Regression

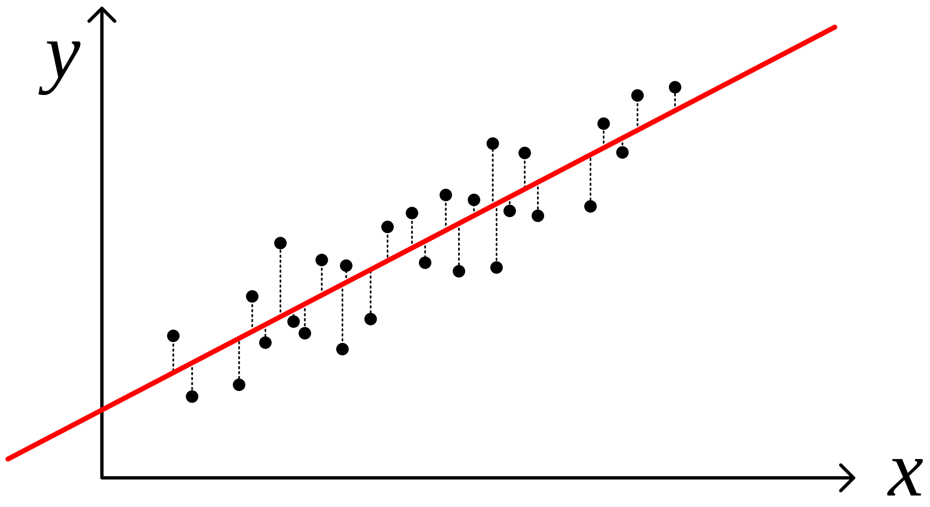
Industrial AI Lab.
Prof. Seungchul Lee
Yunseob Hwang, Juwon Na

Linear Regression

Optimization

$$\min_{\theta_0,\theta_1} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 = \min_{\theta} \|\Phi\theta - y\|_2^2 \qquad \left(\text{same as } \min_{x} \|Ax - b\|_2^2\right)$$

$$\text{solution } \theta^* = (\Phi^T\Phi)^{-1}\Phi^T y$$



Least-Square Solution

y	$\frac{\partial y}{\partial x}$
Ax	A^T
$x^T A$	A
$x^T x$	$2x$
$x^T A x$	$Ax + A^T x$

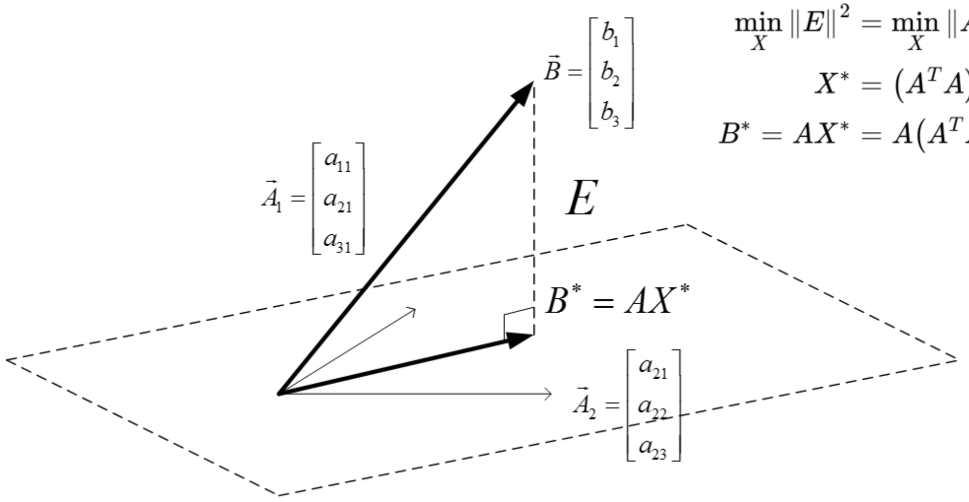
- Scalar Objective: $J = \|Ax - y\|^2$

$$
\begin{aligned}
J(x) &= (Ax - y)^T (Ax - y) \\
&= \left(x^T A^T - y^T\right)(Ax - y) \\
&= x^T A^T A x - x^T A^T y - y^T A x + y^T y
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial J}{\partial x} &= A^T A x + \left(A^T A\right)^T x - A^T y - \left(y^T A\right)^T \\
&= 2A^T A x - 2A^T y = 0
\end{aligned}
$$

$$
\Longrightarrow \quad \left(A^T A\right) x = A^T y
$$

$$
\therefore \quad x^* = \left(A^T A\right)^{-1} A^T y
$$



$$
\begin{aligned}
\min_X \|E\|^2 &= \min_X \|AX - B\|^2 \\
X^* &= \left(A^T A\right)^{-1} A^T B \\
B^* = AX^* &= A\left(A^T A\right)^{-1} A^T B
\end{aligned}
$$

