \to 0$, then we trust our data more. Numerically,

$$\boldsymbol{w}^{\mathrm{MAP}} \to \boldsymbol{w}^{\mathrm{LMS}} = \operatorname{argmin} \sum_n (\boldsymbol{w}^\top \boldsymbol{x}_n - y_n)^2$$

Hyperparameter Tuning and Cross-Validation

How should we choose the right amount of regularization?

Can we tune λ on the training dataset?

No: as this will always set λ to zero, i.e., no regularization, defeating our intention of controlling model complexity

λ is thus a hyperparameter. To tune it,

- We can use a validation set or do cross validation.
- Pick the value of λ that yields lowest error on the testing dataset.

Similar idea applies to tuning learning rate η (or any other hyperparameter) as well.

Tuning by using a validation dataset

Training data are used to learn $f(\cdot)$.

N samples/instances: $\mathcal{D}^{\text{TRAIN}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\text{N}}, y_{\text{N}})\}$

Test data are used to assess the prediction error.

- M samples/instances: $\mathcal{D}^{\text{TEST}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\text{M}}, y_{\text{M}})\}$
- They are used for assessing how well $f(\cdot)$ will do in predicting an unseen $\boldsymbol{x} \notin \mathcal{D}^{\text{TRAIN}}$

Validation data are used to optimize hyperparameter(s).

L samples/instances: $\mathcal{D}^{\text{VAL}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\text{L}}, y_{\text{L}})\}$

Training data, validation and test data should not overlap!

Recipe

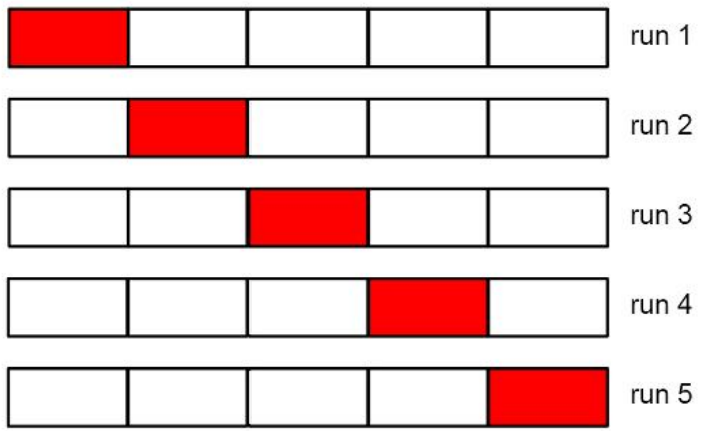
- For each possible value of the hyperparameter (say $\lambda = 1, 3, \cdots, 100$)
  - Train a model using $\mathcal{D}^{\text{TRAIN}}$
  - Evaluate the performance of the model on $\mathcal{D}^{\text{VAL}}$
- Choose the model with the best performance on $\mathcal{D}^{\text{VAL}}$
- Evaluate this model on $\mathcal{D}^{\text{TEST}}$ to get the final prediction error

Cross-validation

What if we do not have validation data?

- We split the training data into S equal parts.
- We use each part in turn as a validation dataset and use the others as a training dataset.
- We choose the hyperparameter such that the model performs the best (based on average, variance, etc.)

