Computer Vision with Deep Learning

Machine Learning Basics

Fatma Güney

COMP411/511 - Fall 2024

PLAN

Supervised Learning

Linear Regression

Ridge Regression

Estimators, Bias & Variance

***Prompt**: A student learning machine learning basics*

Learning Paradigms

- **Supervised Learning**
  - Learn model parameters using dataset of data-label pairs $\{(x_i, y_i)\}_{i=1}^{N}$
  - *Examples*: Classification, regression, structured prediction
- **Unsupervised Learning**
  - Learn model parameters using dataset without labels $\{(x_i)\}_{i=1}^{N}$
  - *Examples*: Clustering, dimensionality reduction, generative models
- **Self-Supervised Learning**
  - Learn model parameters using dataset of data-data pairs $\{(x_i, x_i')\}_{i=1}^{N}$
  - *Examples*: Self-supervised stereo/flow, contrastive learning
- **Reinforcement Learning**
  - Learn model parameters using active exploration from sparse rewards
  - *Examples*: Deep Q-learning, gradient policy, actor critique

Supervised Learning

Classification, Regression, Structured Prediction

Classification / Regression:

$$f : \boxed{\mathcal{X}} \mapsto \boxed{\mathcal{N}} \text{ or } f : \boxed{\mathcal{X}} \mapsto \boxed{\mathcal{R}}$$

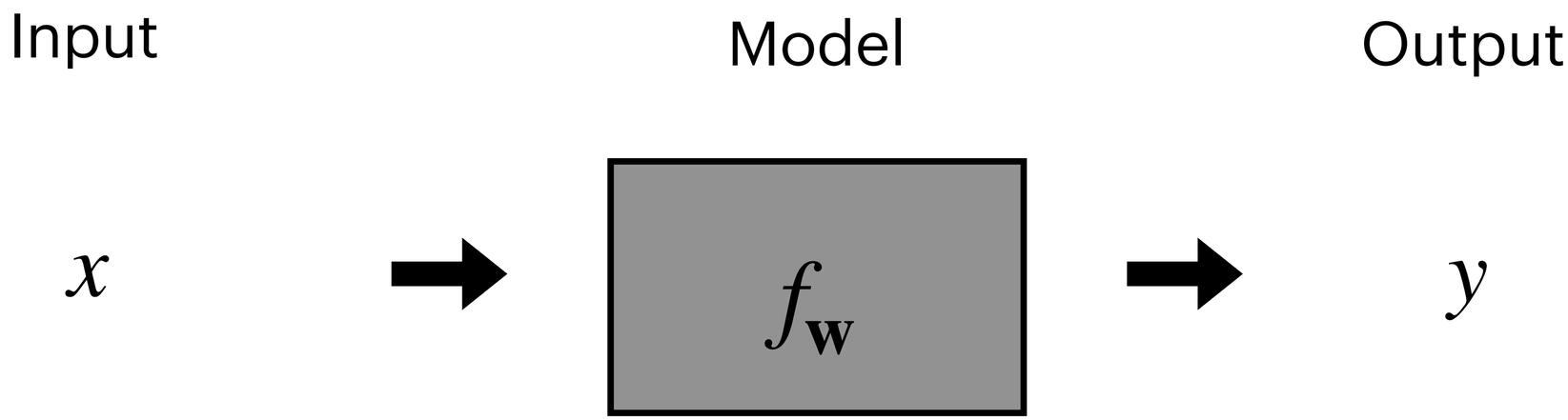
- Input $x \in \mathcal{X}$ can be any kind of object
  - Image, text, audio, sequence of amino acids, ...
- Output $y \in \mathcal{N}$ or $y \in \mathcal{R}$ is a discrete or real number
  - Classification, regression, density estimation, ...

Structured Output Learning:

$$f : \boxed{\mathcal{X}} \mapsto \boxed{\mathcal{Y}}$$

- Input $x \in \mathcal{X}$ can be any kind of object
- Output is a complex structured object
  - Image, text, parse tree, fold of a protein, computer program ...

Supervised Learning



- **Learning:** Estimate parameters $\mathbf{w}$ from training data $\{(x_i, y_i)\}_{i=1}^{N}$
- **Inference:** Make novel predictions $y = f_{\mathbf{w}}(x)$

Classification

Input

Model

$f_{\mathbf{w}}$

Output

“Beach”

- **Mapping:** $f_{\mathbf{w}} : \mathscr{R}^{W \times H} \mapsto \{\text{"Beach", "No beach"}\}$

Regression

Input

Model

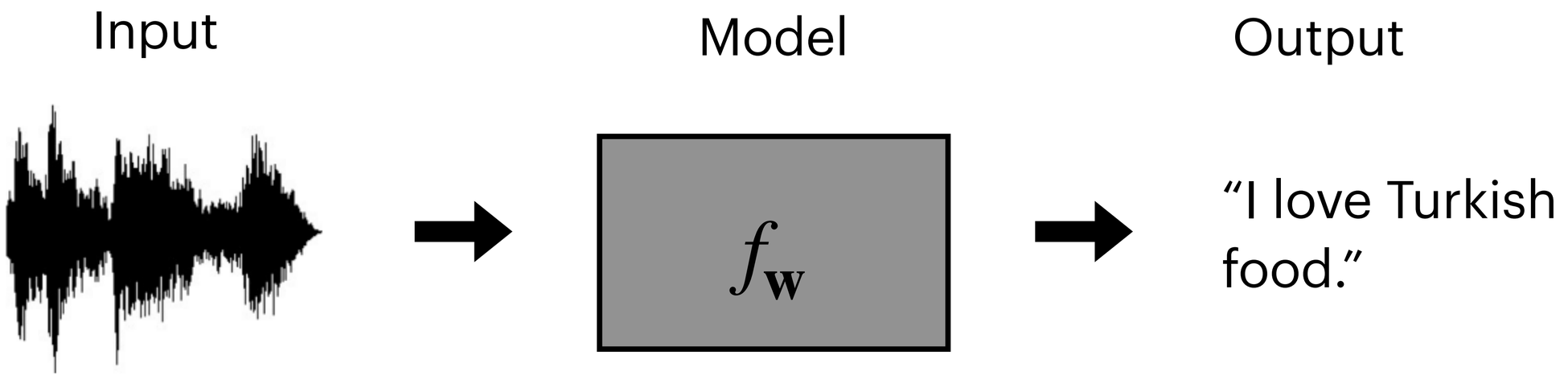
$f_{\mathbf{w}}$

Output

143,52 €

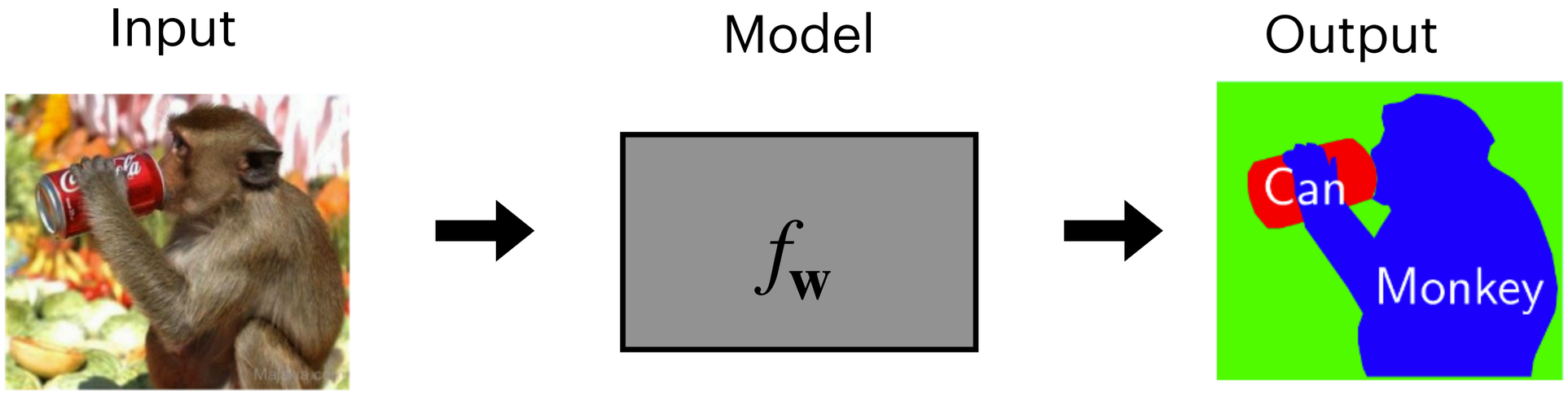
- **Mapping:** $f_{\mathbf{w}} : \mathcal{R}^N \mapsto \mathcal{R}$

Structured Prediction



- **Mapping:** $f_{\mathbf{w}} : \mathscr{R}^N \mapsto \{1,\ldots,C\}^M$

Structured Prediction



- **Mapping:** $f_{\mathbf{w}} : \mathscr{R}^{W \times H} \mapsto \{1, \ldots, C\}^{W \times H}$

