Machine Learning in Practice: a Crash Course

Lecture 4: Linear Regression & Bias-Variance

胡津铭
DolphinDB

Recap

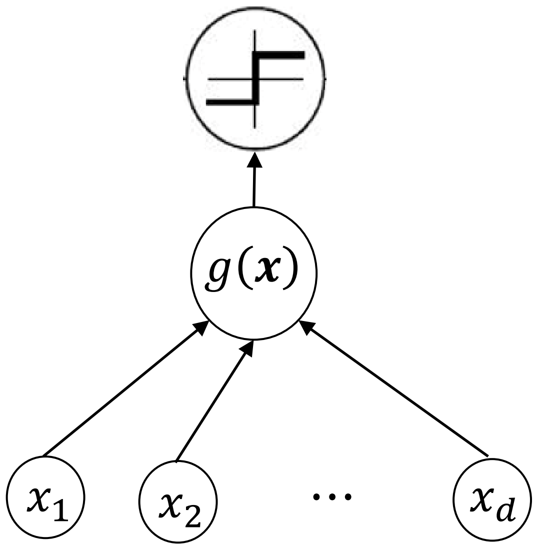
- Our goal (supervised learning):
  - To learn a set of discriminant functions
- Bayesian framework
  - We could design an optimal classifier if we knew:
    - $P(y_i)$: priors and $P(x\ |y_i)$: likelihodd
    - Using training data to estimate $P(y_i)$and $P(x\ |y_i)$
    - $P(y_i|x)$ is computed and be used as the discriminant functions
- Other possible ways?
  - Directly learning discriminant functions from the training data

Today

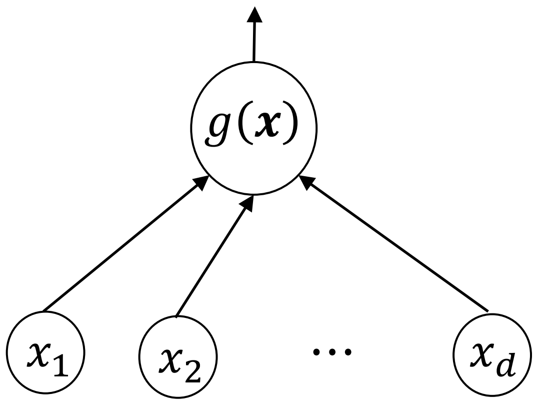
- Linear Regression
- Overfitting
- Bias-Variance

Classification VS Regression

- Both are supervised learning methods
  - Goal: learn a mapping from inputs $\boldsymbol{x}$ to outputs y



- Classification (Categorization, Decision making…)
  - y is a categorical variable



- Regression
  - y is real-valued

Linear model

- Sample: $\boldsymbol{x} \in R^d, \boldsymbol{x} = [x_1, x_2, \cdots x_d]^T$
- Finds a linear function $\boldsymbol{w} = [w_1, w_2, \cdots, w_d]^T \in R^d, b$
- Representation:

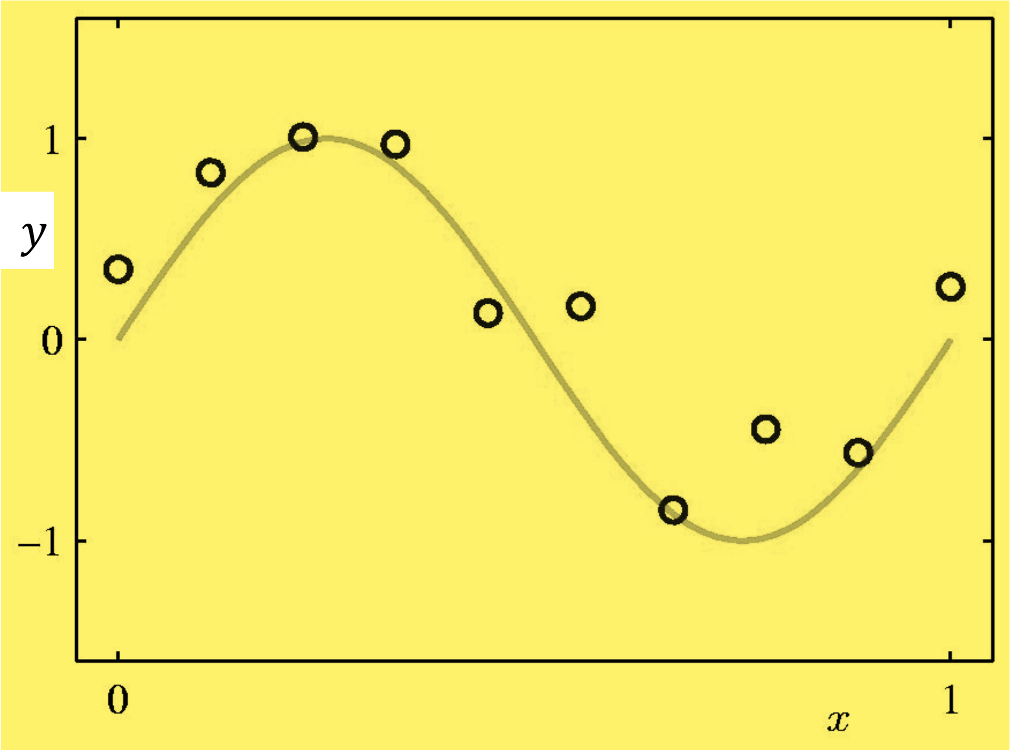
$$f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b$$

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

$$\boldsymbol{x} = [x_1, x_2, \cdots x_d, 1]^T \in R^{d+1}$$

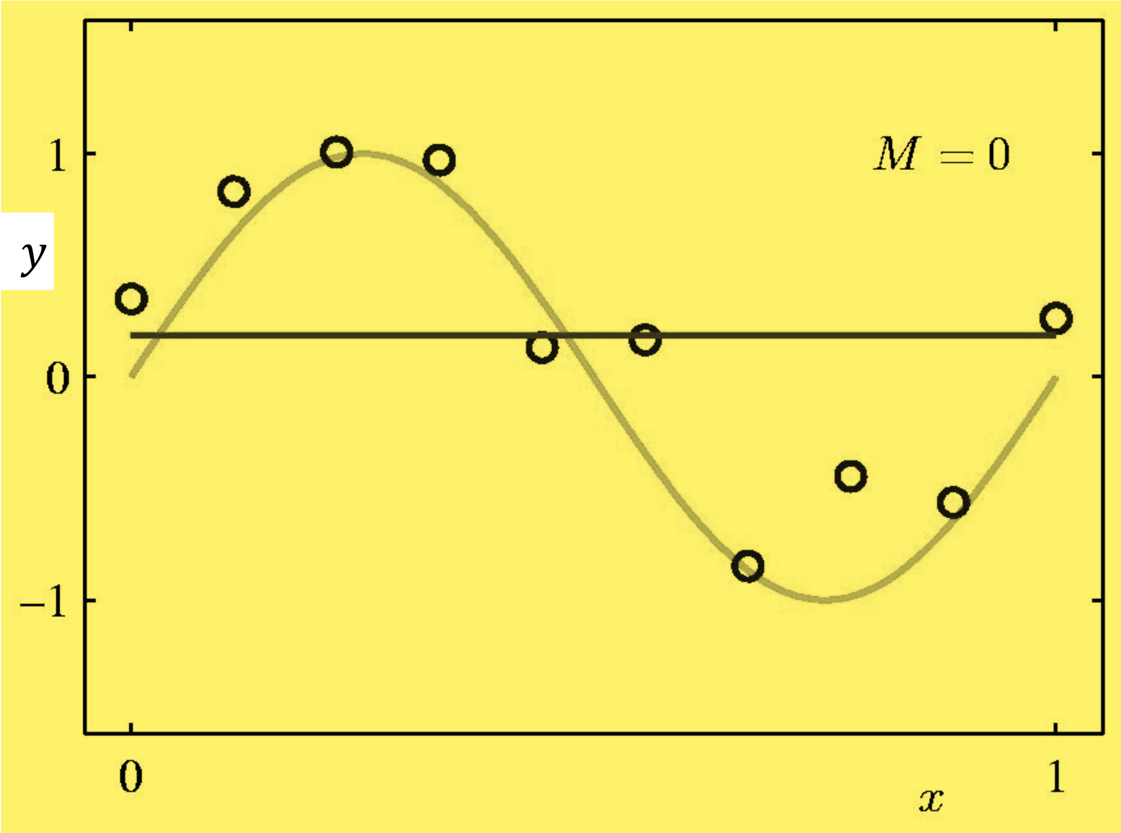
$$\boldsymbol{w} = [w_1, w_2, \cdots w_d, b]^T \in R^{d+1}$$