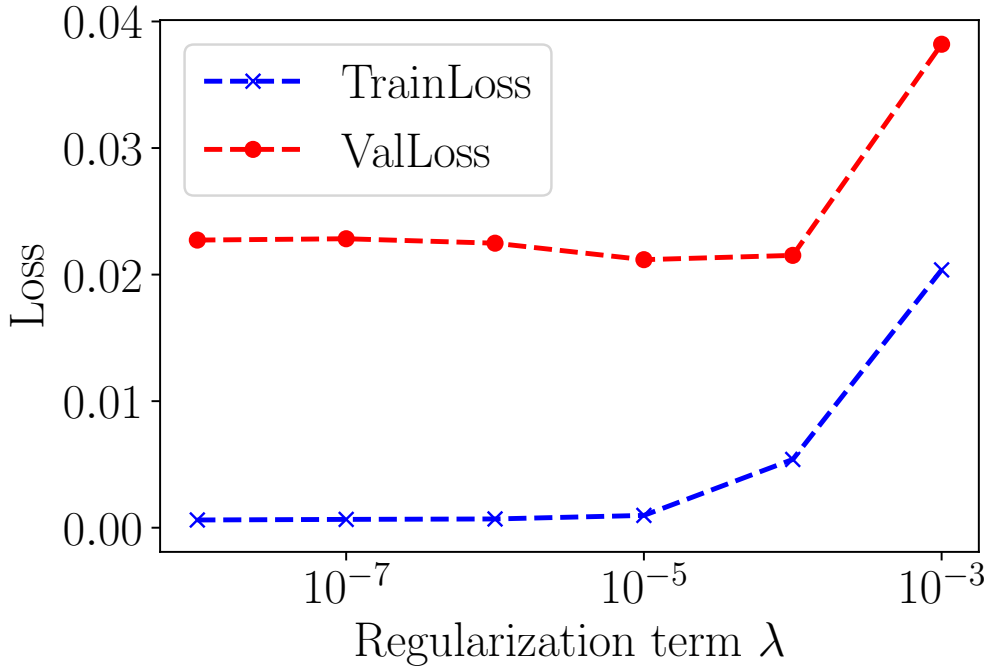
Figure 5: S = 5: 5-fold cross validation



Special case: when S = N, this will be leave-one-out.

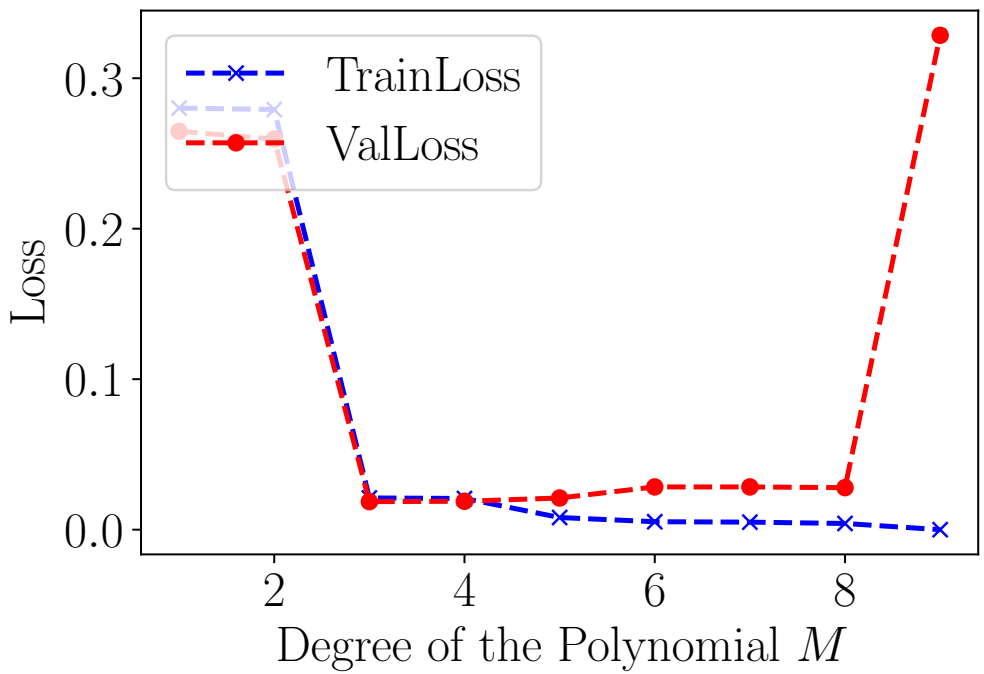
Example: Hyper-parameter Tuning λ

- $\lambda = 10^{-4}$ gives the smallest validation loss
- Strikes a balance between over- and under-fitting



Example: Hyper-parameter Tuning M

- Considering polynomial regression without regularization
- $M = 3$ or $M = 4$ gives the smallest validation loss
- Strikes a balance between over- and under-fitting



Bias-Variance Trade-off

Empirical Risk Minimization

Supervised learning

We aim to build a function $h(\boldsymbol{x})$ to predict the true value y associated with $\boldsymbol{x}$. If we make a mistake, we incur a loss