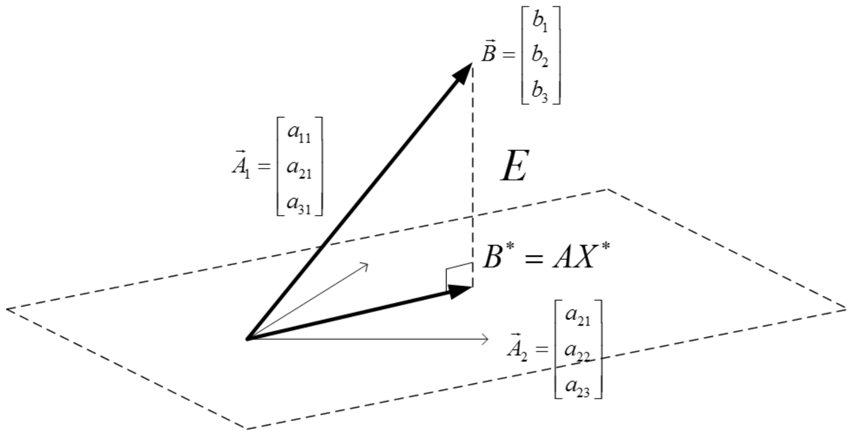
Optimization: Note

$$\begin{array}{ccccc} \text{input} & & \text{feature} & & \text{predicted output} \\ x_i & \rightarrow & \begin{bmatrix} 1 \\ x_i \end{bmatrix} & \rightarrow & \hat{y}_i \end{array}$$

$$\begin{array}{ccccc} \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_m \end{bmatrix} & \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} & = & \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \\ \uparrow \quad\; \uparrow & \uparrow & & \uparrow \\ \vec{A}_1 \quad \vec{A}_2 & \vec{x} & & \vec{B} \end{array}$$

over-determined or projection

$$A(=\Phi) = \left[\vec{A}_1 \ \vec{A}_2\right]$$



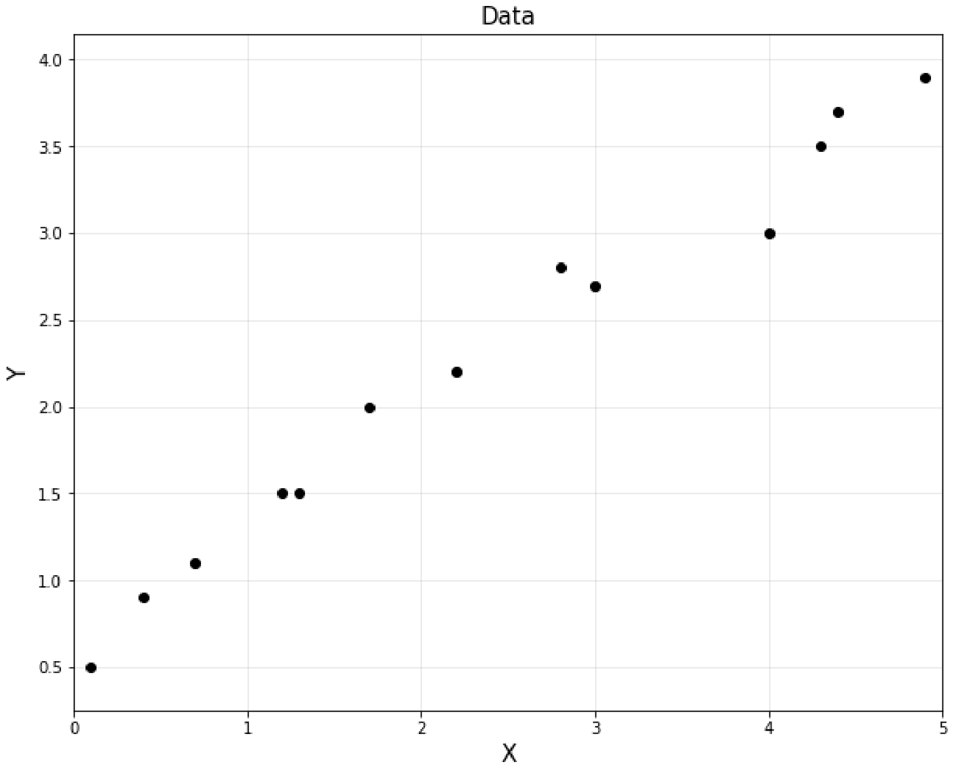
the same principle in a higher dimension

1. Solve using Linear Algebra

- known as *least square*

$$\theta = (A^T A)^{-1} A^T y$$

```
# data points in column vector [input, output]
x = np.array([0.1, 0.4, 0.7, 1.2, 1.3, 1.7, 2.2, 2.8, 3.0, 4.0, 4.3, 4.4, 4.9]).reshape(-1, 1)
y = np.array([0.5, 0.9, 1.1, 1.5, 1.5, 2.0, 2.2, 2.8, 2.7, 3.0, 3.5, 3.7, 3.9]).reshape(-1, 1)
```