Polynomial Curve Fitting



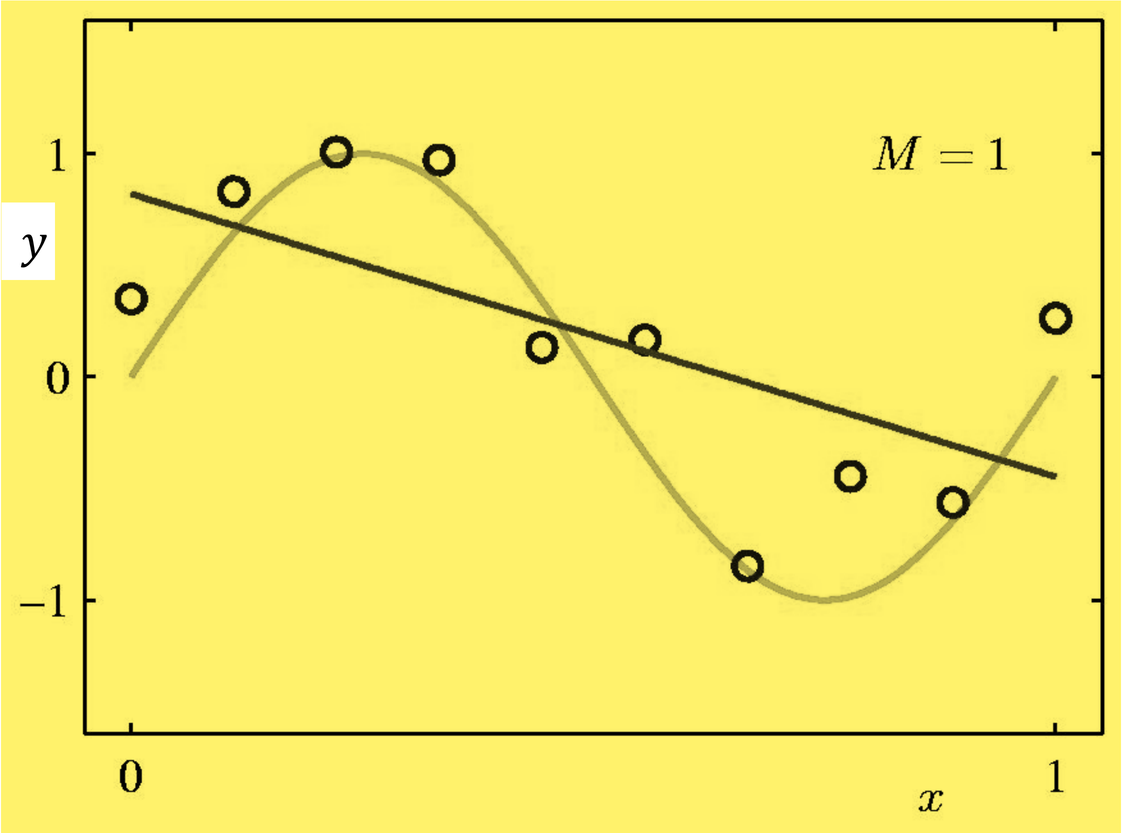
$$f(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

0^{th} Order Polynomial



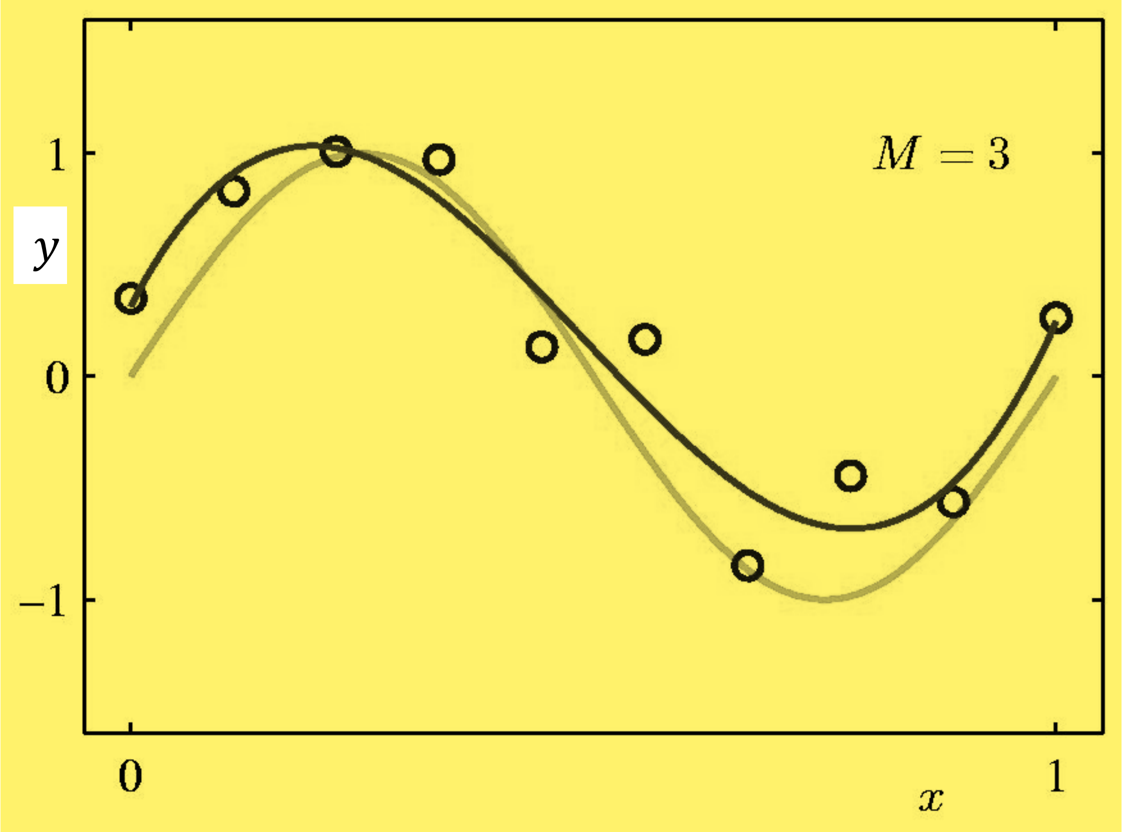
$$f(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

1^{st} Order Polynomial



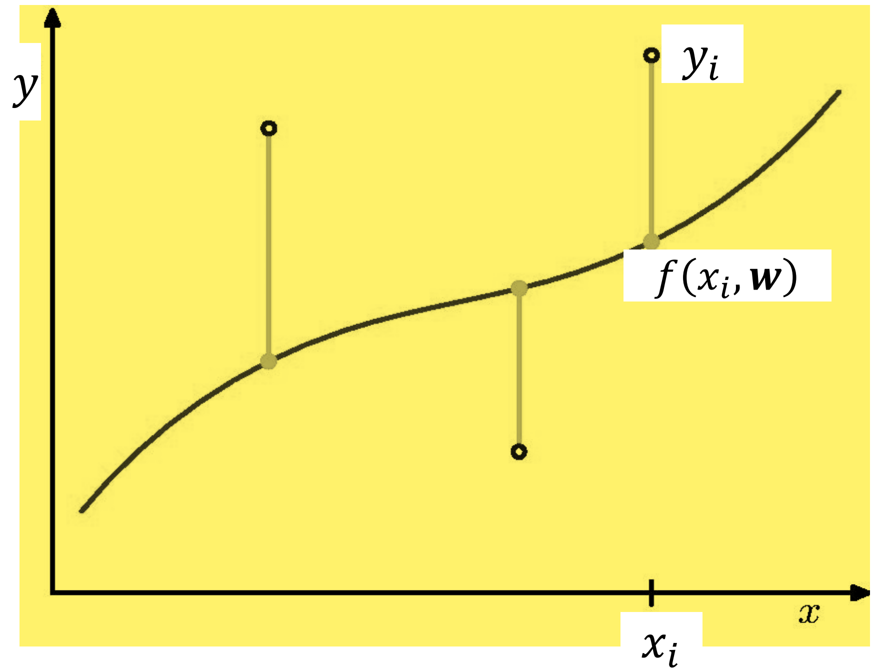
$$f(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

3rd Order Polynomial



$$f(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

Sum-of-Squares Error Function



- Training data：
  - $(x_1, y_1), (x_2, y_2), \cdots (x_n, y_n)$
- To learn f which $f(x) = y$
- Evaluation & Loss： $\mathrm{MSE}(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i, \boldsymbol{w}))^2$

Linear Regression Model

- Training data: $(\boldsymbol{x}_i, y_i)$
- $f(\boldsymbol{x}) = w_0 + \sum_{i=1}^{d} w_i x_i = w_0 + \boldsymbol{w}^T \boldsymbol{x}$
  - $\boldsymbol{w} = [w_1, \cdots, w_d]^T$ and w_0 : unknown parameters or coefficients
  - $\boldsymbol{x}$: Feature vector, the outcome of feature engineering/extraction.

- Minimize the **mean-squared error** :

$$J = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(\boldsymbol{x}_i))^2$$

How to learn the parameters?

- Optimize against the loss function!

$$J = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(\boldsymbol{x}_i))^2$$

- How to?
- 1. Least Square Error Solver
- 2. Gradient Descent

Least Square Errors Solver

$$J = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(\boldsymbol{x}_i))^2$$

- Using matrix notation for convenience

$$X = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n], \qquad \boldsymbol{y} = [y_1, \cdots, y_n]^T$$

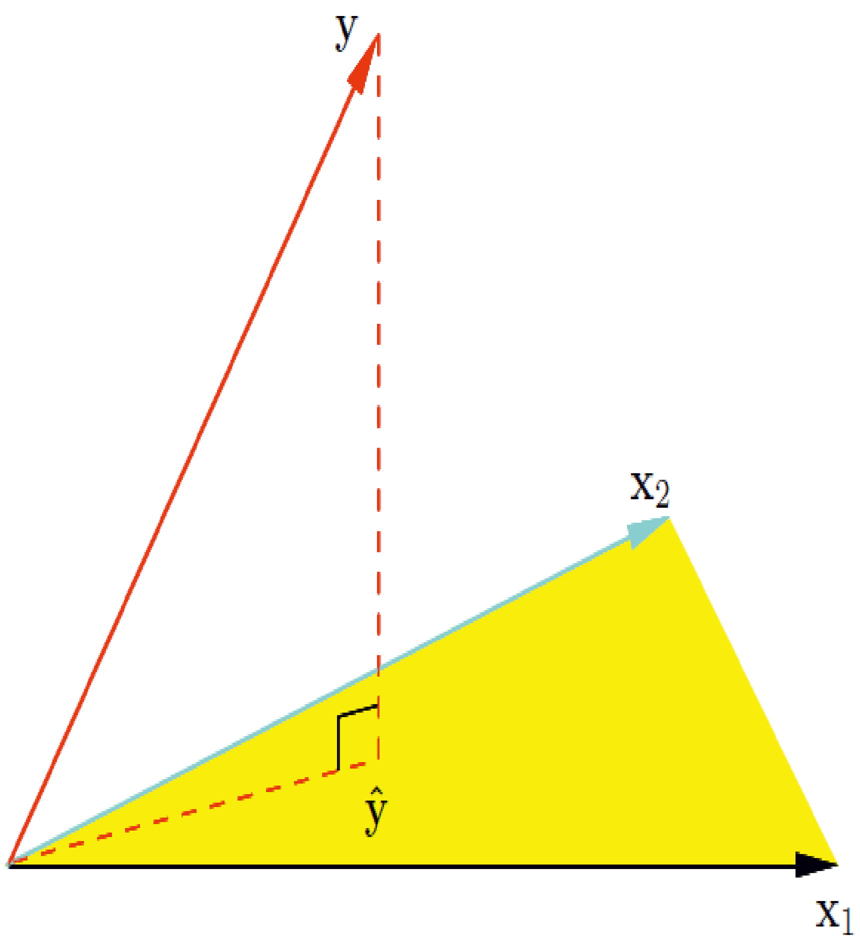
- Loss($\boldsymbol{w}$) = $(\boldsymbol{y} - X^T\boldsymbol{w})^T(\boldsymbol{y} - X^T\boldsymbol{w})$
- Computing the gradient/derivative gives:

$$\nabla J = -2X(\boldsymbol{y} - X^T\boldsymbol{w})$$

- Setting the gradient to zero

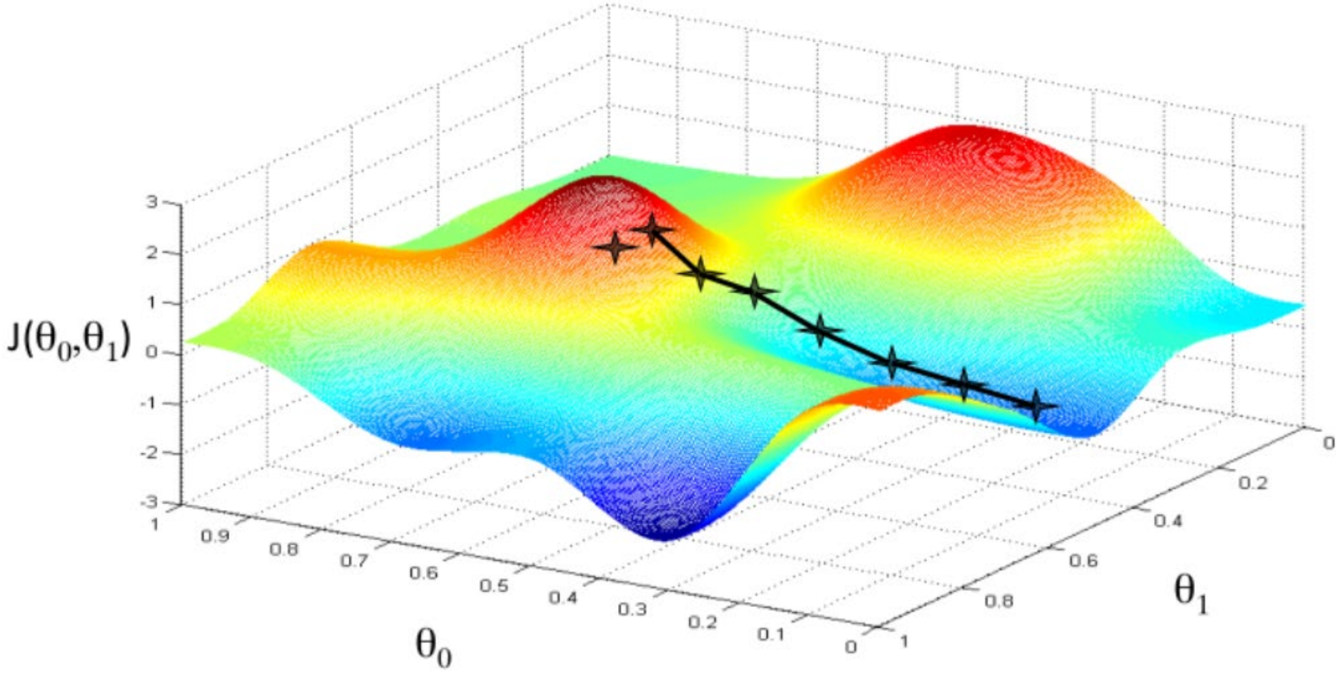
$$XX^T\boldsymbol{w} = X\boldsymbol{y}$$
$$\boldsymbol{w} = (XX^T)^{-1}X\boldsymbol{y}$$

Geometry of least-squares fitting



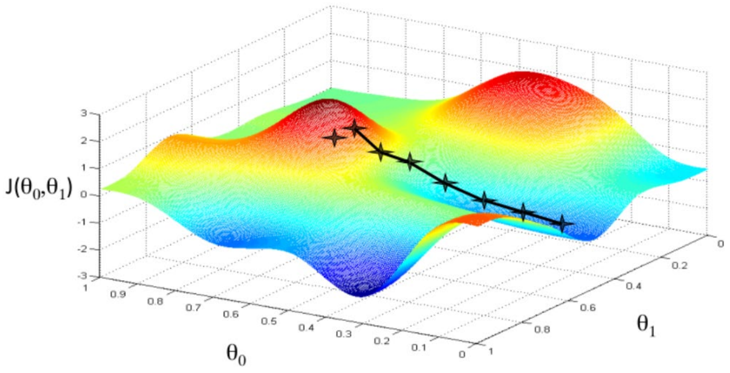
Gradient Descent

Gradient Descent



Gradient Descent

- It can find a local minimum of a function
- One takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point.



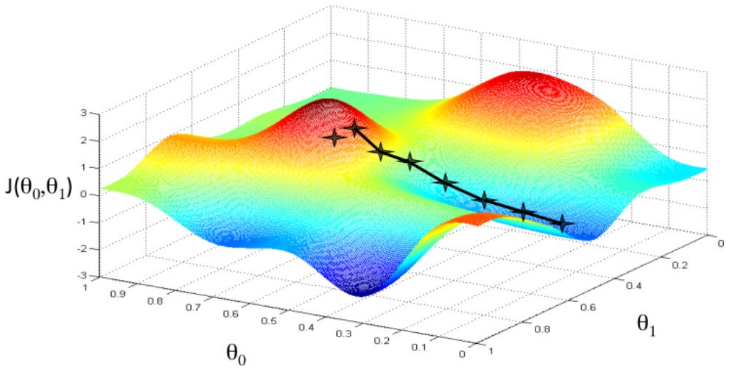
Algorithm 1 (Basic gradient descent)

begin initialize $\mathbf{a}$, criterion $\theta, \eta(\cdot), k = 0$
  do $k \leftarrow k + 1$
    $\mathbf{a} \leftarrow \mathbf{a} - \eta(k)\nabla J(\mathbf{a})$
  until $\eta(k)\nabla J(\mathbf{a}) < \theta$
return $\mathbf{a}$
end

Gradient Descent

Algorithm 1 (Basic gradient descent)

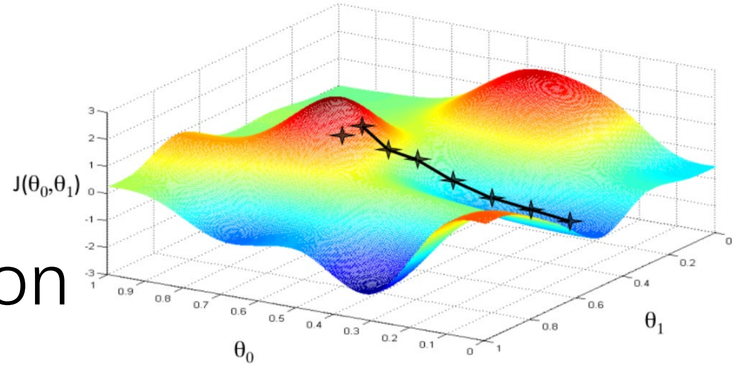
begin initialize $\mathbf{a}$, criterion $\theta, \eta(\cdot), k = 0$
  do $k \leftarrow k + 1$
    $\mathbf{a} \leftarrow \mathbf{a} - \eta(k)\nabla J(\mathbf{a})$
  until $\eta(k)\nabla J(\mathbf{a}) < \theta$
return $\mathbf{a}$
end



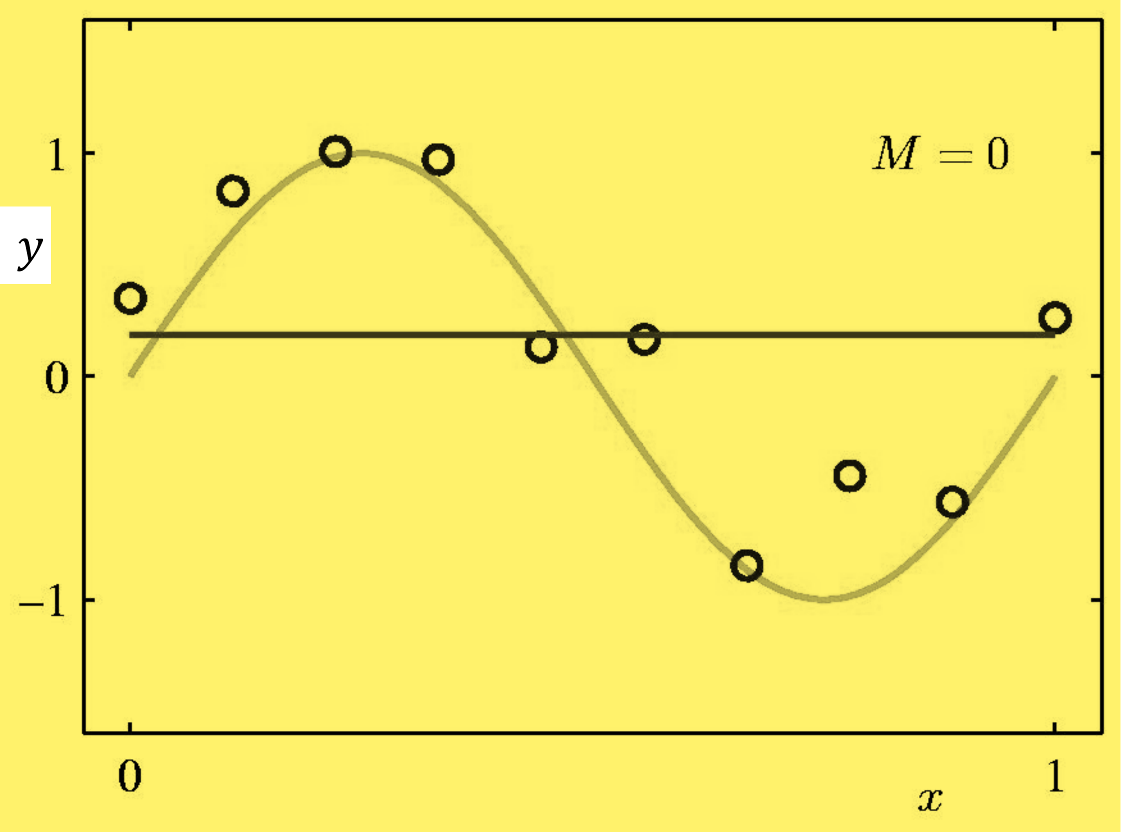
- Time complexity: $O(ndt)$
- Stochastic Gradient Descent(SGD)
  - Randomly select b(often called batch size) samples from the data
  - Time complexity: $O(bdt)$
- SGD(and its variants) are widely used, especially in deep learning