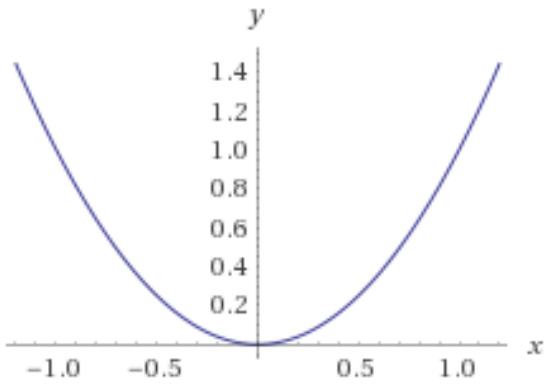
$$\ell(h(\boldsymbol{x}), y)$$

Example:

Quadratic loss function for regression when y is continuous:

$$\ell(h(\boldsymbol{x}), y) = [h(\boldsymbol{x}) - y]^2$$

Ex: when $y = 0$



How good is our predictor?

Risk:

Given the true distribution of data $p(\boldsymbol{x}, y)$, the risk of a given predictor $h(\boldsymbol{x})$ is its expected loss ℓ:

$$R[h(\boldsymbol{x})] = \int_{\boldsymbol{x},y} \ell(h(\boldsymbol{x}), y) p(\boldsymbol{x}, y) d\boldsymbol{x} d\, y$$

However, we cannot compute $R[h(\boldsymbol{x})]$ (we do not know p), so we use the empirical risk, given a training dataset $\mathcal{D}$:

$$R^{\mathrm{EMP}}[h(\boldsymbol{x})] = \frac{1}{N} \sum_{n} \ell(h(\boldsymbol{x}_n), y_n)$$

Intuitively, as $N \to +\infty$,

$$R^{\mathrm{EMP}}[h(\boldsymbol{x})] \to R[h(\boldsymbol{x})]$$

How could this go wrong?

So far, we have been doing empirical risk minimization (ERM)

For linear regression, $h(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$, and we use squared loss ℓ.

$$R^{\text{EMP}}[h(\boldsymbol{x})] = \frac{1}{N} \sum_n \ell(h(\boldsymbol{x}_n), y_n)$$

$$R[h(\boldsymbol{x})] = \int_{\boldsymbol{x},y} \ell(h(\boldsymbol{x}), y) p(\boldsymbol{x}, y) d\boldsymbol{x} dy$$

What could go wrong with ERM?

- Limited Function Class: The function $h(\boldsymbol{x})$ is restricted to a limited class (e.g. linear functions), which does not allow us to perfectly fit y, even if we had infinitely many training data points.
- Limited Data: We don't know $p(\boldsymbol{x}, y)$, so we must hope that we have enough training data that the empirical risk approximates the real risk. Otherwise, we will overfit to the training data.

Bias-Variance Trade-off: Intuition

- High Bias: Model is not rich enough to fit the training dataset and achieve low training loss
- High Variance: If the training dataset changes slightly, the model changes a lot
- Regularization helps find a middle ground

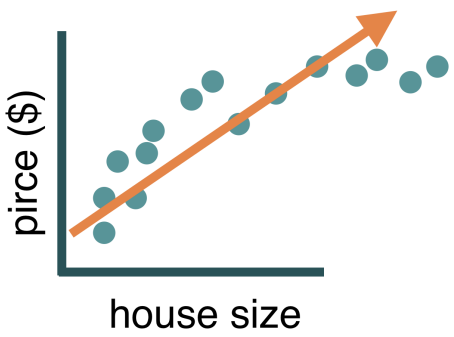


Figure 6: High Bias

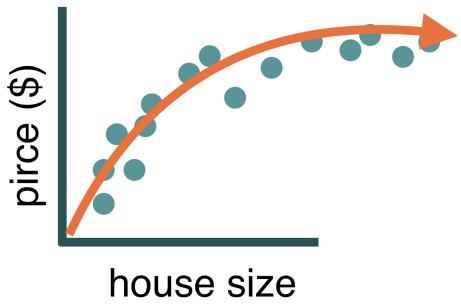


Figure 7: Just Right

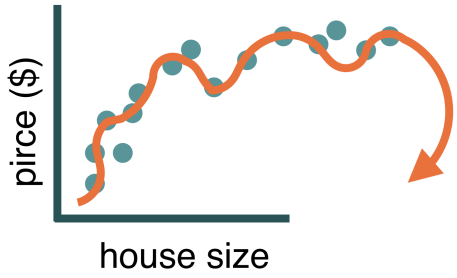


Figure 8: High Variance

Bias/variance tradeoff for regression

Goal: to understand the sources of prediction errors

- $\mathcal{D}$: our training data
- $h_{\mathcal{D}}(\boldsymbol{x})$: our prediction function
  We are using the subscript $\mathcal{D}$ to indicate that the prediction function is learned on the specific set of training data $\mathcal{D}$
- $\ell(h(\boldsymbol{x}), y)$: our square loss function for regression