1. Solve using Linear Algebra

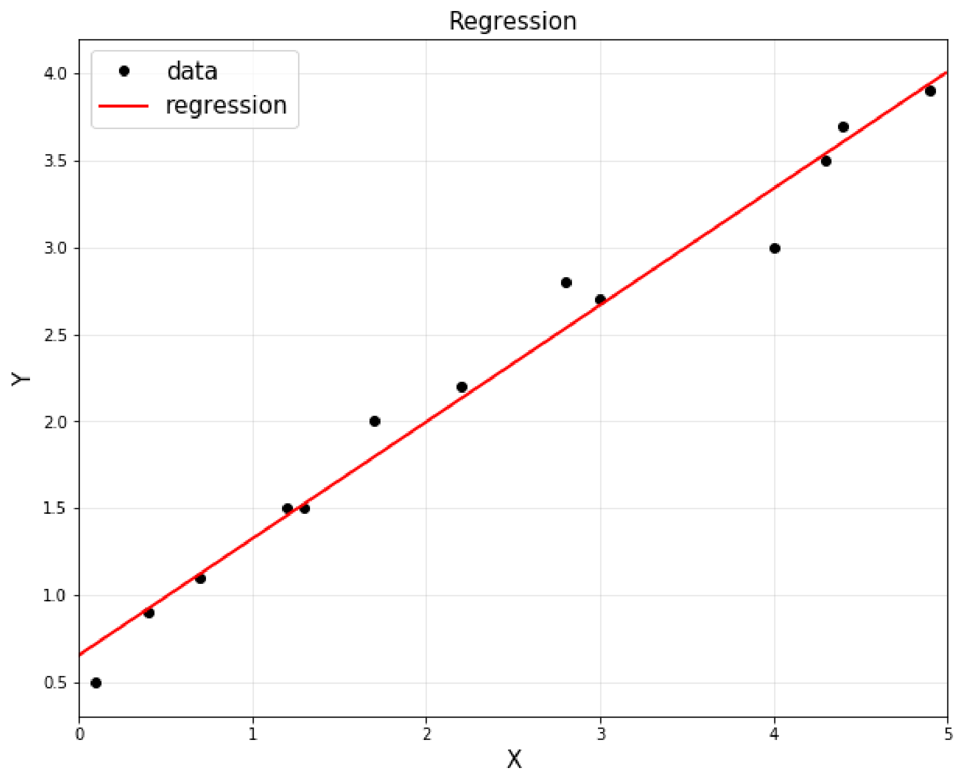
- known as *least square*

$$\theta = (A^T A)^{-1} A^T y$$

```
m = y.shape[0]
#A = np.hstack([np.ones([m, 1]), x])
A = np.hstack([x**0, x])
A = np.asmatrix(A)

theta = (A.T*A).I*A.T*y

print('theta:\n', theta)
```

```
theta:
 [[0.65306531]
 [0.67129519]]
```

```
# to plot
plt.figure(figsize=(10, 8))
plt.title('Regression', fontsize=15)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.plot(x, y, 'ko', label="data")

# to plot a straight line (fitted line)
xp = np.arange(0, 5, 0.01).reshape(-1, 1)
yp = theta[0,0] + theta[1,0]*xp
```



2. Solve using Gradient Descent

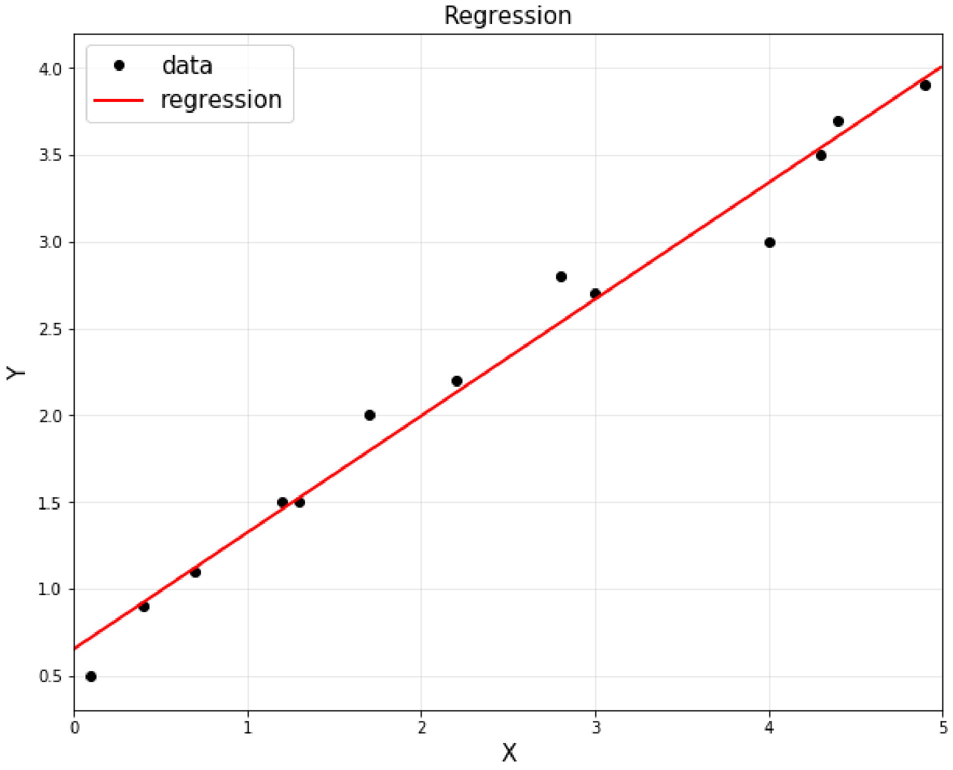
$$\min_{\theta} \; \|\hat{y} - y\|_2^2 = \min_{\theta} \; \|A\theta - y\|_2^2$$