Convex Optimization

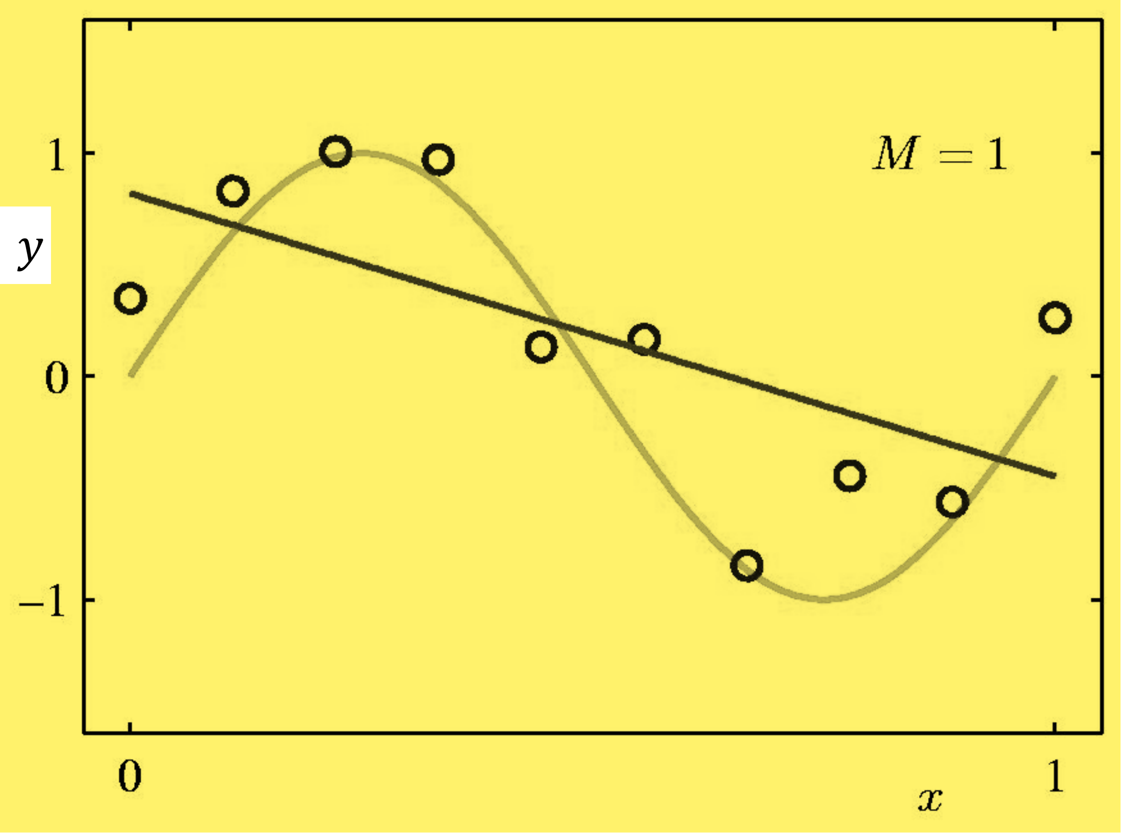
- Least Square Error is a so-called convex problem
- There are a bunch of optimization methods for such problems, called convex optimization
- Gradient Descent: first-order
- Newton's method: second-order
- Quasi Newton's methods
- ……
- If the problem is nonconvex, then can only find the local minimum/optima