$$\ell(h_{\mathcal{D}}(\boldsymbol{x}), y) = [h_{\mathcal{D}}(\boldsymbol{x}) - y]^2$$

- Unknown joint distribution $p(\boldsymbol{x}, y)$

The effect of finite training samples

Every training sample $\mathcal{D}$ is a sample from the following joint distribution of all possible training datasets

$$\mathcal{D} \sim P(\mathcal{D}) = \prod_{n=1}^{N} p(\boldsymbol{x}_n, y_n)$$

Thus, the prediction function $h_{\mathcal{D}}(\boldsymbol{x})$ is a random function with respect to this distribution of possible training datasets. So is also its risk

$$R[h_{\mathcal{D}}(\boldsymbol{x})] = \int_{\boldsymbol{x}} \int_{y} [h_{\mathcal{D}}(\boldsymbol{x}) - y]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy$$

We will now evaluate the expected risk $\mathbb{E}_{\mathcal{D}} R[h_{\mathcal{D}}(\boldsymbol{x})]$: the average risk over the distribution of possible training datasets, $P(\mathcal{D})$.

Bias-Variance Trade-off: Intuition

Error decomposes into 3 terms

$$\mathbb{E}_{\mathcal{D}} R[h_{\mathcal{D}}(\boldsymbol{x})] = \text{VARIANCE} + \text{BIAS}^2 + \text{NOISE}$$

We will prove this result, and interpret what it means...

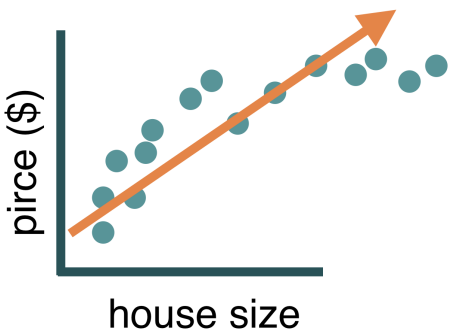


Figure 9: High Bias

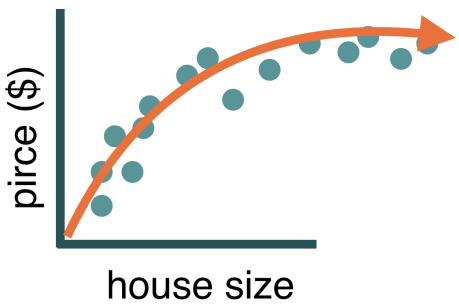


Figure 10: Just Right

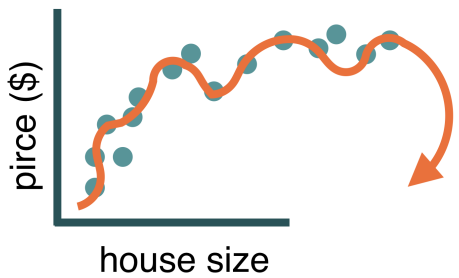


Figure 11: High Variance

Average over the distribution of the training data

Expected risk

$$\mathbb{E}_{\mathcal{D}}\left[R[h_{\mathcal{D}}(\boldsymbol{x})]\right] = \int_{\mathcal{D}} \int_{\boldsymbol{x}} \int_{y} [h_{\mathcal{D}}(\boldsymbol{x}) - y]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy \; P(\mathcal{D}) d\mathcal{D}$$

Namely, the randomness with respect to $\mathcal{D}$ is marginalized out.

Averaged prediction

$$\mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x}) = \int_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x}) P(\mathcal{D}) d\mathcal{D}$$

Namely, if we have seen many training datasets, we predict with the average of the prediction functions learned on each training dataset.