$$\begin{aligned} f &= (A\theta - y)^T(A\theta - y) = (\theta^T A^T - y^T)(A\theta - y) \\ &= \theta^T A^T A\theta - \theta^T A^T y - y^T A\theta + y^T y \end{aligned}$$

$$\nabla f = A^T A\theta + A^T A\theta - A^T y - A^T y = 2(A^T A\theta - A^T y)$$

$$\theta \leftarrow \theta - \alpha \nabla f$$

```
theta = np.random.randn(2,1)
theta = np.asmatrix(theta)

alpha = 0.001

for _ in range(1000):
    df = 2*(A.T*A*theta - A.T*y)
    theta = theta - alpha*df

print (theta)
```



Linear Basis Function Models

Function Approximation

- Select coefficients among a well-defined function (basis) that closely matches a target function in a task-specific way

Construct Explicit Feature Vectors

- Consider linear combinations of fixed nonlinear functions
  - Polynomial
  - Radial Basis Function (RBF)

$$\hat{y} = \sum_{i=0}^{d} \theta_i b_i(x) = \Phi\theta$$

$$\Phi = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & & \\ 1 & x_m & x_m^2 \end{bmatrix} \implies \hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_m \end{bmatrix} = \Phi\theta$$

Regression with Polynomial fitting

- Polynomial (here, quad is used as an example)

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \text{noise}$$

Different perspective:
- Approximate a target function as a linear combination of basis

$$\phi(x_i) = \begin{bmatrix} 1 \\ x_i \\ x_i^2 \end{bmatrix}$$

$$\hat{y} = \sum_{i=0}^{d} \theta_i b_i(x) = \Phi\theta$$

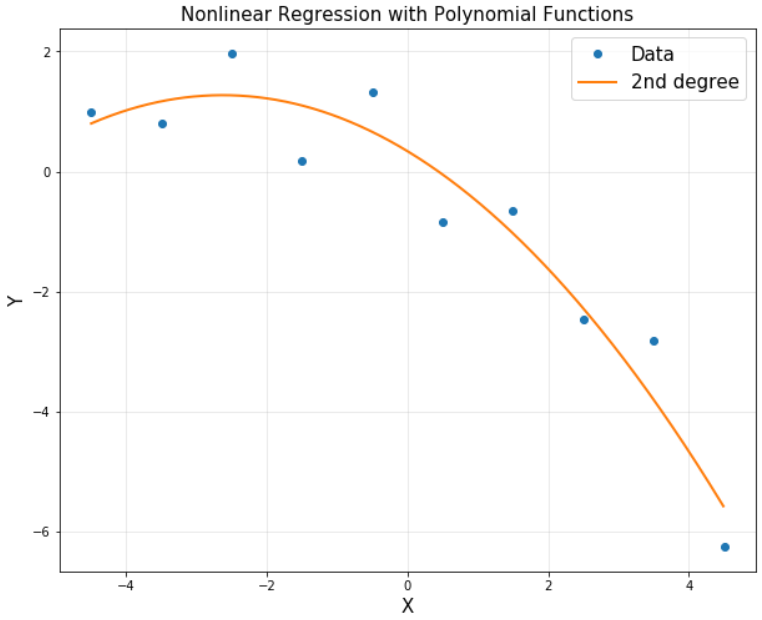
$$\Phi = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & & \\ 1 & x_m & x_m^2 \end{bmatrix} \implies \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \\ 1 & x_m & x_m^2 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} \Rightarrow \begin{bmatrix} | & | & | \\ b_0(x) & b_1(x) & b_2(x) \\ | & | & | \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$$