Structured Prediction

Input

Model

$f_{\mathbf{w}}$

Output

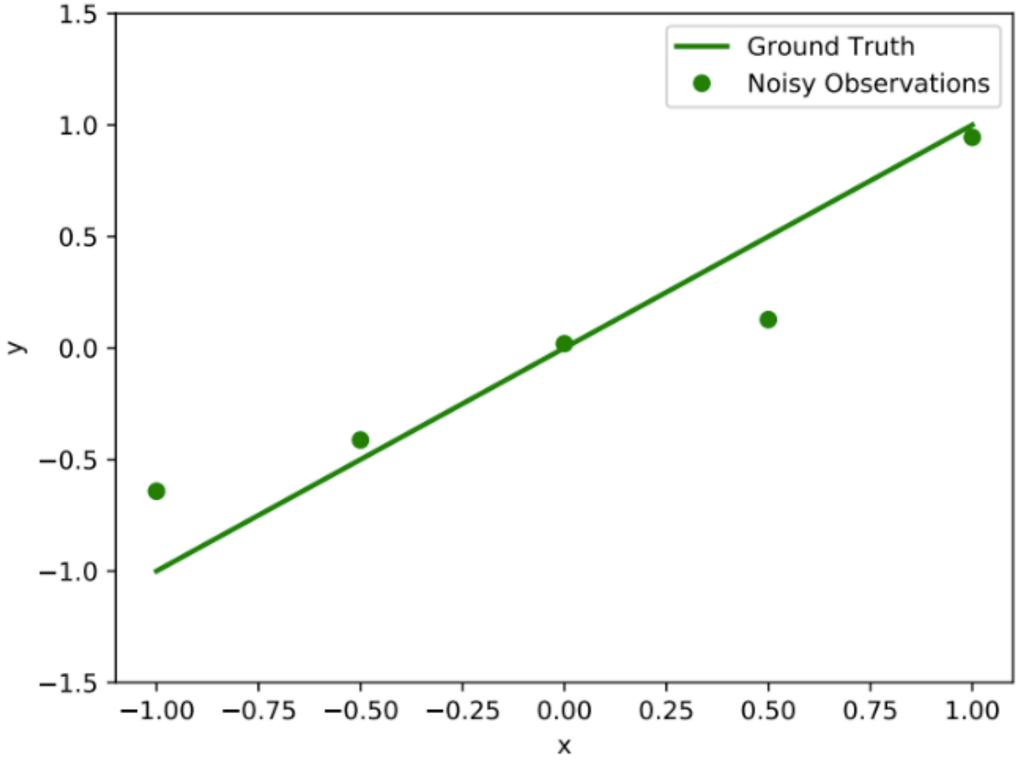
- **Mapping:** $f_{\mathbf{w}} : \mathscr{R}^{W \times H \times N} \mapsto \{0,1\}^{M^3}$
- **Suppose:** 32^3 voxels, binary variable per voxel (occupied/free)
- **Question**: How many different reconstruction? $2^{32^3} = 2^{32768}$
- **Comparison**: Number of atoms in the universe? $\approx 2^{273}$

Linear Regression

Linear Regression

Let $\mathcal{X}$ denote a dataset of size N and let $(\mathbf{x}_i, y_i) \in \mathcal{X}$ denote its elements ($y_i \in \mathcal{R}$).

Goal: Predict y for a previously unseen input $\mathbf{x}$. The input $\mathbf{x}$ may be multidimensional.



Linear Regression

The **error function** $E(\mathbf{w})$ measures the displacement along the y dimension between the data points (green) and the model $f(\mathbf{x}, \mathbf{w})$ (red) specified by the parameters $\mathbf{w}$.

$$
\begin{aligned}
f(\mathbf{x}, \mathbf{w}) &= \mathbf{w}^T \mathbf{x} \\
E(\mathbf{w}) &= \sum_{i=1}^{N} (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2 \\
&= \sum_{i=1}^{N} (\mathbf{x}_i^T \mathbf{w} - y_i)^2 \\
&= \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2
\end{aligned}
$$

Linear Regression

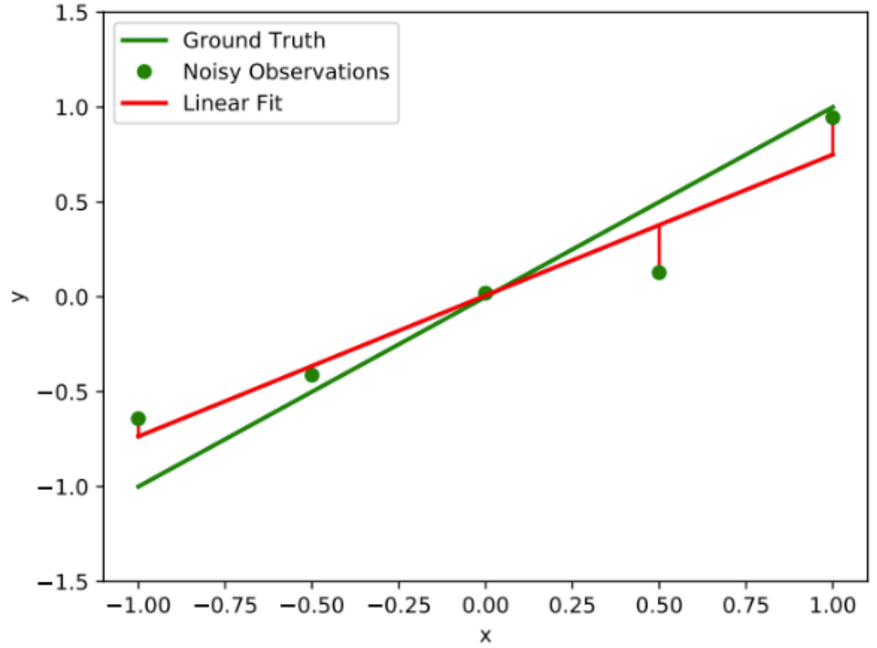
The **error function** $E(\mathbf{w})$ measures the displacement along the y dimension between the data points (green) and the model $f(\mathbf{x}, \mathbf{w})$ (red) specified by the parameters $\mathbf{w}$.

$$f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x}$$

$$E(\mathbf{w}) = \sum_{i=1}^{N} (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$$

$$= \sum_{i=1}^{N} (\mathbf{x}_i^T \mathbf{w} - y_i)^2$$

$$= \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$



Here: $\mathbf{x} = [1, x]^T \Rightarrow f(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x$

Linear Regression

The **gradient of the error function** with respect to the parameters $\mathbf{w}$ is given by:

$$\begin{aligned}\nabla_{\mathbf{w}} E(\mathbf{w}) &= \nabla_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 \\ &= \nabla_{\mathbf{w}} (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y}) \\ &= \nabla_{\mathbf{w}} (\mathbf{w}^T \mathbf{X}^T \mathbf{X}\mathbf{w} - 2\mathbf{w}^T \mathbf{X}^T \mathbf{y} + \mathbf{y}^T \mathbf{y}) \\ &= 2\mathbf{X}^T \mathbf{X}\mathbf{w} - 2\mathbf{X}^T \mathbf{y}\end{aligned}$$

As $E(\mathbf{w})$ is quadratic and convex in $\mathbf{w}$, its minimizer (w.r.t. $\mathbf{w}$) is given in closed form:

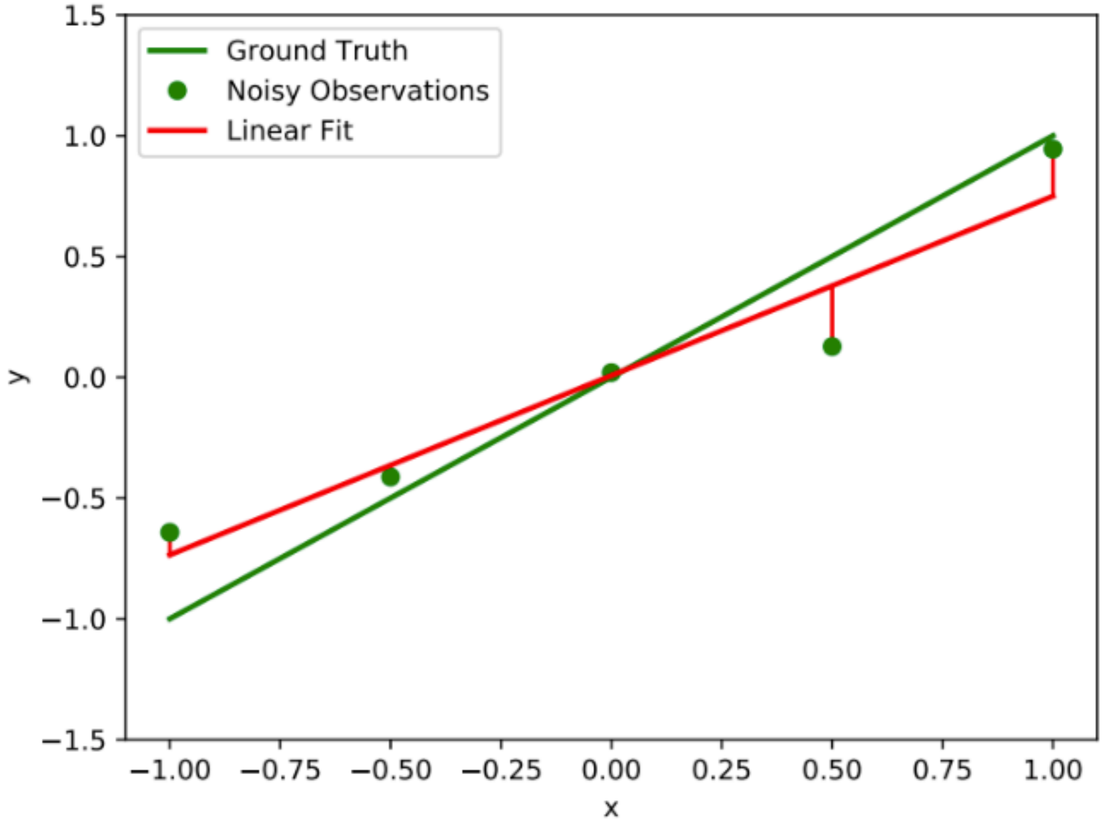
$$\nabla_{\mathbf{w}} E(\mathbf{w}) = 0$$