Variance

We will subtract the averaged prediction from the averaged risk

$$
\begin{aligned}
\mathbb{E}_{\mathcal{D}} R[h_{\mathcal{D}}(\boldsymbol{x})] &= \int_{\mathcal{D}} \int_{\boldsymbol{x}} \int_{y} [h_{\mathcal{D}}(\boldsymbol{x}) - y]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy \; P(\mathcal{D}) d\mathcal{D} \\
&= \int_{\mathcal{D}} \int_{\boldsymbol{x}} \int_{y} [h_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x}) \\
&\qquad + \mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x}) - y]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy \; P(\mathcal{D}) d\mathcal{D} \\
&= \underbrace{\int_{\mathcal{D}} \int_{\boldsymbol{x}} \int_{y} [h_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x})]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy \; P(\mathcal{D}) d\mathcal{D}}_{\text{VARIANCE}} \\
&\qquad + \int_{\mathcal{D}} \int_{\boldsymbol{x}} \int_{y} [\mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x}) - y]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy \; P(\mathcal{D}) d\mathcal{D}
\end{aligned}
$$

Where does the cross-term go?

It is zero

$$\int_{\mathcal{D}} \int_{\boldsymbol{x}} \int_{y} [h_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x})][\mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x}) - y] p(\boldsymbol{x}, y) d\boldsymbol{x} dy \; P(\mathcal{D}) d\mathcal{D}$$

$$= \int_{\boldsymbol{x}} \int_{y} \left\{ \int_{\mathcal{D}} [h_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x})] P(\mathcal{D}) d\mathcal{D} \right\} [\mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x}) - y] p(\boldsymbol{x}, y) d\boldsymbol{x} dy$$

$$= 0 \leftarrow \text{(the integral within the braces vanishes, by definition)}$$

Analyzing the variance

Understanding the variance

$$\int_{\mathcal{D}} \int_{\boldsymbol{x}} \int_{y} [h_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x})]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy \; P(\mathcal{D}) d\mathcal{D}$$

$$= \int_{\boldsymbol{x}} \int_{y} \left(\int_{\mathcal{D}} [h_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x})]^2 \; P(\mathcal{D}) d\mathcal{D} \right) p(\boldsymbol{x}, y) d\boldsymbol{x} dy$$

For each $(\boldsymbol{x}, y)$ pair, we compute the squared difference of $h_{\mathcal{D}}(\boldsymbol{x})$ (the prediction with training dataset $\mathcal{D}$) and the averaged prediction $\mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x})$: the average (over all $(\boldsymbol{x}, y) \sim p$) variance of the prediction over $\mathcal{D}$.

How can we reduce the variance?

- Use a lot of data (ie, increase the size of $\mathcal{D}$)
- Use a simple $h(\cdot)$ so that $h_{\mathcal{D}}(\boldsymbol{x})$ does not vary much across different training datasets. An extreme example is $h(\boldsymbol{x}) = \text{const}$.

The remaining item

$$\mathbb{E}_{\mathcal{D}} R[h_{\mathcal{D}}(\boldsymbol{x})] = \int_{\mathcal{D}} \int_{\boldsymbol{x}} \int_{y} [h_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x})]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy \; P(\mathcal{D}) d\mathcal{D}$$
$$+ \int_{\mathcal{D}} \int_{\boldsymbol{x}} \int_{y} [\mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x}) - y]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy \; P(\mathcal{D}) d\mathcal{D}$$

The integrand has no dependency on $\mathcal{D}$ anymore and simplifies to

$$\int_{\boldsymbol{x}} \int_{y} [\mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x}) - y]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy$$

We will apply a similar add-and-subtract trick, by using an averaged target y (what we want to predict from $\boldsymbol{x}$):

$$\mathbb{E}_y[y|\boldsymbol{x}] = \int_y y p(y|\boldsymbol{x}) dy$$

Bias and noise

Decompose again

$$\int_{\boldsymbol{x}} \int_{y} [\mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x}) - y]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy$$

$$= \int_{\boldsymbol{x}} \int_{y} [\mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_y[y|\boldsymbol{x}] + \mathbb{E}_y[y|\boldsymbol{x}] - y]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy$$

$$= \underbrace{\int_{\boldsymbol{x}} \int_{y} [\mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_y[y|\boldsymbol{x}]]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy}_{\text{BIAS}^2}$$

$$+ \underbrace{\int_{\boldsymbol{x}} \int_{y} [\mathbb{E}_y[y|\boldsymbol{x}] - y]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy}_{\text{NOISE}}$$

Where is the cross-term?