$$\implies \theta^* = (\Phi^T \Phi)^{-1} \Phi^T y$$

Regression with Polynomial fitting

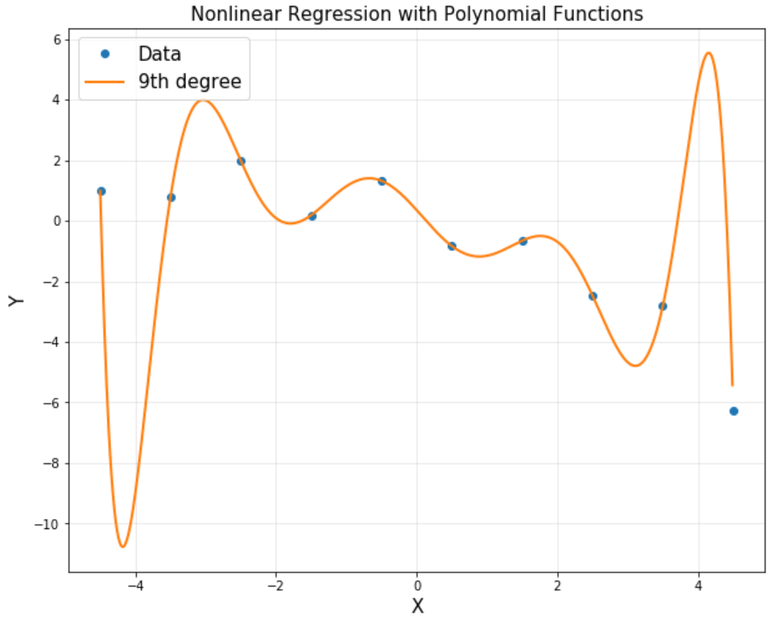
```
fp1 = np.polyfit(x[:, 0], y[:, 0], deg = 2)
f1 = np.poly1d(fp1)

print(fp1)
```

```
[-0.13504129 -0.71070424  0.33669063]
```

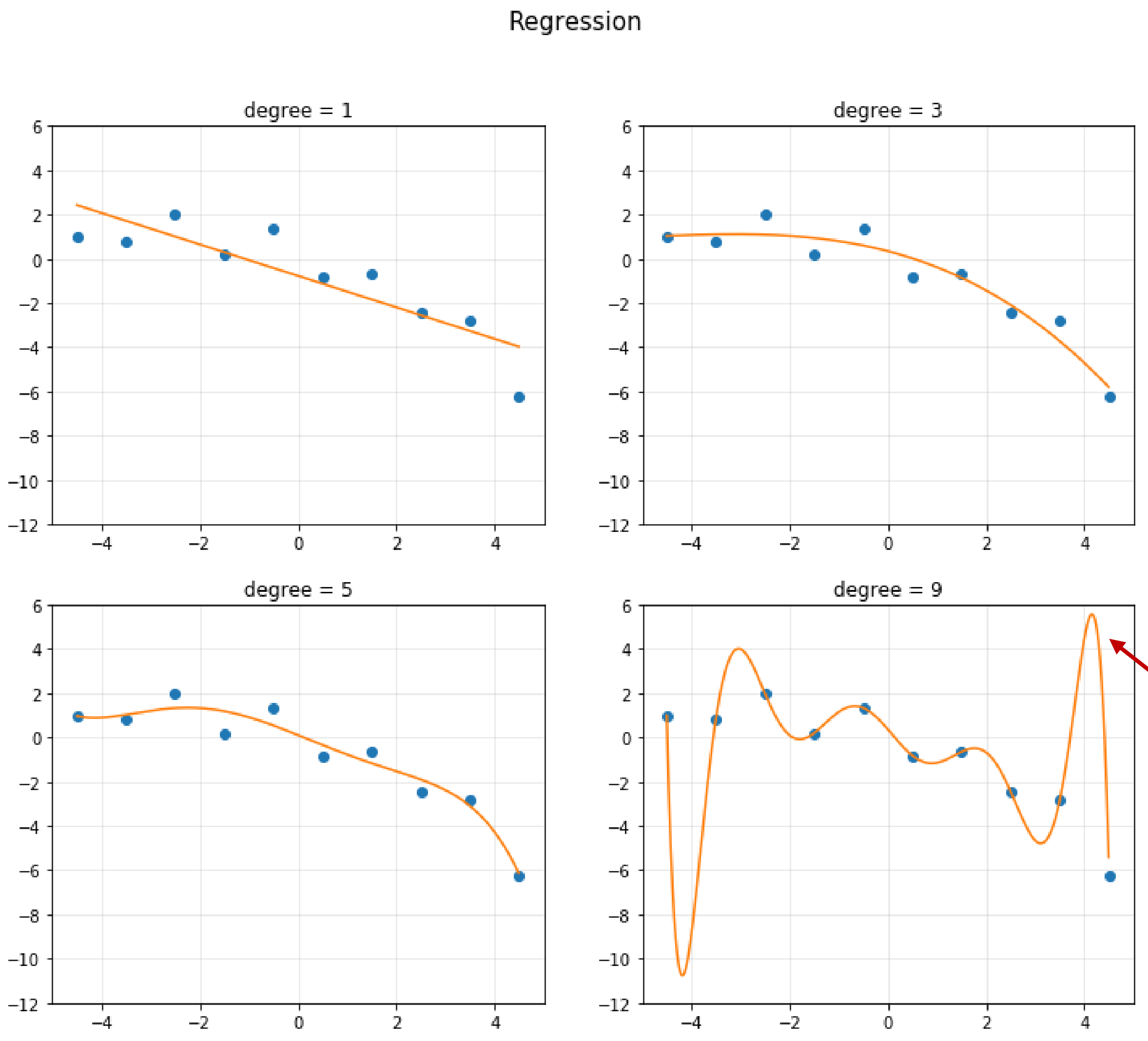



```
fp1 = np.polyfit(x[:, 0], y[:, 0], deg = 9)
f1 = np.poly1d(fp1)
```