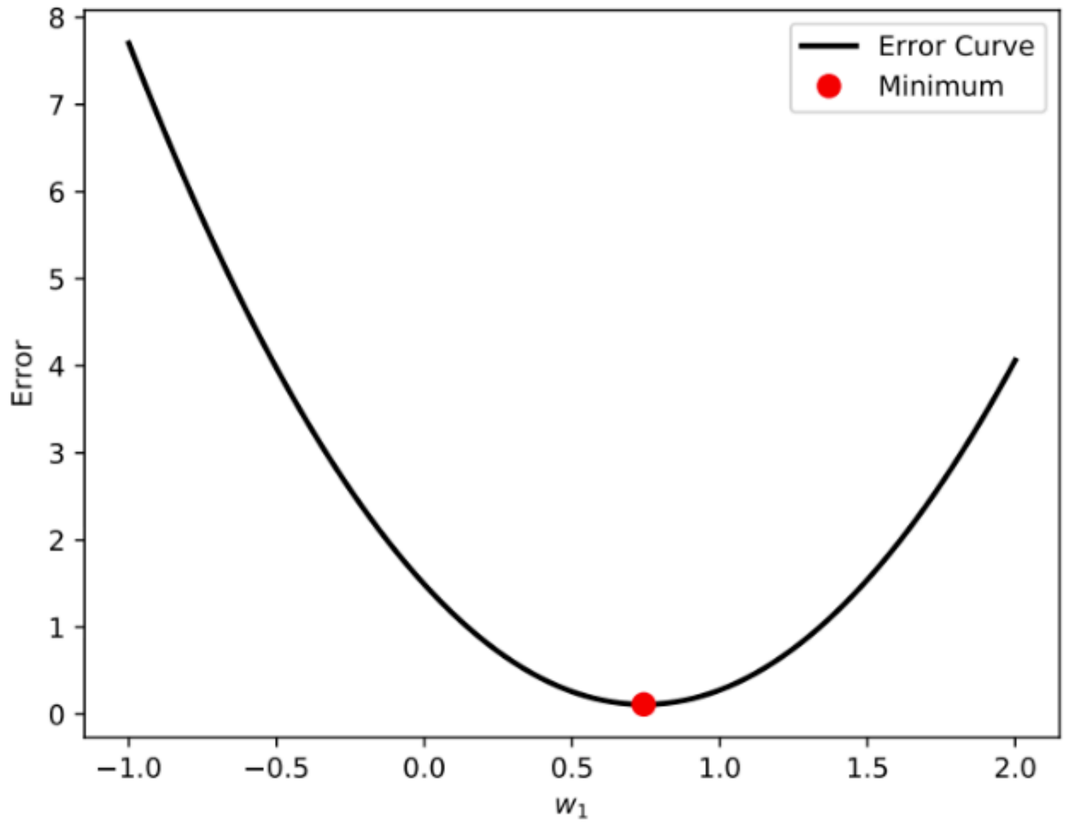
$$\Rightarrow \mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

The matrix $(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T$ is also called **Moore-Penrose inverse** or pseudoinverse.

Example: Line Fitting

Line Fitting





Linear least squares fit of model $f(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x$ (red) to data points (green). Errors are also shown in red. Right: Error function $E(\mathbf{w})$ w.r.t. parameter w_1.

Example: Polynomial Curve Fitting

Polynomial Curve Fitting

Let us choose **a polynomial of order** M to model dataset $\mathcal{X}$:

$$f(x, \mathbf{w}) = \sum_{j=0}^{M} w_j x^j = \mathbf{w}^T \mathbf{x} \qquad \text{with features} \qquad \mathbf{x} = (1, x^1, x^2, \ldots, x^M)^T$$

Tasks:

- **Training**: Estimate $\mathbf{w}$ from dataset $\mathcal{X}$
- **Inference**: Predict y for novel x given estimated $\mathbf{w}$

Note:

- Features can be anything, including multi-dimensional inputs (e.g., images, audio), radial basis functions, sine/cosine functions, etc. In this example: monomials.

Polynomial Curve Fitting

Let us choose **a polynomial of order** M to model dataset $\mathcal{X}$:

$$f(x, \mathbf{w}) = \sum_{j=0}^{M} w_j x^j = \mathbf{w}^T \mathbf{x} \qquad \text{with features} \qquad \mathbf{x} = (1, x^1, x^2, \ldots, x^M)^T$$

How can we estimate $\mathbf{w}$ from $\mathcal{X}$?

- Define **an error function**

$$E(\mathbf{w}) = \sum_{i=1}^{N} \left(f(x_i, \mathbf{w}) - y_i \right)^2$$

- **Goal**: Optimize this error function w.r.t. parameters $\mathbf{w}$

Polynomial Curve Fitting

The **error function** from above is quadratic in $\mathbf{w}$ but not in x:

$$E(\mathbf{w}) = \sum_{i=1}^{N} \left(f(x_i, \mathbf{w}) - y_i \right)^2 = \sum_{i=1}^{N} \left(\mathbf{w}^T \mathbf{x}_i - y_i \right)^2 = \sum_{i=1}^{N} \left(\sum_{j=0}^{M} w_j x_i^j - y_i \right)^2$$

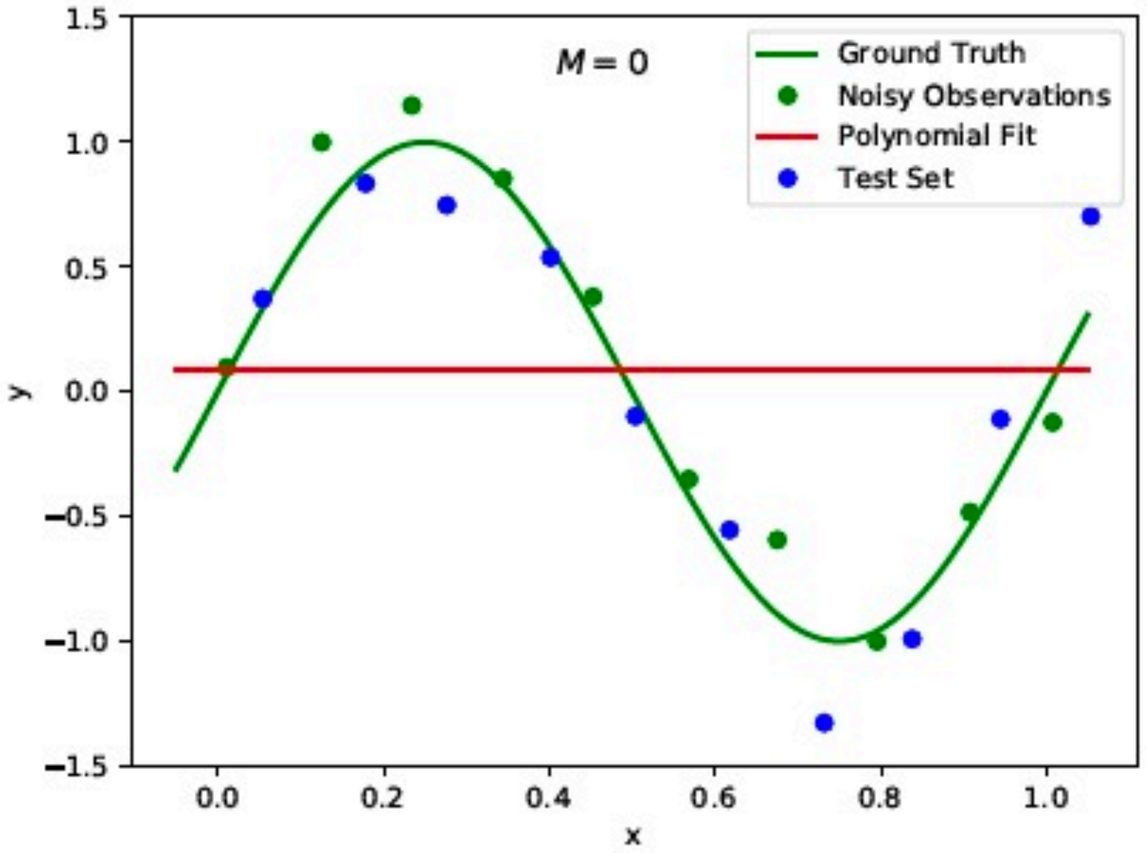
It can be rewritten in the **matrix-vector notation** (i.e., as linear regression problem):

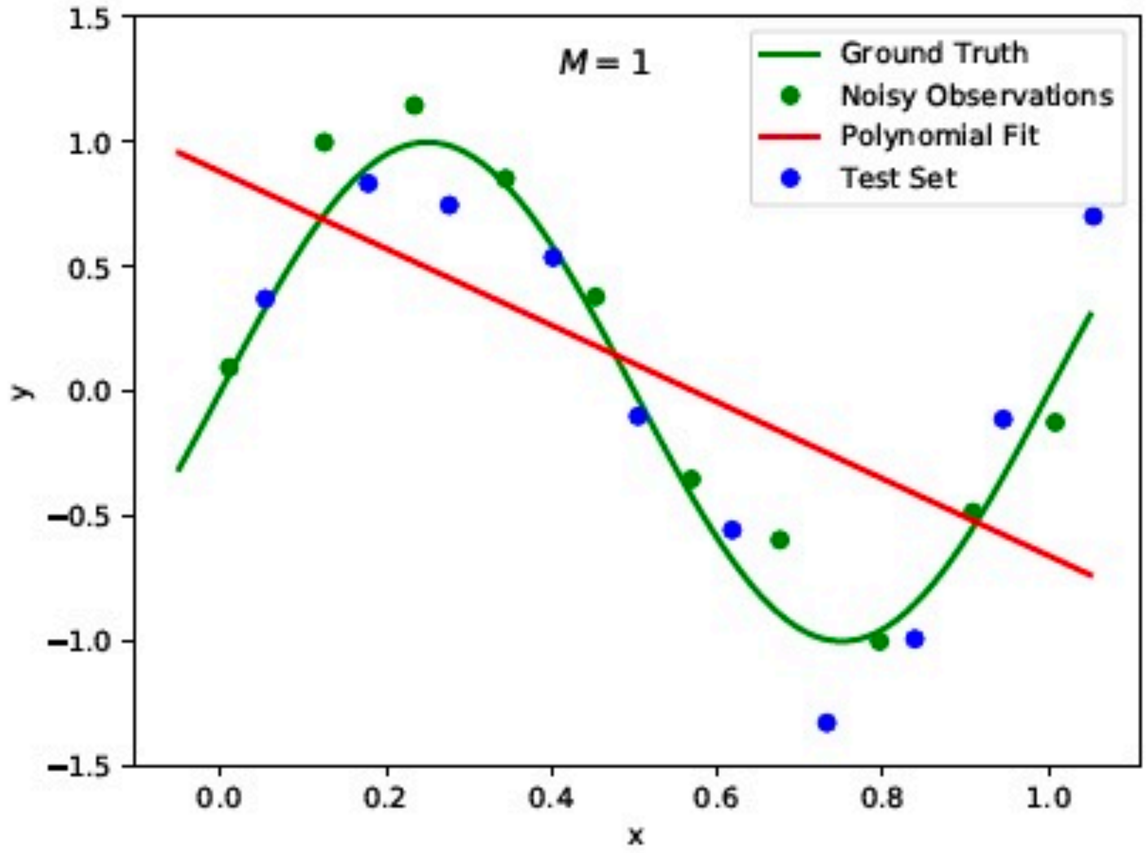
$$E(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