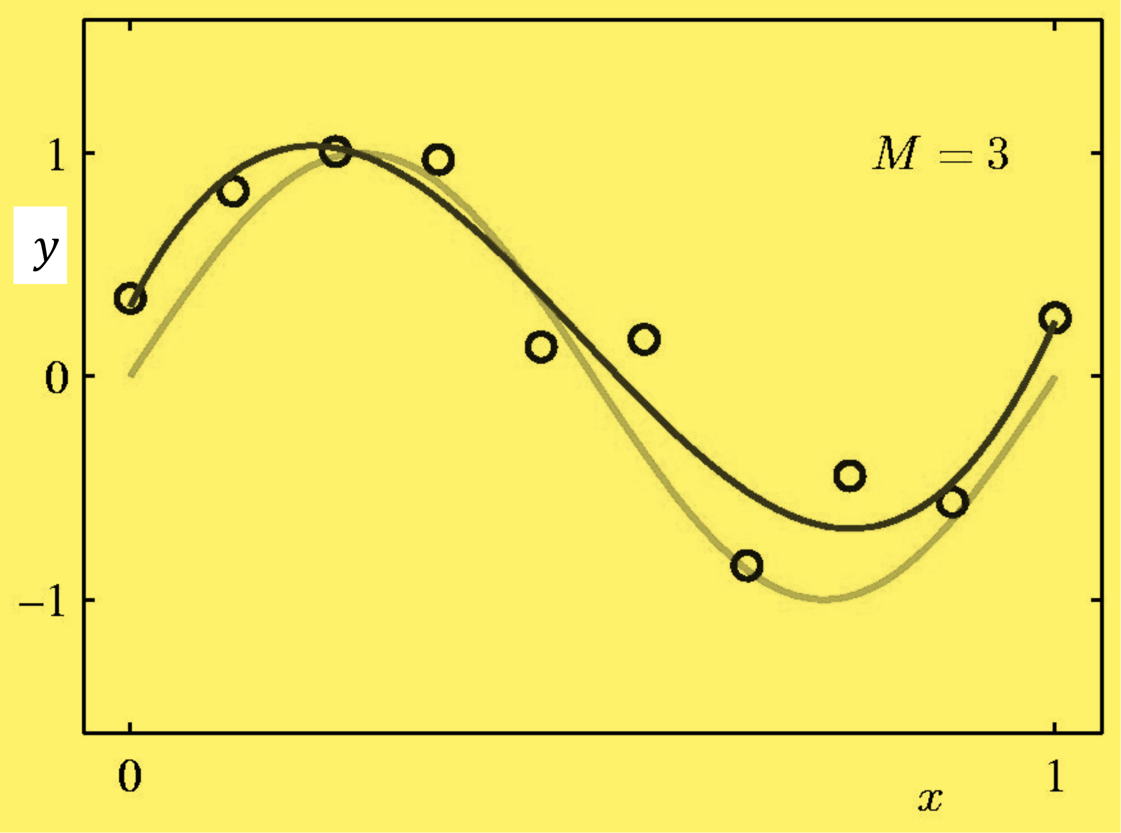
0^{th} Order Polynomial



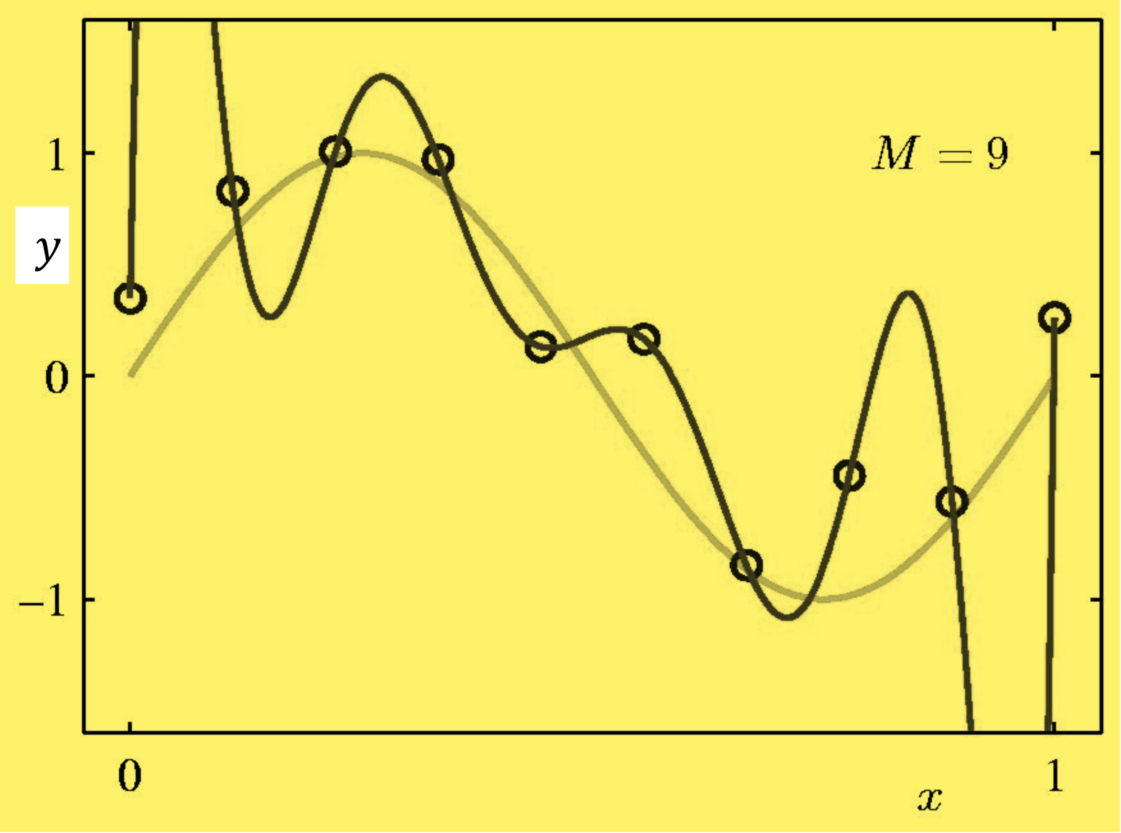
1^{st} Order Polynomial



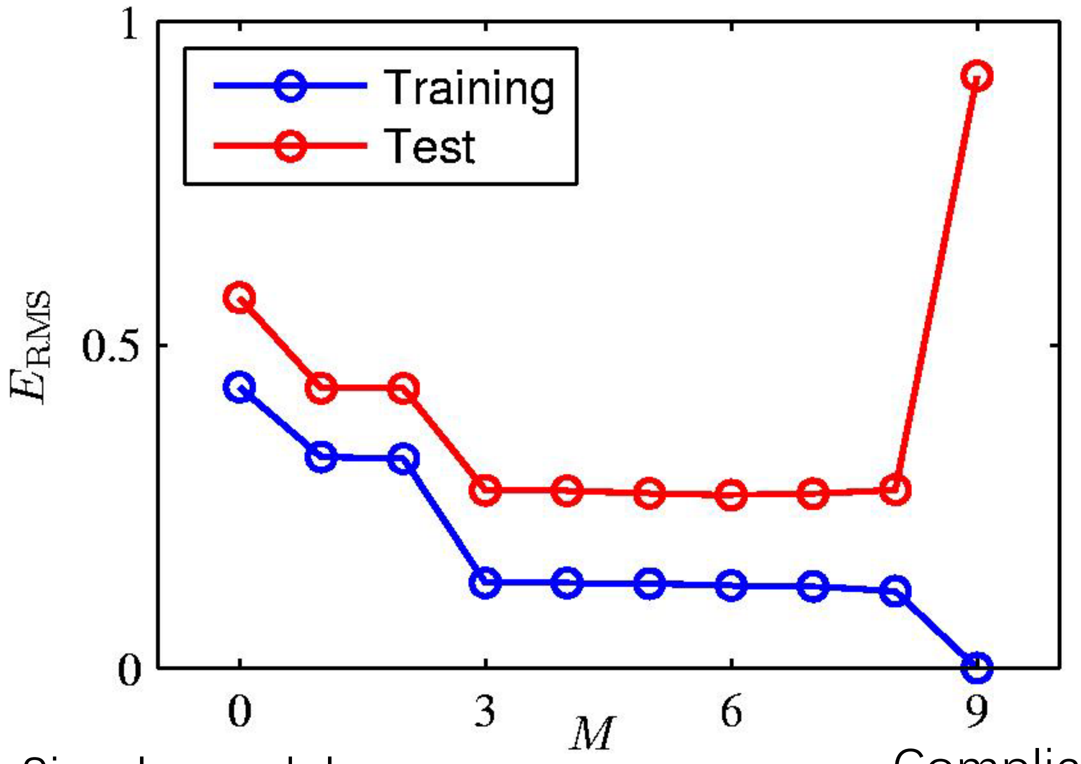
3rd Order Polynomial



9th Order Polynomial



Overfitting



Simple model

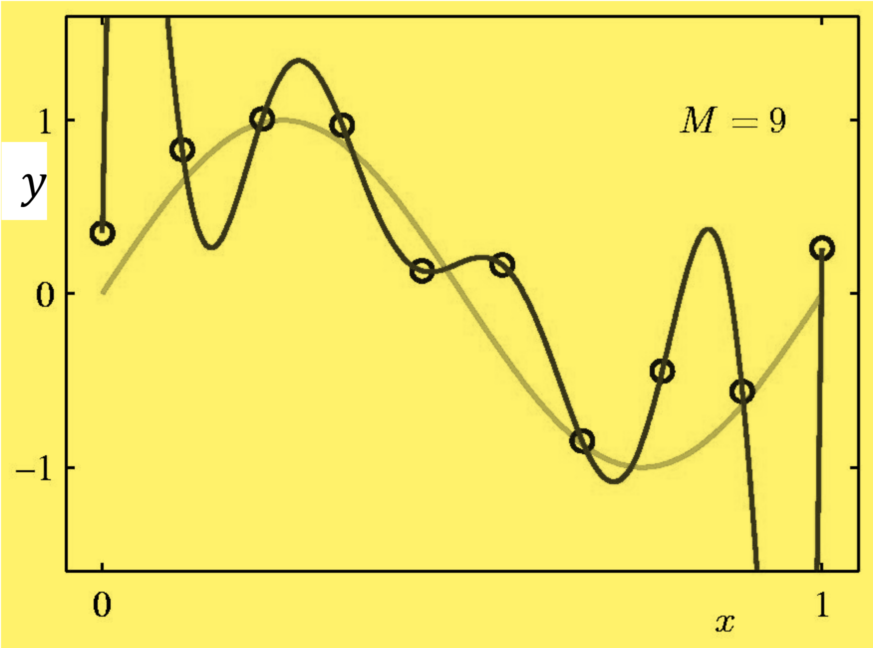
Complicated model

Model complexity

Issues with MSE Criterion

- Coefficients
- Overfit the noise in the data!

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

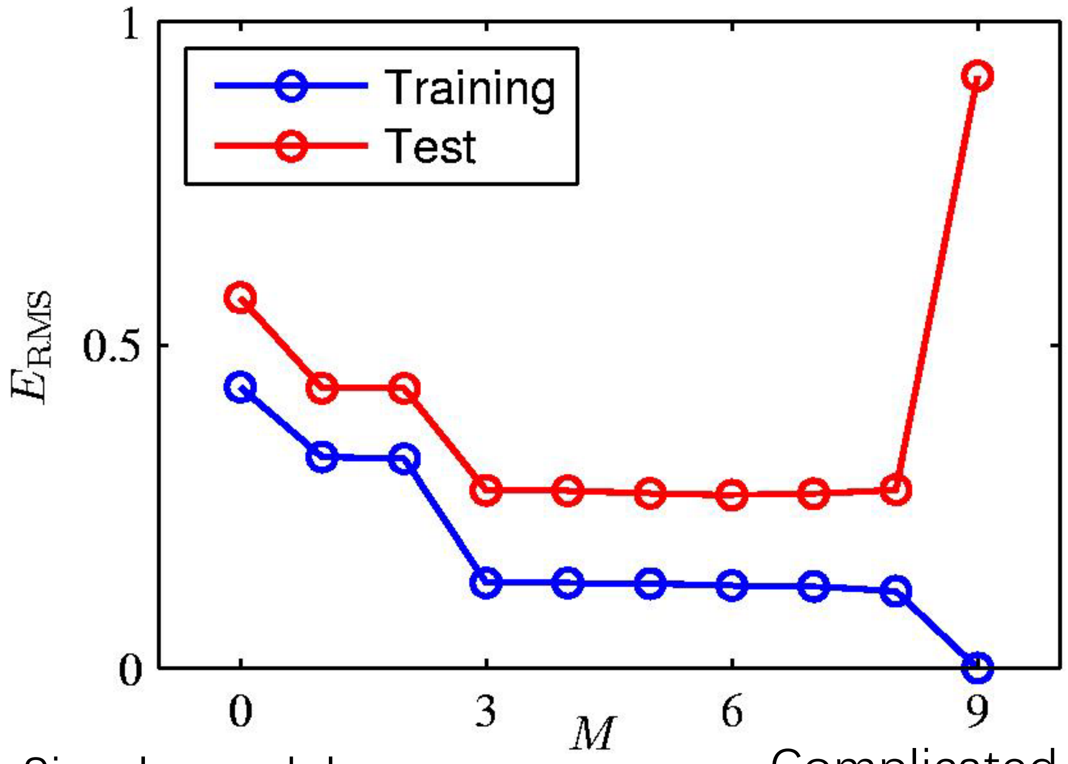


Ridge Regression

- How to control the size of the coefficients in Regression?
- $\boldsymbol{w}^* = \operatorname{argmin} \sum_{i=1}^{n} \left(y_i - \boldsymbol{x}_i^T \boldsymbol{w}\right)^2 + \lambda \sum_{j=1}^{p} w_j^2$
- Local smoothness

Regularization

- Some of our priors to the model
- Reduce the size of hypothesis space
- Some good models may not be learned, but avoid learning some bad models