10 input points with degree 9 (or 10)



Polynomial Fitting with Different Degrees

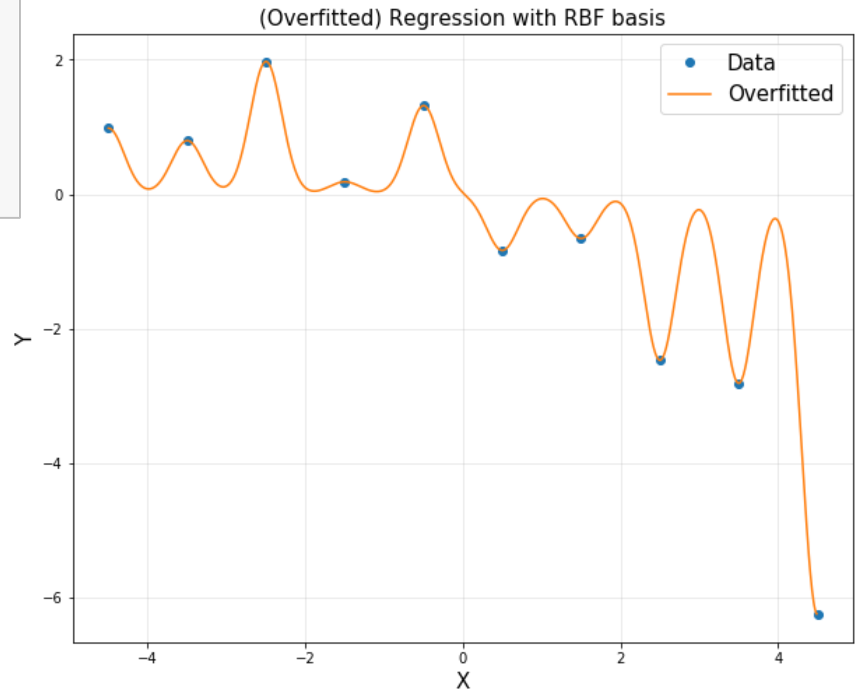


Low error on input data points, but high error nearby

Regression with RBF basis

```
xp = np.arange(-4.5, 4.5, 0.01).reshape(-1, 1)

d = 10
u = np.linspace(-4.5, 4.5, d)
sigma = 0.2

A = np.hstack([np.exp(-(x-u[i])**2/(2*sigma**2)) for i in range(d)])
rbfbasis = np.hstack([np.exp(-(xp-u[i])**2/(2*sigma**2)) for i in range(d)])

A = np.asmatrix(A)
rbfbasis = np.asmatrix(rbfbasis)

theta = (A.T*A).I*A.T*y
yp = rbfbasis*theta
```