Take-home exercise: Show that it is zero

Analyzing the noise

How can we reduce noise?

$$\int_{\boldsymbol{x}} \int_{y} [\mathbb{E}_y[y|\boldsymbol{x}] - y]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy = \int_{\boldsymbol{x}} \left(\int_{y} [\mathbb{E}_y[y|\boldsymbol{x}] - y]^2 p(y|\boldsymbol{x}) dy \right) p(\boldsymbol{x}) d\boldsymbol{x}$$

There is nothing we can do. This quantity depends on $p(\boldsymbol{x}, y)$ only; choosing $h(\cdot)$ or the training dataset $\mathcal{D}$ will not affect it. Note that the integral inside the parentheses is the *variance* (noise) of the posterior distribution $p(y|\boldsymbol{x})$ at the given $\boldsymbol{x}$.

Figure 12: Somewhat difficult posterior

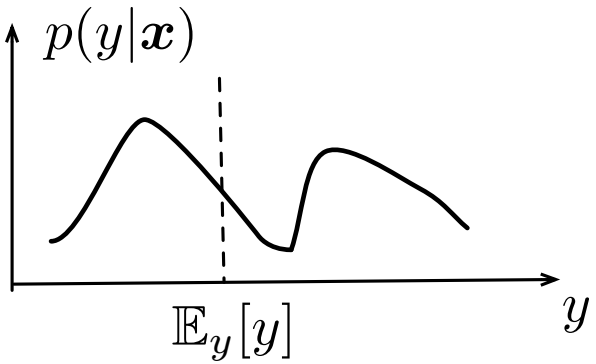
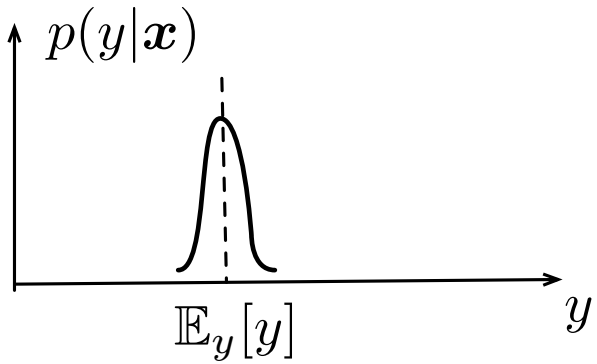


Figure 13: Somewhat easy posterior



Analyzing the bias term

Understanding the bias

$$\int_{\boldsymbol{x}} \int_{y} [\mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_{y}[y|\boldsymbol{x}]]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy$$

For each $(\boldsymbol{x}, y)$ pair, we compute the loss of our averaged prediction $\mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x})$ compared to the expected value of y given $\boldsymbol{x}$, which we compute as $\mathbb{E}_{y}[y|\boldsymbol{x}] = \int_{y} y p(y|\boldsymbol{x}) dy$. Then we take the average over all pairs $(\boldsymbol{x}, y) \sim p(\boldsymbol{x}, y)$.

How can we reduce the bias?

It can be reduced by using more complex models. We shall choose $h(\cdot)$ to be as flexible as possible: the better $h(\cdot)$ approximates $\mathbb{E}_{y}[y|\boldsymbol{x}]$, the smaller the bias. However, this will increase the VARIANCE term.