Simple model

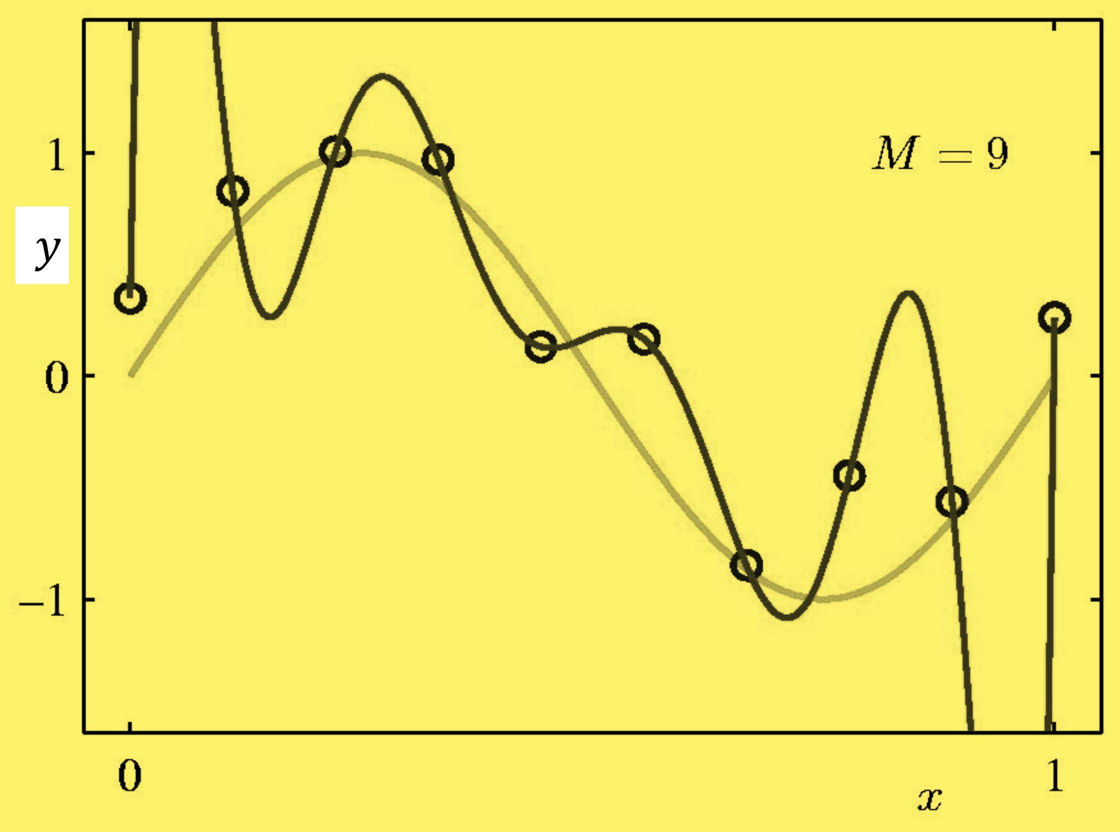
Complicated model

Model complexity

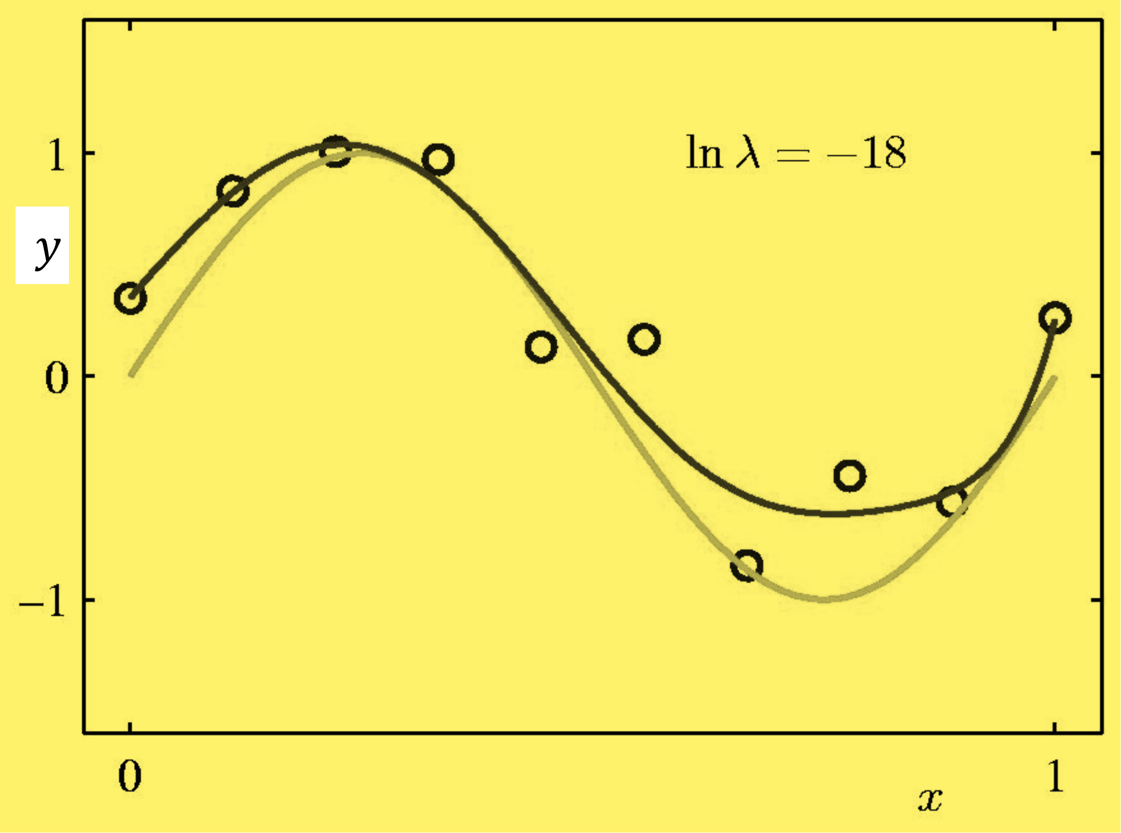
Polynomial Coefficients

	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

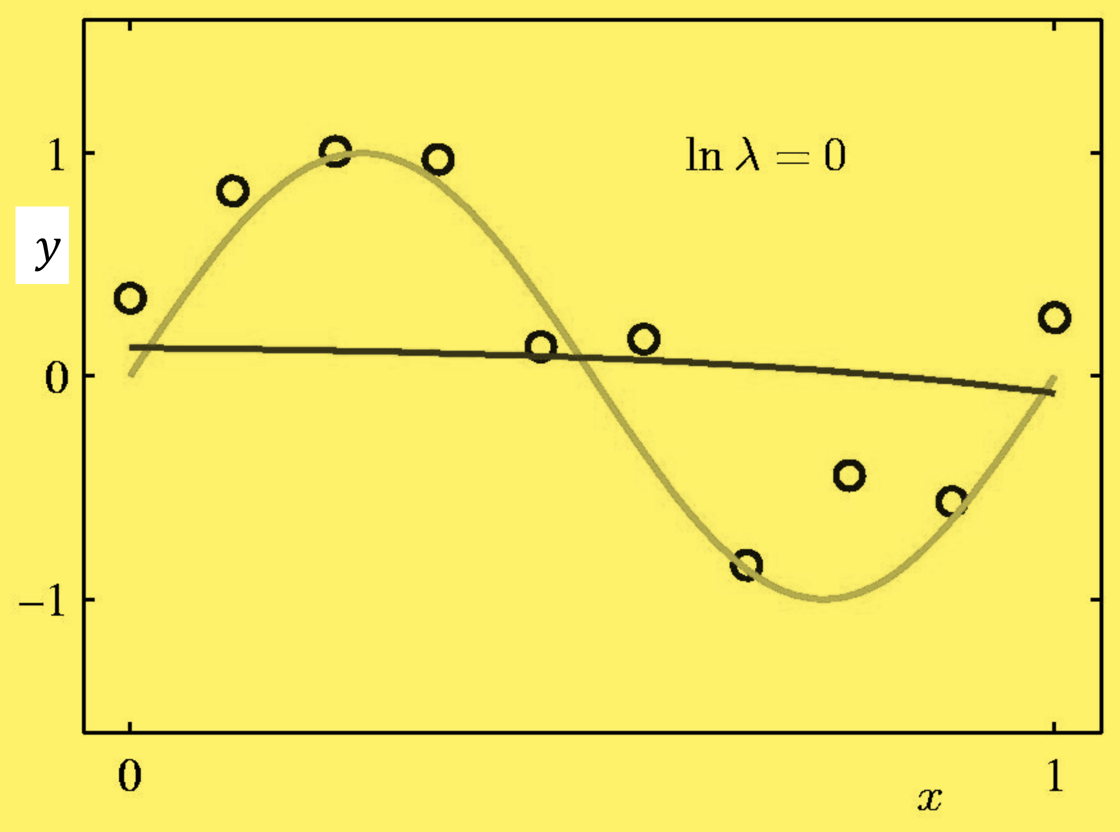
9th Order Polynomial



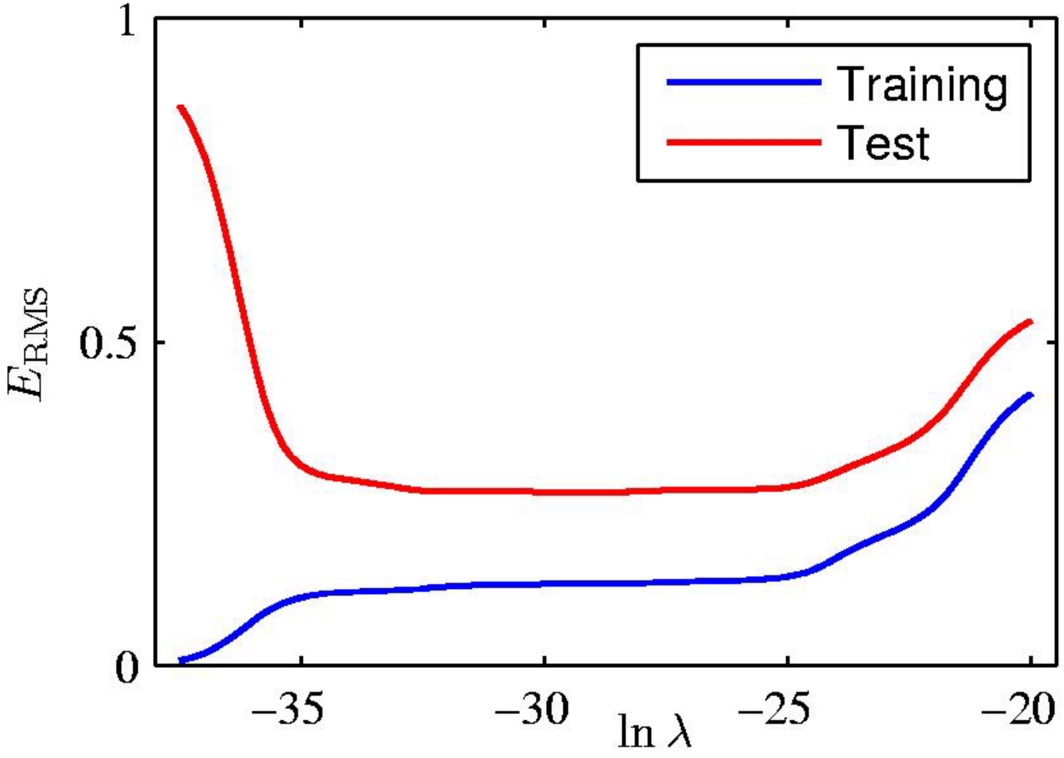
Regularization ($\ln\lambda = -18$)