$$\boxed{\theta = (A^T A)^{-1} A^T y}$$

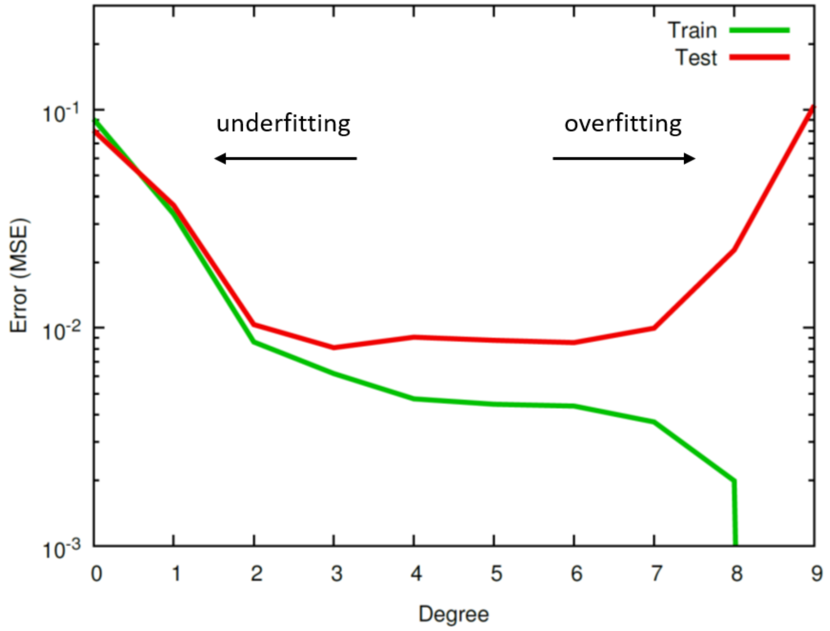


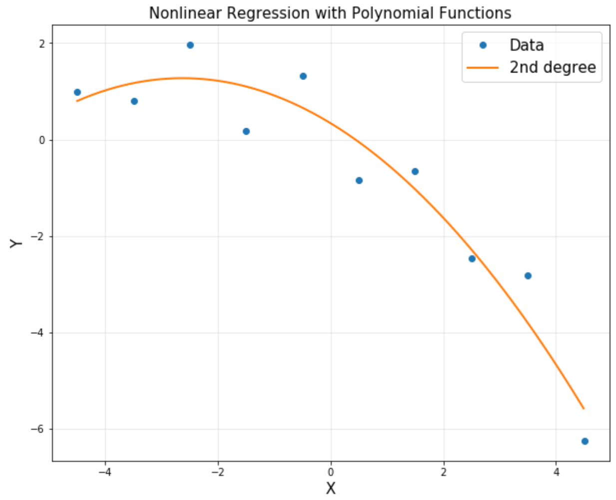
- With many features, our prediction function becomes very expensive
- Can lead to overfitting

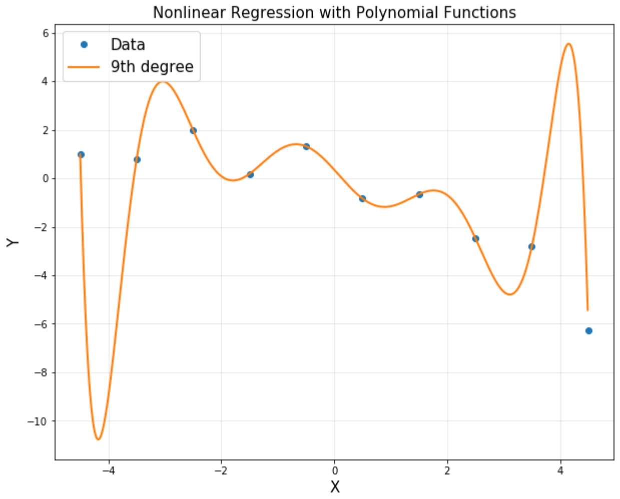
Regularization

Issue with Rich Representation

- Low error on input data points, but high error nearby
- Low error on training data, but high error on testing data







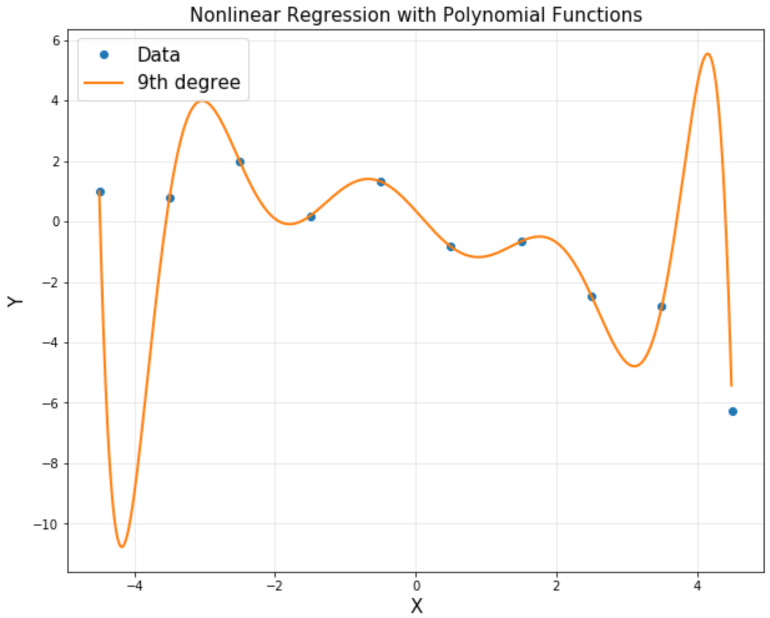
Generalization Error

- Fundamental problem: we are optimizing parameters to solve

$$\min_{\theta} \sum_{i=1}^{m} \ell(y_i, \hat{y}_i) = \min_{\theta} \sum_{i=1}^{m} \ell(y_i, \Phi\theta)$$

- But what we really care about is loss of prediction on new data (x, y)
  – also called generalization error