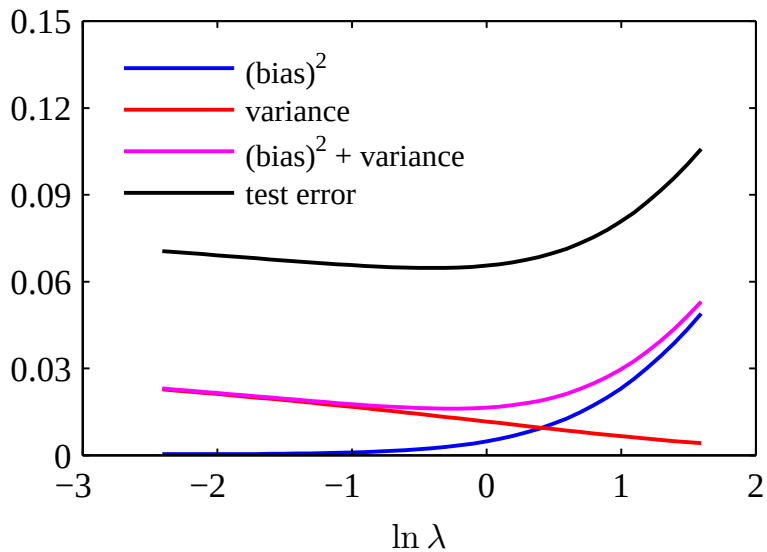
Bias/variance tradeoff

Error decomposes into 3 terms

$$\mathbb{E}_{\mathcal{D}} R[h_{\mathcal{D}}(\boldsymbol{x})] = \text{VARIANCE} + \text{BIAS}^2 + \text{NOISE}$$

where the first and the second term are inherently in conflict in terms of choosing what kind of $h(\boldsymbol{x})$ we should use (unless we have an infinite amount of data).

If we can compute all terms analytically, they will look like this



Summary of risk components

The average risk (with quadratic loss) can be decomposed as:

$$\mathbb{E}_{\mathcal{D}} R[h_{\mathcal{D}}(\boldsymbol{x})] = \underbrace{\int_{\mathcal{D}} \int_{\boldsymbol{x}} \int_{y} [h_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x})]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy \; P(\mathcal{D}) d\mathcal{D}}_{\text{VARIANCE: error due to training dataset}}$$

$$+ \underbrace{\int_{\boldsymbol{x}} \int_{y} [\mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_{y}[y|\boldsymbol{x}]]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy}_{\text{BIAS}^2\text{: error due to the model approximation}}$$

$$+ \underbrace{\int_{\boldsymbol{x}} \int_{y} [\mathbb{E}_{y}[y|\boldsymbol{x}] - y]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy}_{\text{NOISE: error due to randomness of } y}$$

Here we define: $h_{\mathcal{D}}(\boldsymbol{x})$ as the output of the model trained on $\mathcal{D}$, $\mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x})$ as the expectation of the model over all datasets $\mathcal{D}$, and $\mathbb{E}_{y}[y|\boldsymbol{x}]$ as the expected value of y.

Example: Why regularized linear regression could be helpful?

Model

$$h(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$$

Consider the best possible (linear) $h^*(\boldsymbol{x})$

$$\boldsymbol{w}^* = \text{argmin}_{\boldsymbol{w}} \int_{\boldsymbol{x}} [\mathbb{E}_y[y|\boldsymbol{x}] - \boldsymbol{w}^\top \boldsymbol{x}]^2 p(\boldsymbol{x}) d\boldsymbol{x}$$

Note that this linear model assumes the knowledge of joint distribution, thus, not achievable. Intuitively, it is the *best* linear model that can predict the data most accurately.

More refined decomposition of the bias

$$\int_{\boldsymbol{x}} [\mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_y[y|\boldsymbol{x}]]^2 p(\boldsymbol{x}) d\boldsymbol{x} = \int_{\boldsymbol{x}} [h^*(\boldsymbol{x}) - \mathbb{E}_y[y|\boldsymbol{x}]]^2 p(\boldsymbol{x}) d\boldsymbol{x} + \int_{\boldsymbol{x}} [\mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x}) - h^*(\boldsymbol{x})]^2 p(\boldsymbol{x}) d\boldsymbol{x}$$

- Model bias: the price we pay for choosing linear functions to model data. This is the difference between the prediction of the best possible linear model and the actual target.
- Estimation bias: the difference between the optimal model and the estimated model.

Normally, the estimation bias is zero if we do not regularize.