with feature matrix $\mathbf{X}$, observation vector $\mathbf{y}$ and weight vector $\mathbf{w}$:

$$\mathbf{X} = \begin{pmatrix} \vdots & \vdots & \vdots & & \vdots \\ 1 & x_i & x_i^2 & \dots & x_i^M \\ \vdots & \vdots & \vdots & & \vdots \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} \vdots \\ y_i \\ \vdots \end{pmatrix} \qquad \mathbf{w} = \begin{pmatrix} w_0 \\ \vdots \\ w_M \end{pmatrix}$$

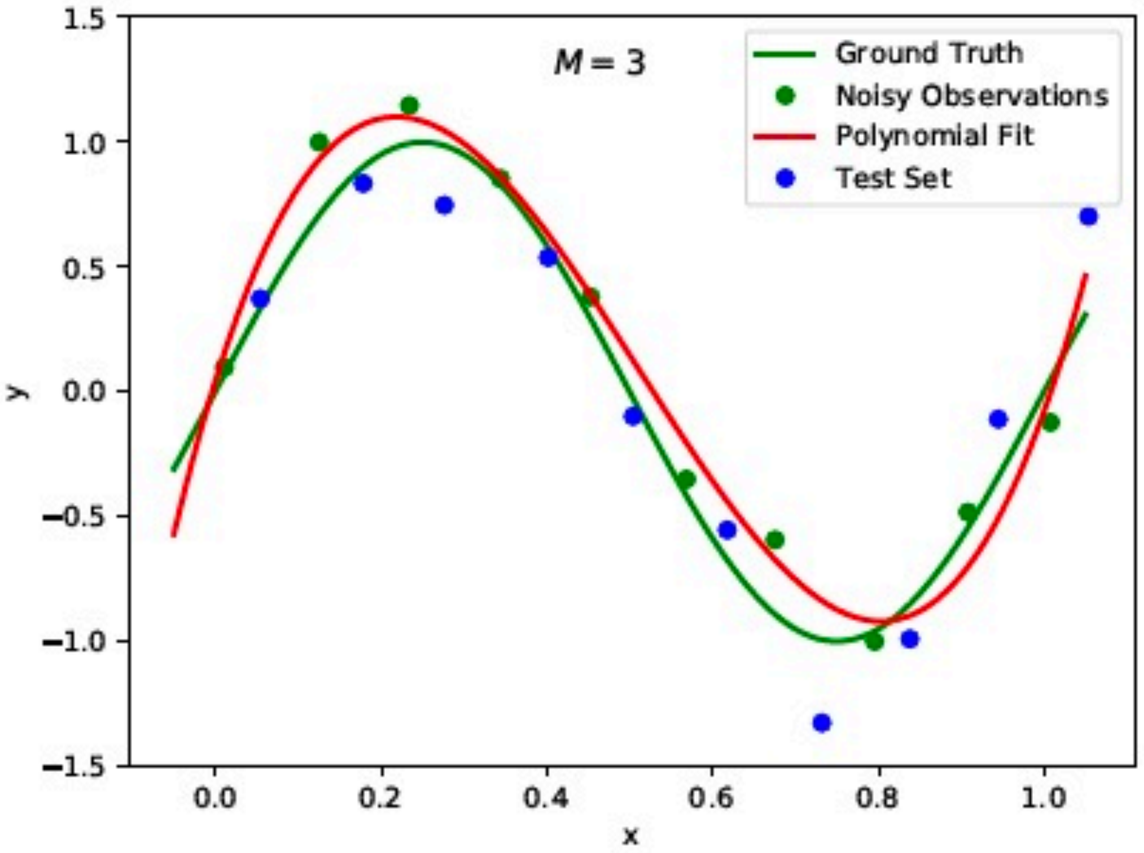
Polynomial Curve Fitting

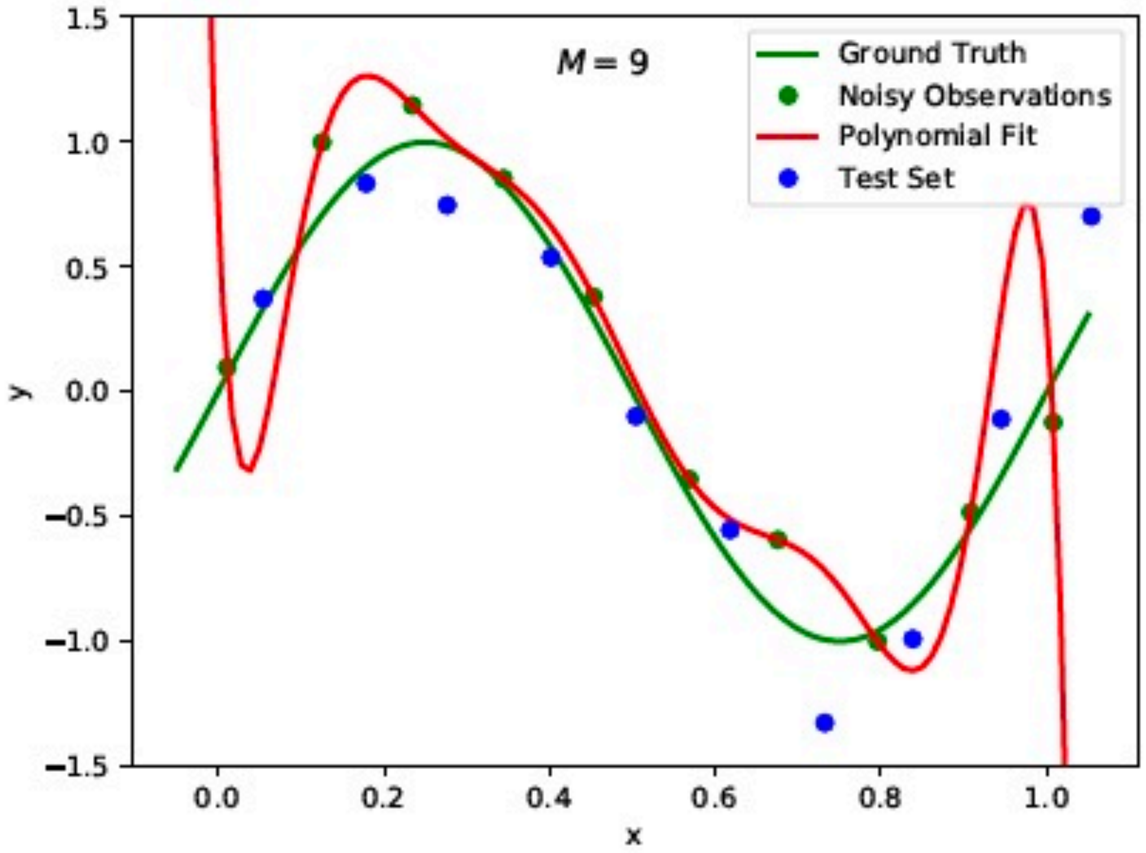




Plots of polynomials of various degrees M (red) fitted to the data (green). We observe underfitting ($M = 0/1$) and overfitting ($M = 9$). This is a model selection problem.