Regularization ($\ln\lambda = 0$)



Regularization



Complicated model

Simple model

Model complexity

Bias & Variance Decomposition

The Bias-Variance Decomposition

$$
\begin{aligned}
\left(y - f(\boldsymbol{x})\right)^2 &= \left(y - E(y|\boldsymbol{x}) + E(y|\boldsymbol{x}) - f(\boldsymbol{x})\right)^2 \\
&= \left(y - E(y|\boldsymbol{x})\right)^2 + \left(E(y|\boldsymbol{x}) - f(\boldsymbol{x})\right)^2 + 2\left(y - E(y|\boldsymbol{x})\right)\left(E(y|\boldsymbol{x}) - f(\boldsymbol{x})\right)
\end{aligned}
$$

- Expected Prediction Error:

$$
\mathrm{EPE}(f) = \int \left(f(\boldsymbol{x}) - E(y|\boldsymbol{x})\right)^2 p(\boldsymbol{x}) d\boldsymbol{x} + \int \mathrm{var}(y|\boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x}
$$

$$
\boxed{\left[f(\boldsymbol{x}; D) - E_D\left(f(\boldsymbol{x}; D)\right) + E_D\left(f(\boldsymbol{x}; D)\right) - E(y|\boldsymbol{x})\right]^2}
$$

$$
\boxed{\left(f(\boldsymbol{x}; D) - E_D\left(f(\boldsymbol{x}; D)\right)\right)^2 + \left(E_D\left(f(\boldsymbol{x}; D)\right) - E(y|\boldsymbol{x})\right)^2 + 2\left(f(\boldsymbol{x}; D) - E_D\left(f(\boldsymbol{x}; D)\right)\right)\left(E_D\left(f(\boldsymbol{x}; D)\right) - E(y|\boldsymbol{x})\right)}
$$

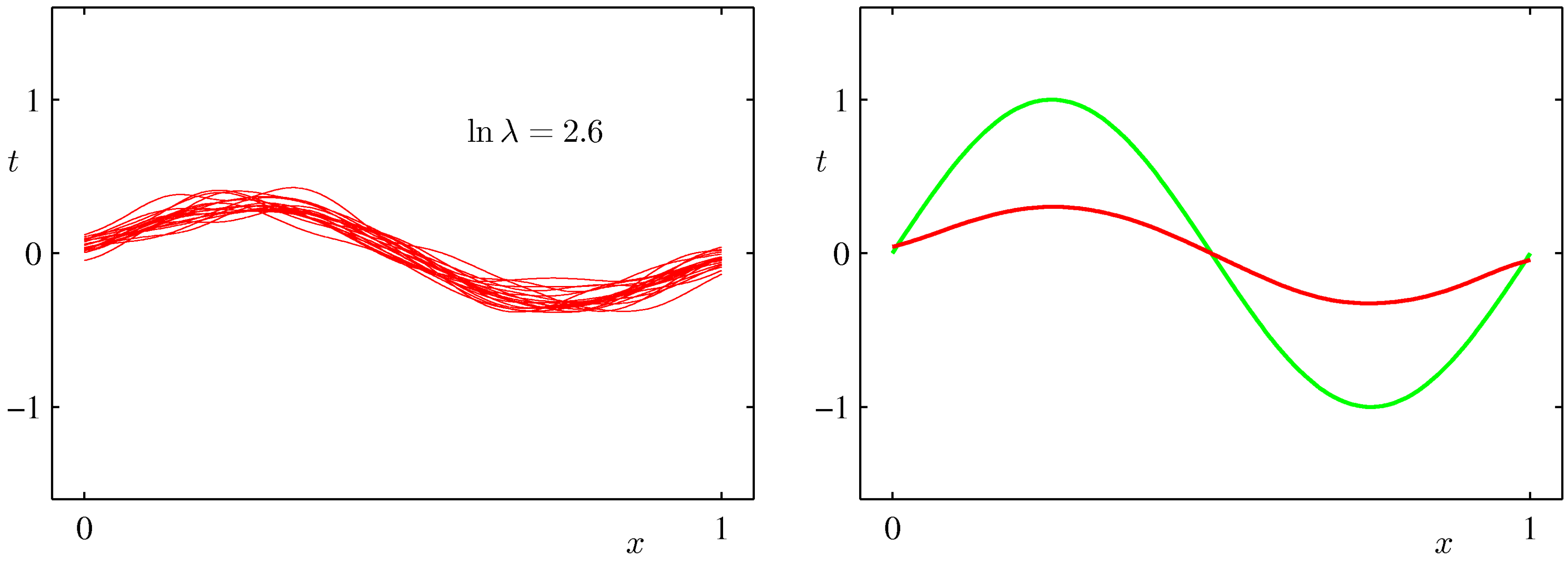
$$
\underbrace{E_D\left\{\left[f(\boldsymbol{x}; D) - E_D\left(f(\boldsymbol{x}; D)\right)\right]^2\right\}}_{\text{Variance}} + \underbrace{\left\{E_D\left(f(\boldsymbol{x}; D)\right) - E(y|\boldsymbol{x})\right\}^2}_{(\text{Bias})^2}
$$

The Bias-Variance Decomposition

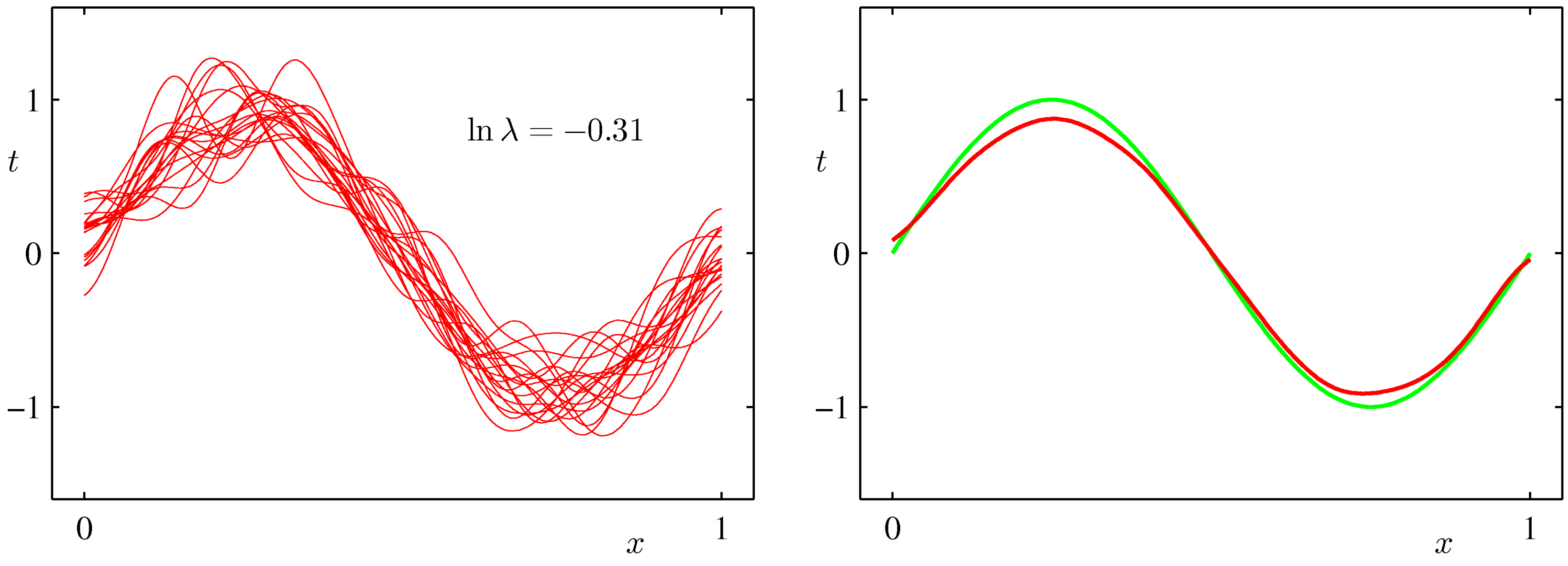
- Through some derivations……
- Prediction Error = bias + variance + noise