Bias/variance tradeoff for regularized linear regression

We can only adjust estimation bias

$$\int_{\boldsymbol{x}} [\mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x}; \lambda) - h^*(\boldsymbol{x})]^2 p(\boldsymbol{x}) d\boldsymbol{x}$$

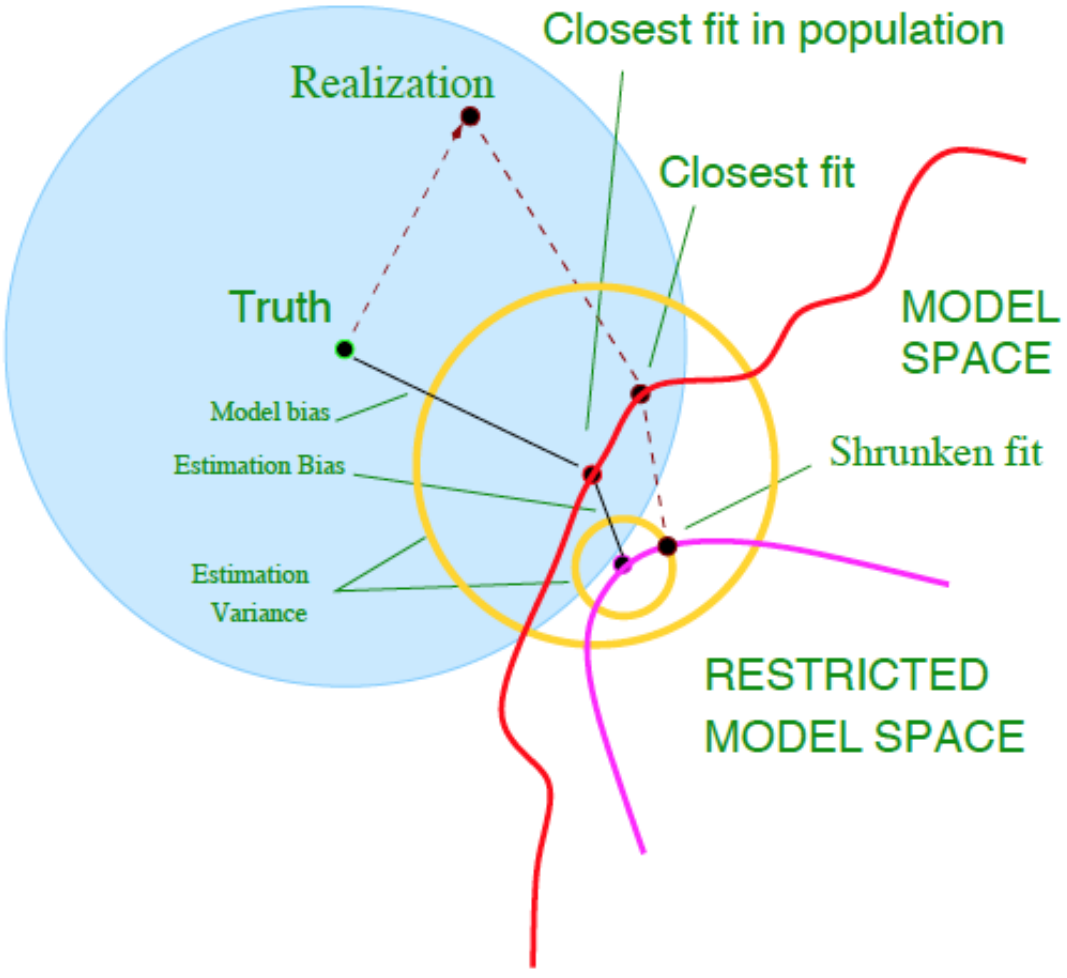
where $h(\boldsymbol{x}; \lambda)$ is the estimated model with regularized linear regression (parameterized with λ).

This term will not be zero anymore!

Thus, bias goes up.

But, as long as this is balanced with a decrease in variance, we are willing to do so.

Visualizing the tradeoff



Lecture Summary

- Validation datasets (or cross-validation) are used to determine model hyperparameters.
- Many ML models use empirical risk minimization to find the optimal parameters.
- ERM leads to an error consisting of bias, variance, and noise terms.
  - Variance: Due to only optimizing over an empirical sample of the complete $(\boldsymbol{x}, y)$ distribution.
  - Bias: Due to our choosing a model that does not fit the exact $(\boldsymbol{x}, y)$ relationship.
  - Noise: Due to the output y's randomness with respect to the input $\boldsymbol{x}$.
- Choosing a more complex model improves the bias, but increases the variance (and vice versa for less complex models).
- The noise is independent of the model that we choose.