Polynomial Curve Fitting





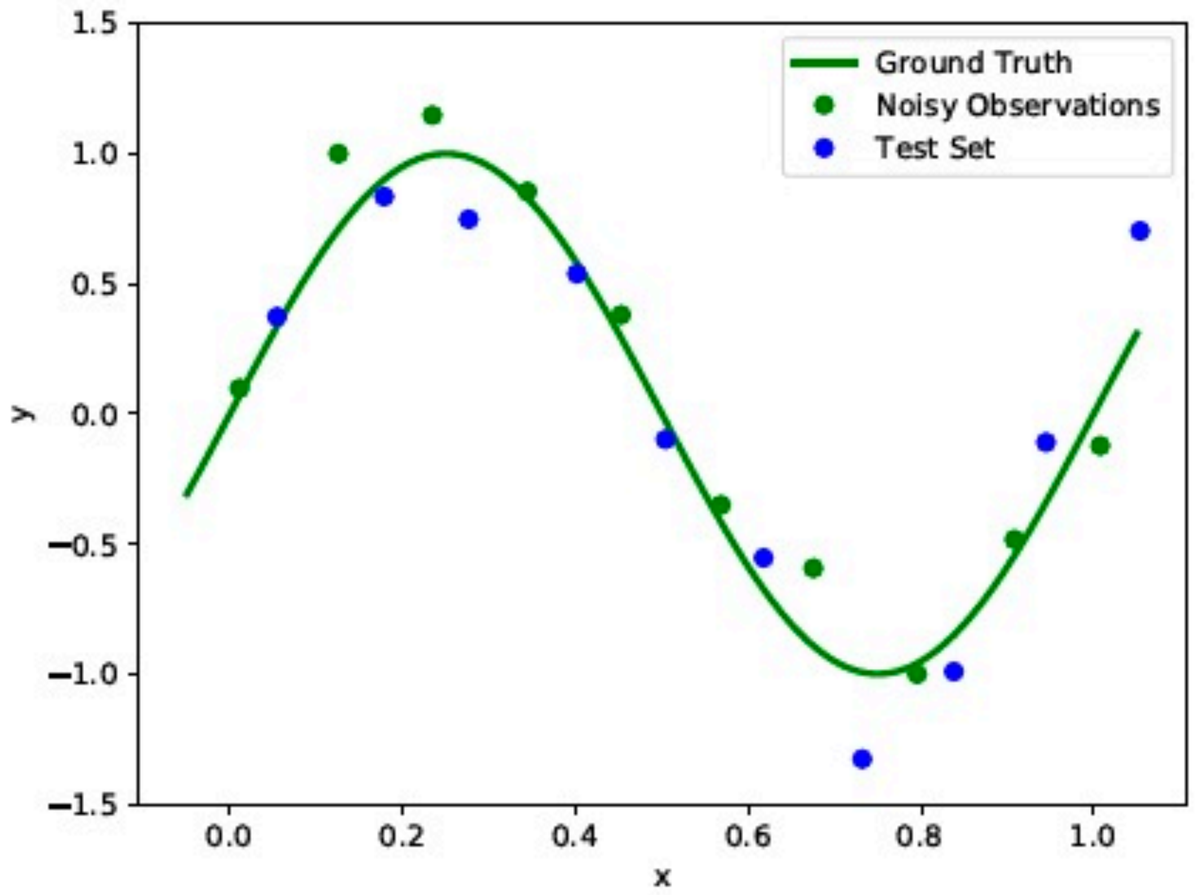
Plots of polynomials of various degrees M (red) fitted to the data (green). We obtain better fits ($M = 3$) and overfitting ($M = 9$). This is a model selection problem.

Capacity, Overfitting, and Underfitting

Goal:

- Perform well on new, previously unseen inputs (test set, blue), not only on the training set (green)
- This is called **generalization**
- Assumption: training and test data independent and identically (i.i.d.) drawn from distribution $p_{data}(x, y)$
- Here: $p_{data}(x) = \mathcal{U}(0,1)$

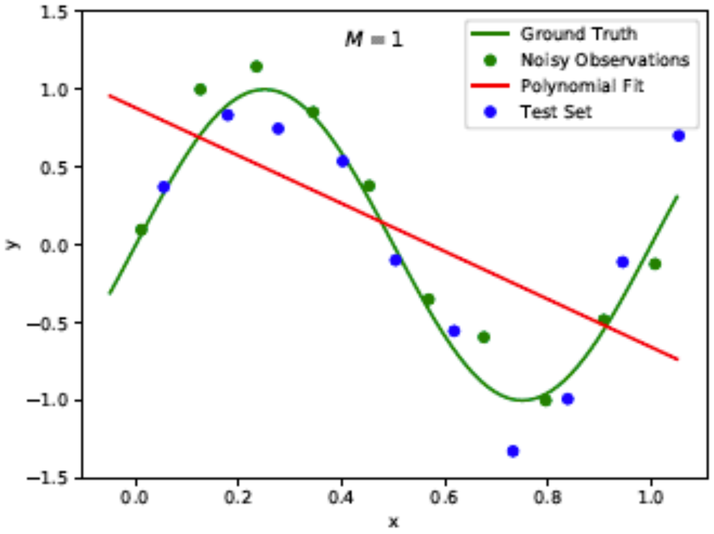
  $p_{data}(y \mid x) = \mathcal{N}\left(\sin(2\pi x), \sigma\right)$



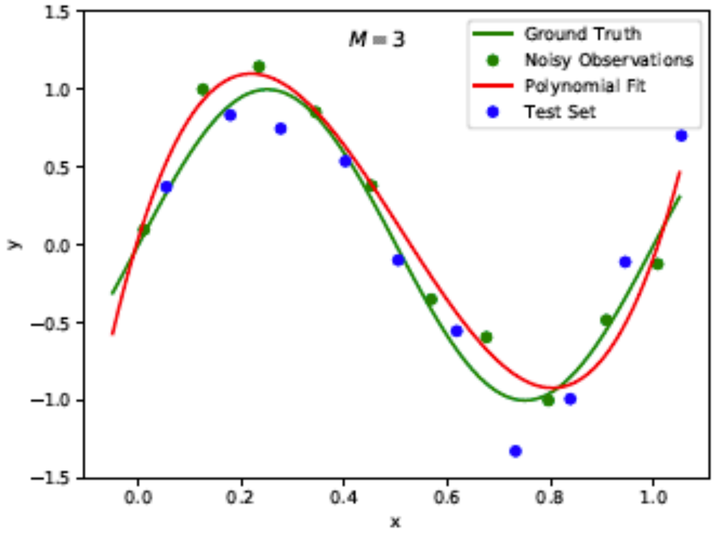
Capacity, Overfitting, and Underfitting

Terminology:

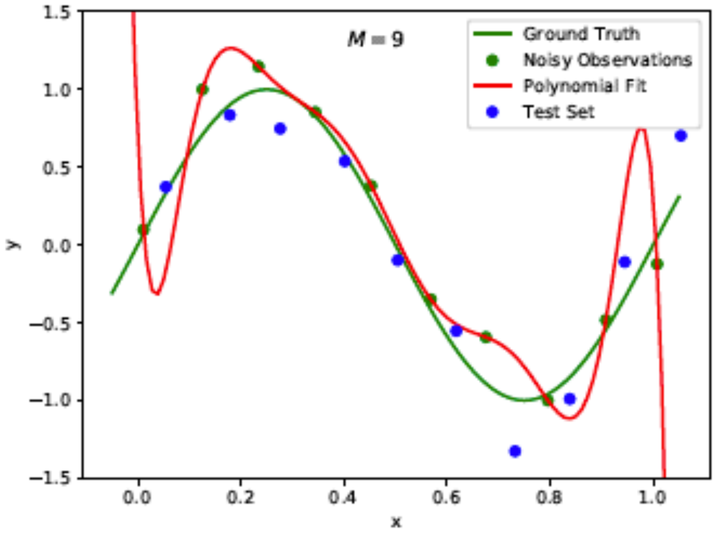
- **Capacity**: Complexity of functions which can be represented by model f
- **Underfitting**: Model too simple, does not achieve low error on training set
- **Overfitting**: Training error small, but test error (= generalization error) large