The Bias-Variance Decomposition

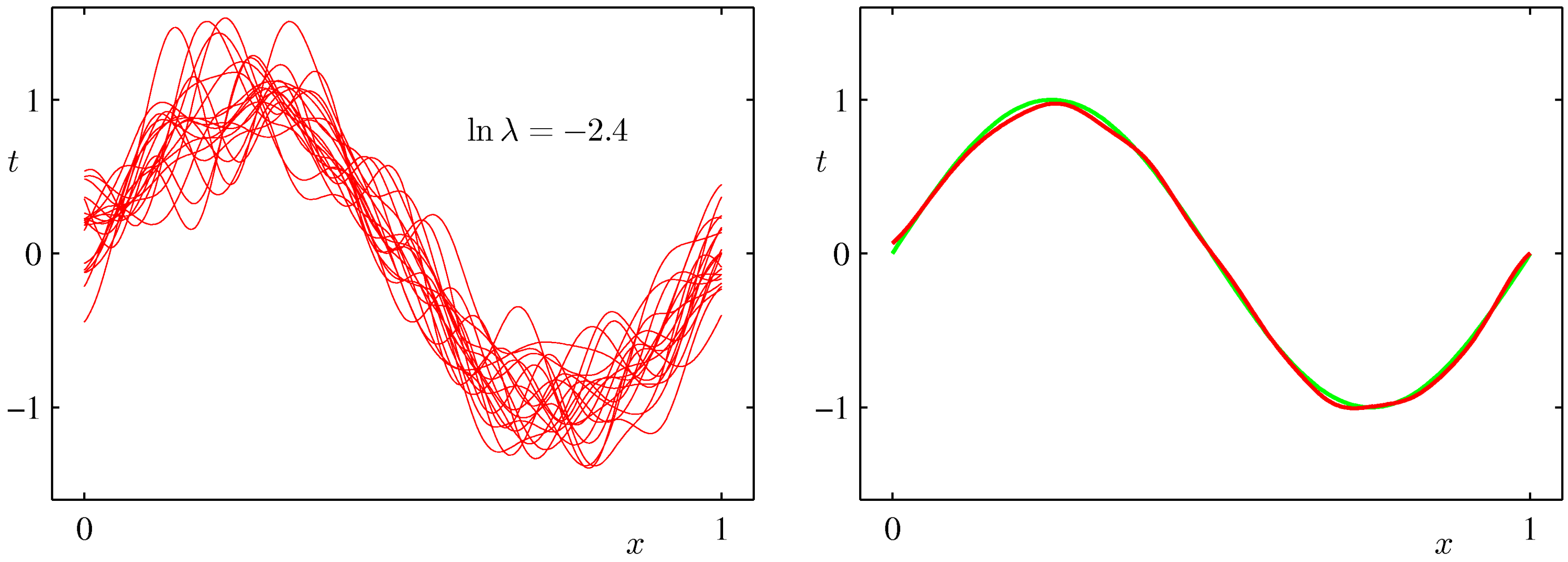
- Example: varying the degree of regularization for some dataset, λ



The Bias-Variance Decomposition



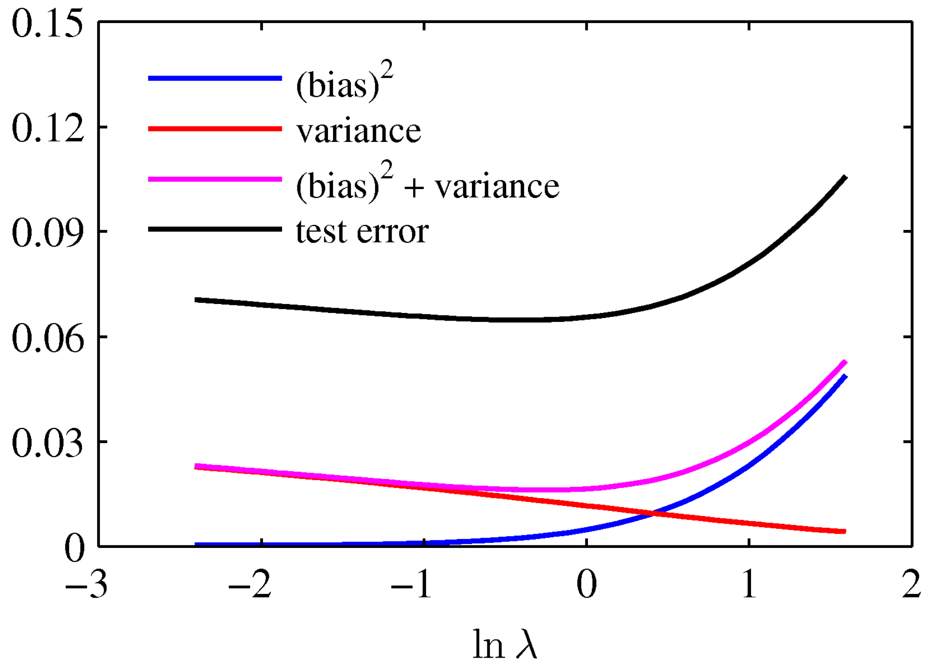
The Bias-Variance Decomposition



The Bias-Variance Trade-off

- Over-regularized model (large λ) →high bias

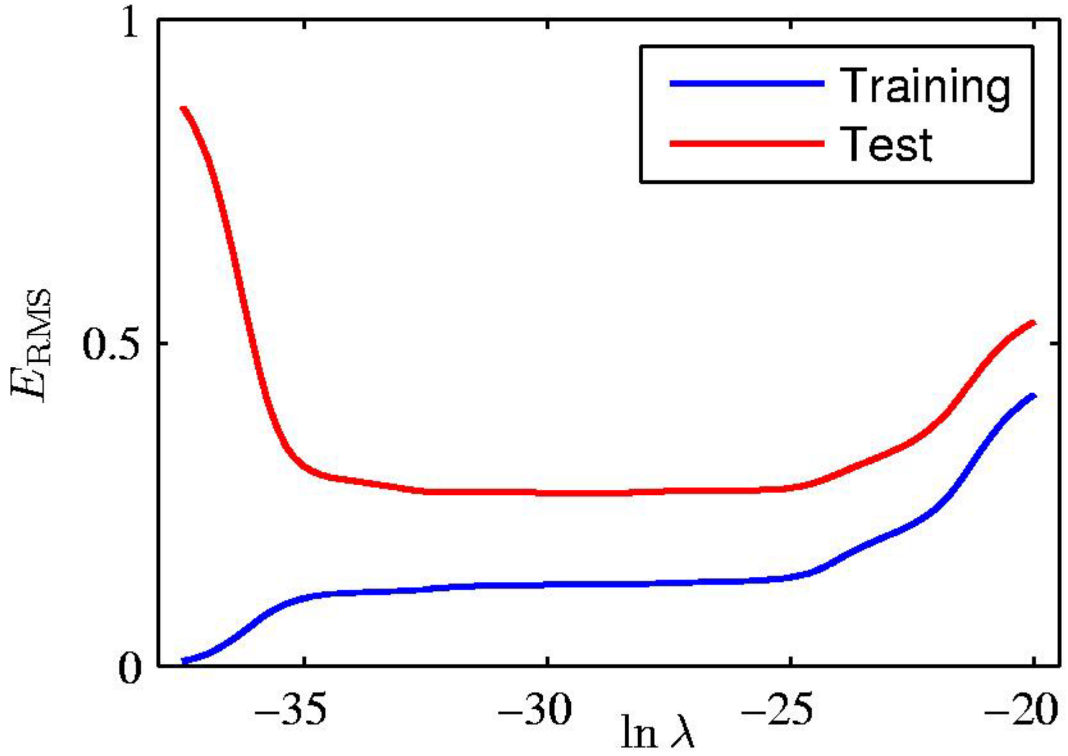
- Under-regularized model (small λ) → high variance.



Underfitting VS Overfitting

- high bias, low variance → underfitting
- high variance, low bias → overfitting
- low bias, low variance → what we want
- Bias: the training error
- Variance: the difference between training error and test error

Underfitting VS Overfitting



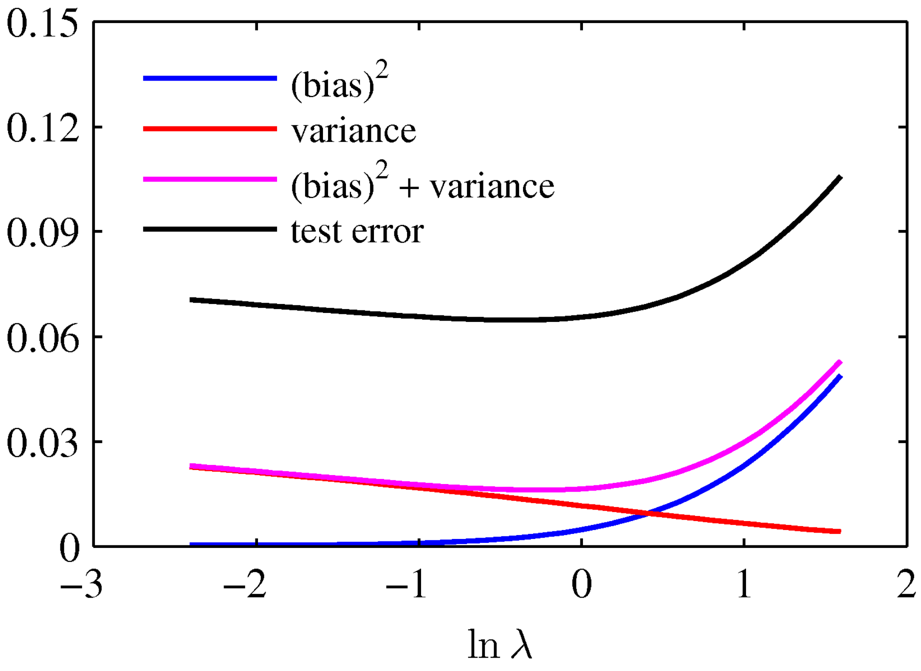
Complicated model

Simple model

Model complexity

How to avoid underfitting/overfitting?

- Underfitting: lower the bias
  - Add more features
  - Use more complex models
  - Decrease the regularization part
- Overfitting: lower the variance
  - Decrease some feature
  - Use simpler models
  - **Add more regularizations**
  - Add more data



Make choice against the test data

- There are many choices in ML models
  - Features
  - Regularization terms
  - Learning rate
  - Different models
  - ……
- How to decide which choice is the best?
  - Look at the training error?
  - Look at the testing error?
- If we make a lot of choices against the test data
  - Can the test performance represent the real-world performance?
  - Overfit the test data!

Train Val Test Split

- Split the dataset into three parts:
  - Train: train the model
  - Val: make choices
    - tune hyperparameters
    - choose models
    - choose features
  - Test: evaluate the final model
- K-Fold Cross-Validation

1	2	3	4	5
Train	Train	Validation	Train	Train

Question?

Thanks and welcome to give us suggestions and feedbacks afterwards.