Capacity too low



Capacity about right

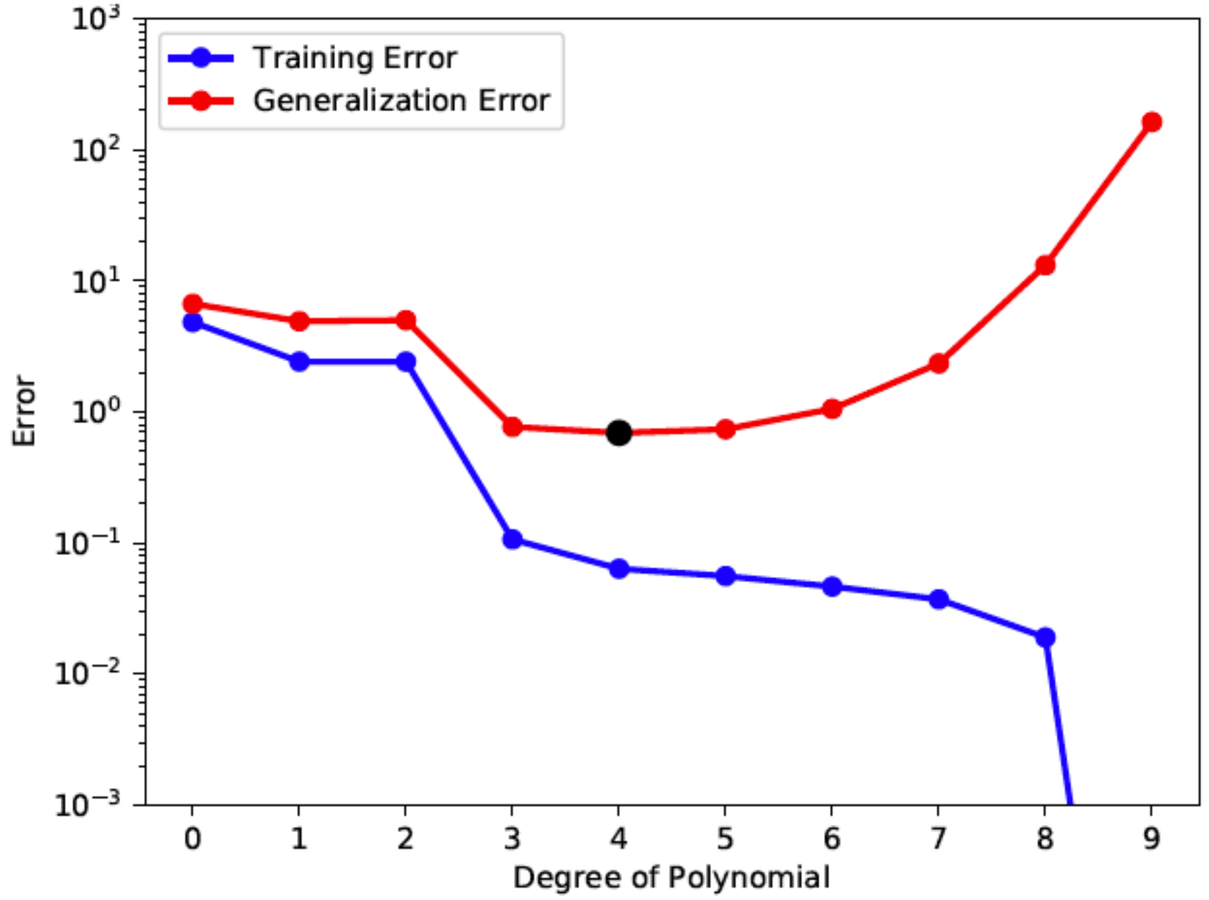


Capacity too large

Capacity, Overfitting, and Underfitting

Example: Generalization error for various polynomial degrees M

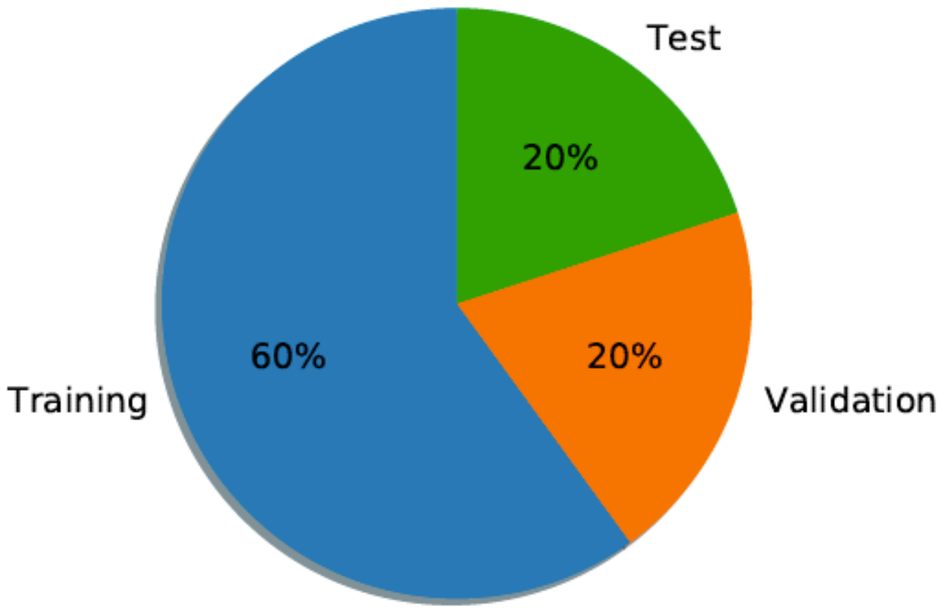
- ✦ Model selection: Select model with the smallest generalization error



Capacity, Overfitting, and Underfitting

General Approach: Split dataset into training, validation and test set

- Choose hyperparameters (e.g., degree of polynomial, learning rate in neural net, ..) using validation set. Important: Evaluate once on test set (typically not available).



- When dataset is small, use (k-fold) cross validation instead of a fixed split.

Ridge Regression

Ridge Regression

Polynomial Curve Model:

$$f(x, \mathbf{w}) = \sum_{j=0}^{M} w_j x^j = \mathbf{w}^T \mathbf{x} \qquad \text{with features} \qquad \mathbf{x} = (1, x^1, x^2, \ldots, x^M)^T$$

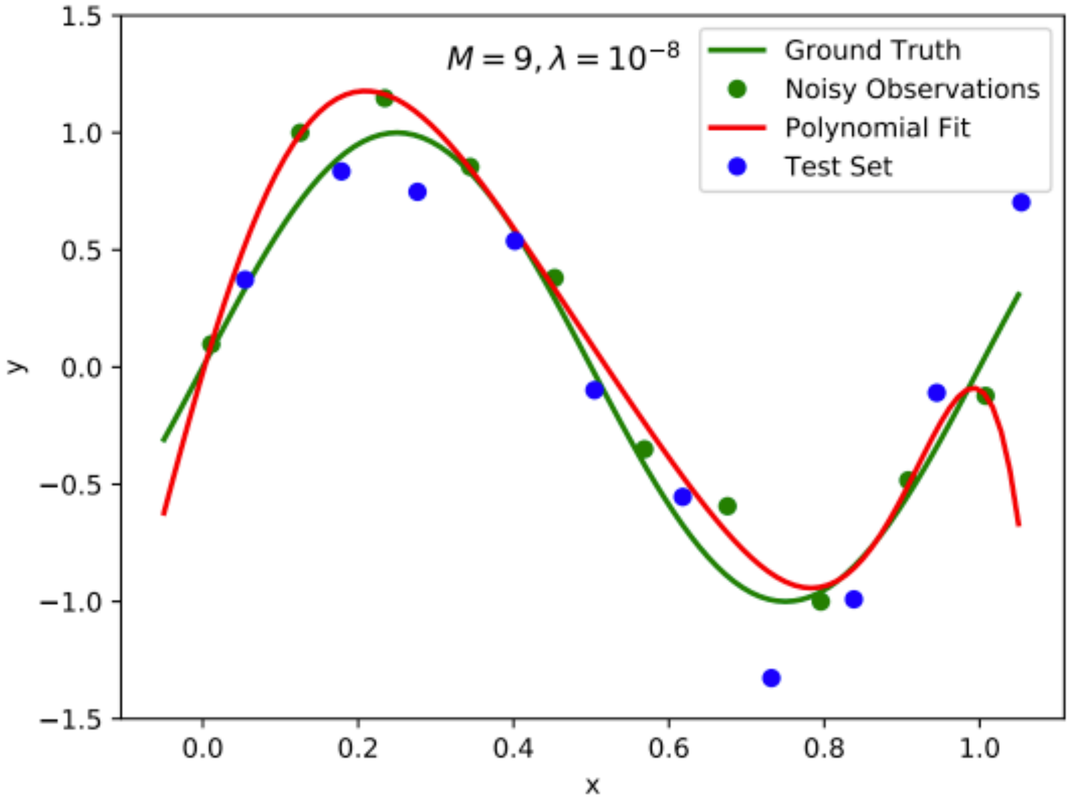
Ridge Regression:

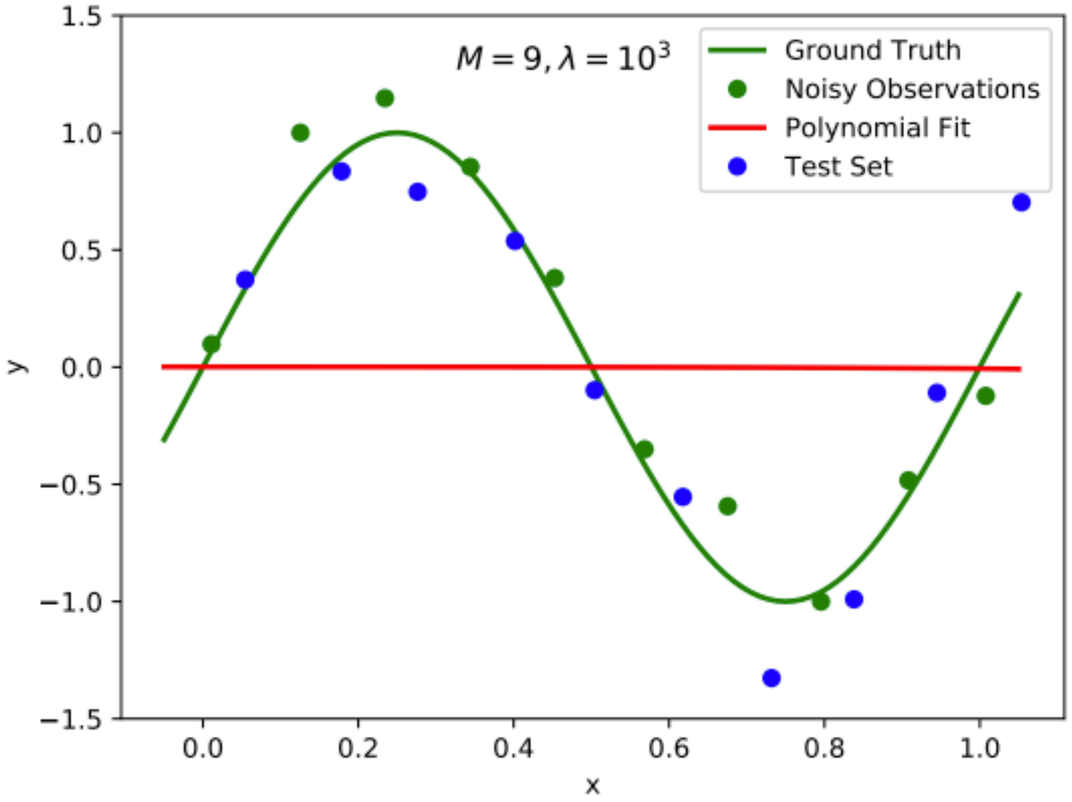
$$E(\mathbf{w}) = \sum_{i=1}^{N} \left(f(x_i, \mathbf{w}) - y_i \right)^2 + \lambda \sum_{j=0}^{M} w^2$$

$$= \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_2^2$$

- Idea: Discourage large parameters by adding a regularization term with strength λ
- Closed form solution: $\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$

Ridge Regression

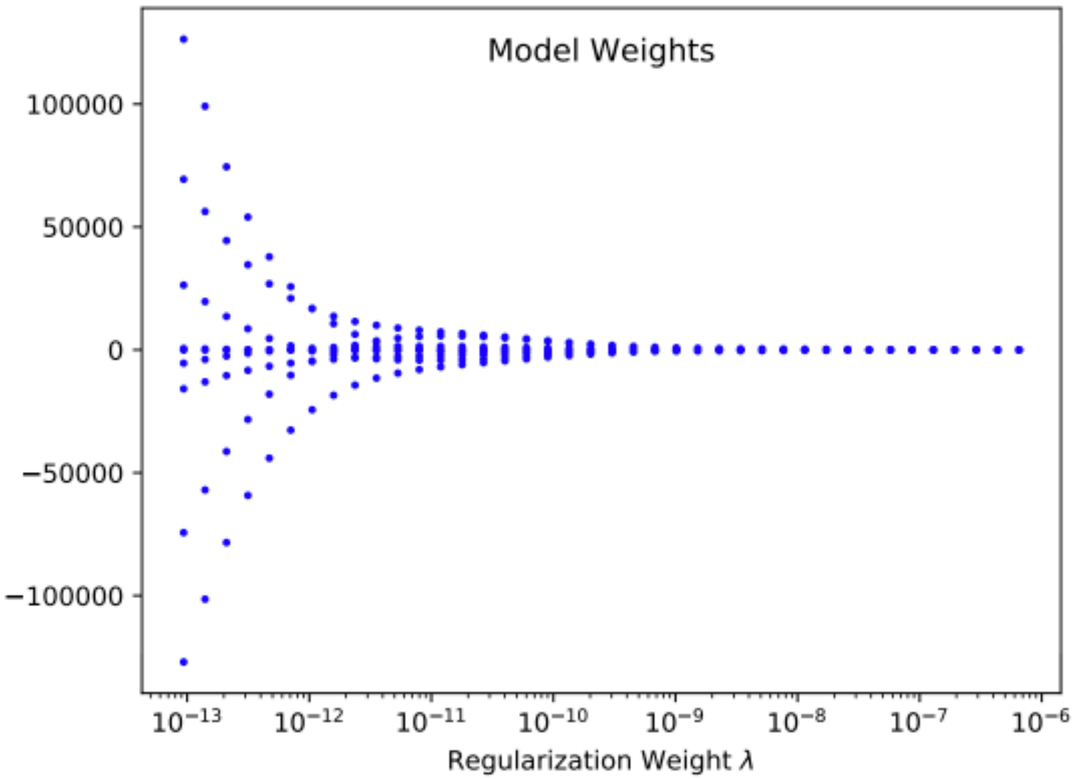


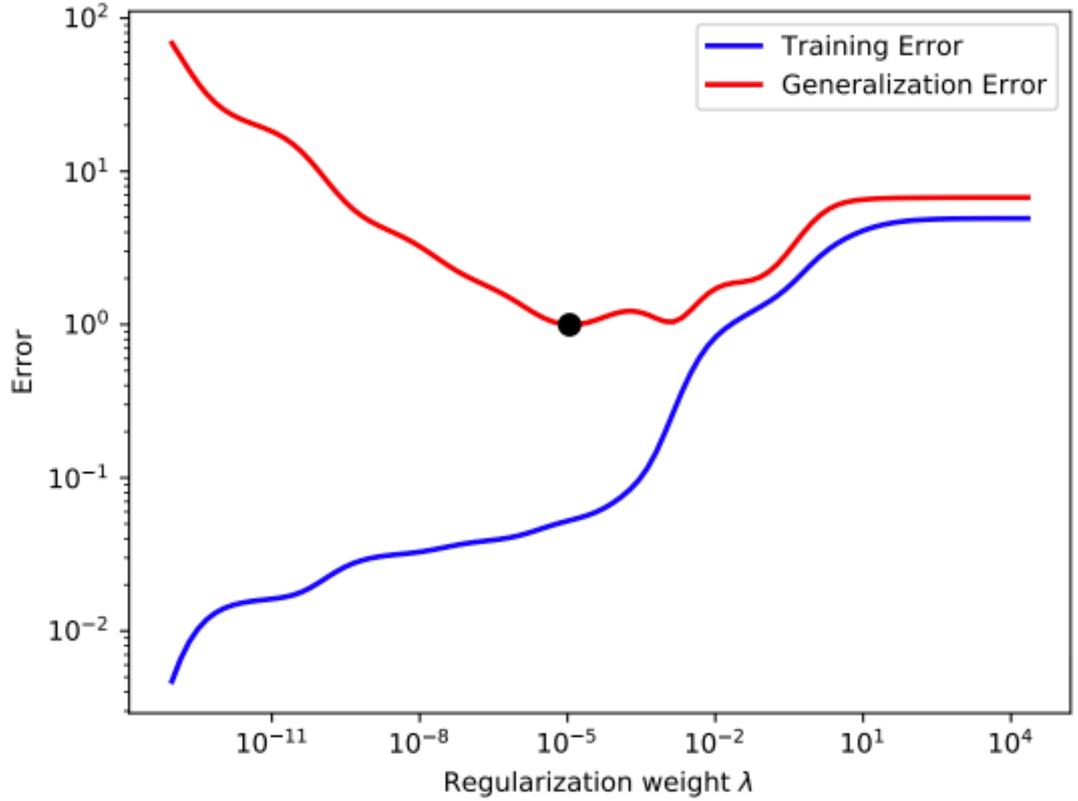


Plots of polynomial with degree $M = 9$ fitted to 10 data points using ridge regression.

Left: weak regularization ($\lambda = 10^{-8}$). **Right**: strong regularization ($\lambda = 10^{3}$).

Ridge Regression





Left: With low regularization, parameters can become very large (ill-conditioning).
Right: Select model with the smallest generalization error on the validation set.

Maximum Likelihood Estimation

Estimators, Bias & Variance

Point Estimator

- A point estimator $g(\cdot)$ is function that maps a dataset $\mathcal{X}$ to model parameters $\hat{\mathbf{w}}$:

$$\hat{\mathbf{w}} = g(\mathcal{X})$$

- Example: Estimator of ridge regression model: $\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$
- We use the hat notation to denote that $\hat{\mathbf{w}}$ is an estimate
- A good estimator is a function that returns a parameter set close to the true one
- The data $\mathcal{X} = \{(x_i, y_i)\}$ is drawn from a random process $(x_i, y_i) \sim p_{data}(\cdot)$
- Thus, any function of the data is random and $\hat{\mathbf{w}}$ is a random variable.

Estimators, Bias & Variance

Properties of Point Estimators

Bias:

$$\text{Bias}(\hat{\mathbf{w}}) = \mathscr{E}(\hat{\mathbf{w}}) - \mathbf{w}$$

- ✦ Expectation over datasets $\mathscr{X}$
- ✦ $\hat{\mathbf{w}}$ is unbiased $\iff \text{Bias}(\hat{\mathbf{w}}) = 0$
- ✦ A good estimator has little bias.

Variance:

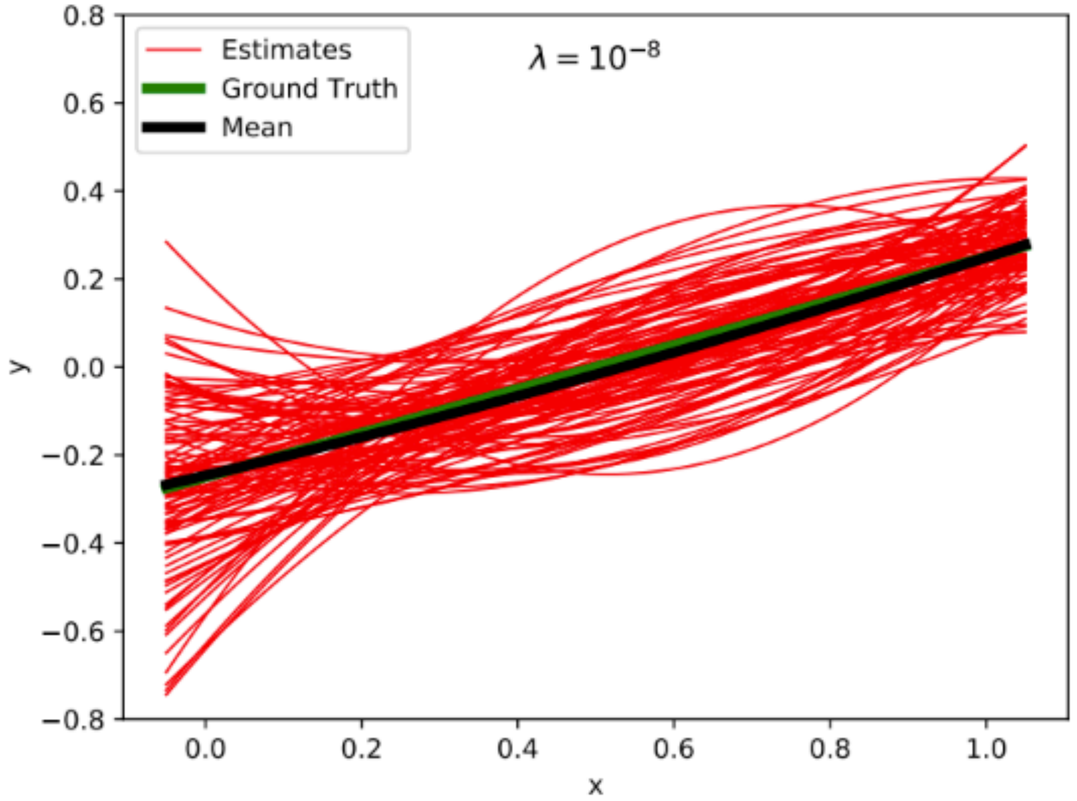
$$\text{Var}(\hat{\mathbf{w}}) = \mathscr{E}(\hat{\mathbf{w}}^2) - \mathscr{E}(\mathbf{w})^2$$

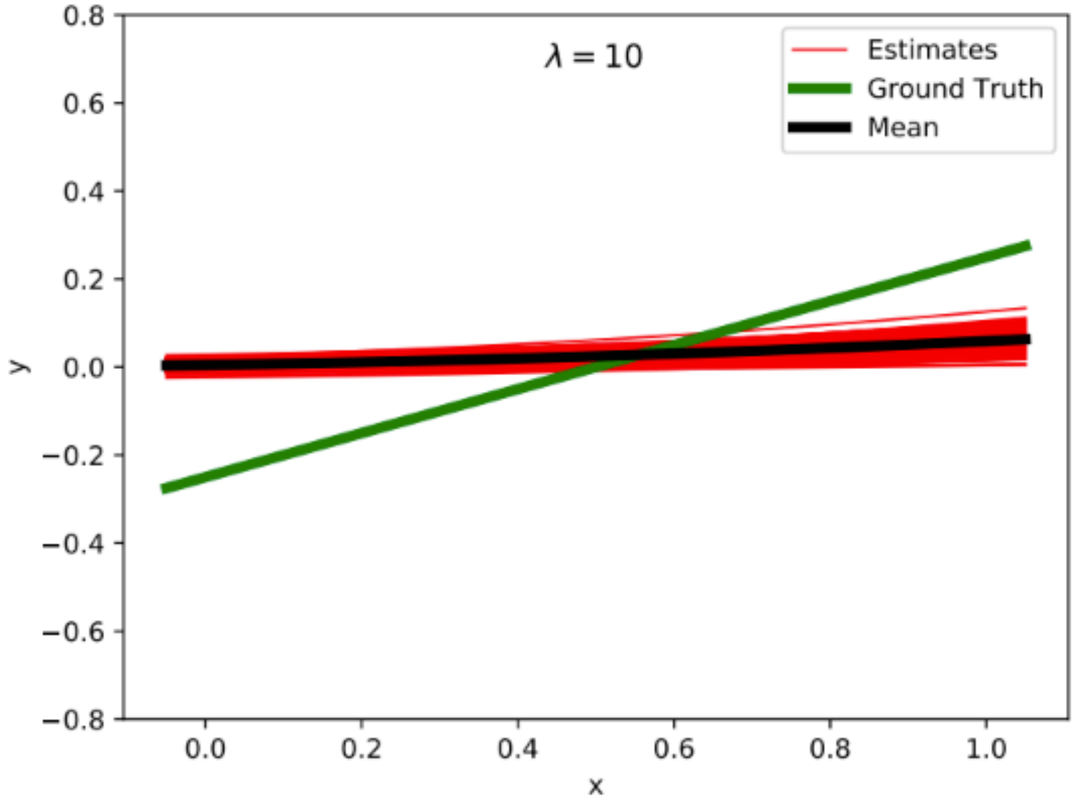
- ✦ Variance over datasets $\mathscr{X}$
- ✦ $\sqrt{\text{Var}(\hat{\mathbf{w}})}$ is called “standard error”
- ✦ A good estimator has low variance.

Bias-Variance Dilemma:

- ✦ Statistical learning theory tells us that we can’t have both $\Rightarrow$ there is a trade-off

Estimators, Bias & Variance

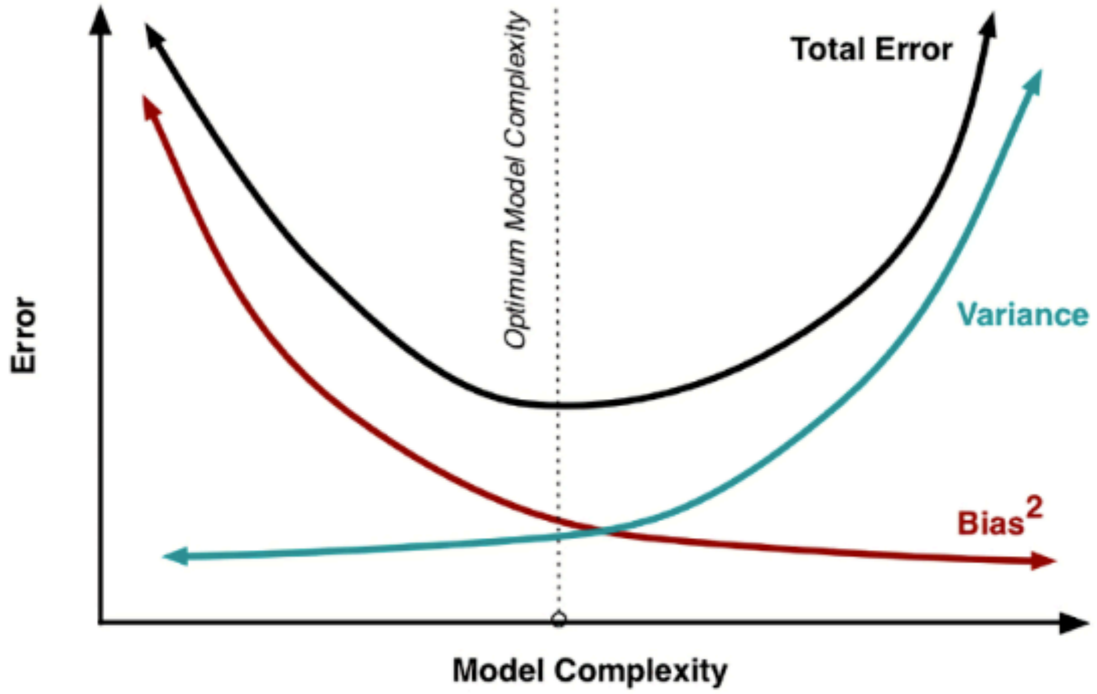




Ridge regression with weak ($\lambda = 10^{-8}$) and strong ($\lambda = 10$) regularization.

Green: True model. Black: Plot of model with mean parameters $\bar{\mathbf{w}} = \mathscr{E}(\mathbf{w})$.

Estimators, Bias & Variance



- There is a bias-variance tradeoff: $\mathscr{E}[(\hat{\mathbf{w}} - \mathbf{w})^2] = \text{Bias}(\hat{\mathbf{w}})^2 + \text{Var}(\hat{\mathbf{w}})$
- Or not? In deep neural networks the test error decreases with network width!
  `https://www.bradyneal.com/bias-variance-tradeoff-textbooks-update`

Estimators, Bias & Variance

Maximum Likelihood Estimation

- We now reinterpret our results by taking a probabilistic viewpoint
- Let $\mathcal{X} = \{(x_i, y_i)\}_{i=1}^N$ be a dataset with samples drawn i.i.d. from p_{data}
- Let the model $p_{model}(y \mid \mathbf{x}, \mathbf{w})$ be a parametric family of probability distributions
- The conditional **maximum likelihood estimator** for $\mathbf{w}$ is given by

$$\begin{aligned}\hat{\mathbf{w}}_{ML} &= \arg\max_{\mathbf{w}} p_{model}(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) \\ &= \arg\max_{\mathbf{w}} \prod_{i=1}^{N} p_{model}(y_i \mid \mathbf{x}_i, \mathbf{w}) \\ &= \arg\max_{\mathbf{w}} \boxed{\sum_{i=1}^{N} \log p_{model}(y_i \mid \mathbf{x}_i, \mathbf{w})}\end{aligned}$$

Log-likelihood

- Note: In this lecture we consider log to be the natural logarithm.

Maximum Likelihood Estimation

Example: Assuming $p_{model}(y \mid \mathbf{x}, \mathbf{w}) = \mathcal{N}(y \mid \mathbf{w}^T\mathbf{x}, \sigma)$, we obtain:

$$\hat{\mathbf{w}}_{ML} = \arg\max_{\mathbf{w}} \sum_{i=1}^{N} \log p_{model}(y_i \mid \mathbf{x}_i, \mathbf{w})$$

$$= \arg\max_{\mathbf{w}} \sum_{i=1}^{N} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(\mathbf{w}^T\mathbf{x}_i - y_i)^2} \right]$$

$$= \arg\max_{\mathbf{w}} - \sum_{i=1}^{N} \frac{1}{2} \log (2\pi\sigma^2) - \sum_{i=1}^{N} \frac{1}{2\sigma^2} (\mathbf{w}^T\mathbf{x}_i - y_i)^2$$

$$= \arg\max_{\mathbf{w}} - \sum_{i=1}^{N} (\mathbf{w}^T\mathbf{x}_i - y_i)^2$$

$$= \arg\min_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

Maximum Likelihood Estimation

We see that choosing $p_{model}(y \mid \mathbf{x}, \mathbf{w})$ to be Gaussian causes maximum likelihood to yield exactly the same least squares estimator derived before:

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

Variations:

- If we were choosing $p_{model}(y \mid \mathbf{x}, \mathbf{w})$ as a Laplace distribution, we would obtain an estimator that minimizes ??

- Assuming a Gaussian distribution over the parameters $\mathbf{w}$ and performing a maximum a-posteriori (MAP) estimation yields ridge regression:

$$\arg \max_{\mathbf{w}} p(\mathbf{w} \mid y, \mathbf{x}) = \arg \max_{\mathbf{w}} p(y \mid \mathbf{x}, \mathbf{w}) p(\mathbf{w})$$

Maximum Likelihood Estimation

We see that choosing $p_{model}(y \mid \mathbf{x}, \mathbf{w})$ to be Gaussian causes maximum likelihood to yield exactly the same least squares estimator derived before:

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

Remarks:

- **Consistency**: As the number of training samples approaches infinity $N \to \inf$, the maximum likelihood (ML) estimate converges to the true parameters
- **Efficiency**: The ML estimate converges most quickly as N increases
- These theoretical considerations make ML estimators appealing.