- Divide data into training set, and validation (testing) set



Representational Difficulties

- With many features, prediction function becomes very expressive (model complexity)
  - Choose less expensive function (e.g., lower degree polynomial, fewer RBF centers, larger RBF bandwidth)
  - Keep the magnitude of the parameter small
  - Regularization: penalize large parameters θ

$$\min \|\Phi\theta - y\|_2^2 + \lambda\|\theta\|_2^2$$

  - λ: regularization parameter, trades off between low loss and small values of θ

Regularization

- Often, overfitting associated with very large estimated parameters
- We want to balance
  - how well function fits data
  - magnitude of coefficients

$$\text{Total cost} = \underbrace{\text{measure of fit}}_{RSS(\theta)} + \ \lambda \cdot \underbrace{\text{measure of magnitude of coefficients}}_{\lambda \cdot ||\theta||_2^2}$$

$$\implies \min \ ||\Phi\theta - y||_2^2 + \lambda||\theta||_2^2$$

  - multi-objective optimization
  - λ is a tuning parameter

Regularization (Shrinkage Methods)

- the second term, $\lambda \cdot ||\theta||_2^2$, called a shrinkage penalty, is small when $\theta_1, \cdots, \theta_d$ are close to zeros, and so it has the effect of shrinking the estimates of θ_j towards zero

- the tuning parameter λ serves to control the relative impact of these two terms on the regression coefficient estimates

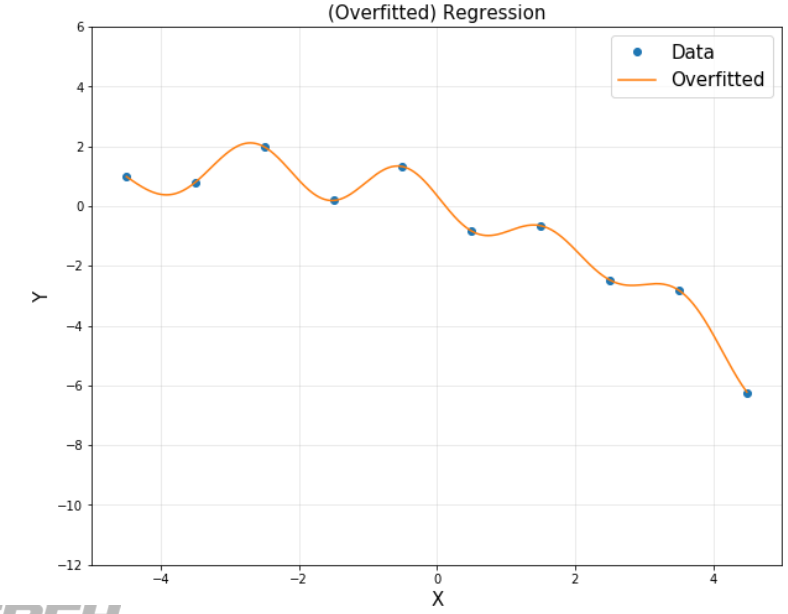
- known as a *ridge regression*

Ridge Regularization

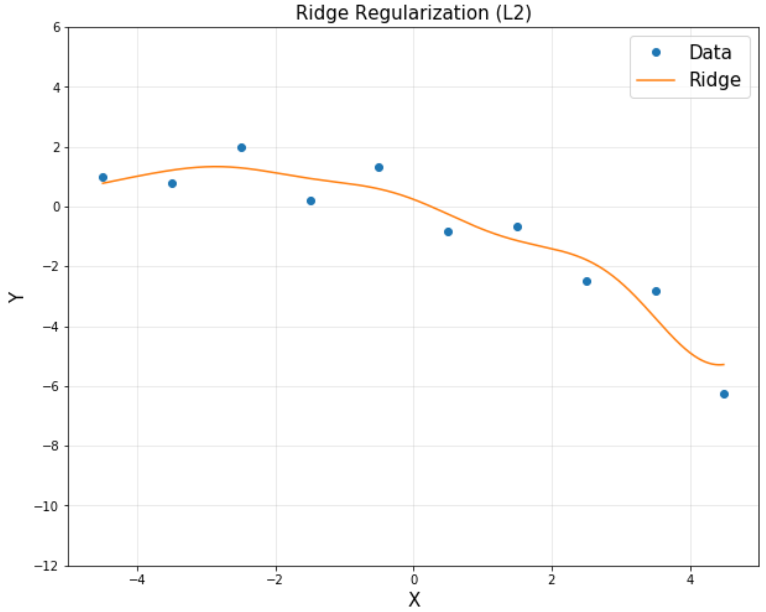
- Start from rich representation. Then, regularize coefficients θ

$$\text{Total cost} = \underbrace{\text{measure of fit}}_{RSS(\theta)} + \lambda \cdot \underbrace{\text{measure of magnitude of coefficients}}_{\lambda \cdot \|\theta\|_2^2}$$

$$\Longrightarrow \min \|\Phi\theta - y\|_2^2 + \lambda\|\theta\|_2^2$$



$\Longrightarrow$



Let's Use L_1 Norm

- Ridge regression

$$\text{Total cost} = \underbrace{\text{measure of fit}}_{RSS(\theta)} + \ \lambda \cdot \underbrace{\text{measure of magnitude of coefficients}}_{\lambda \cdot ||\theta||_2^2}$$

$$\Longrightarrow \ \min \ ||\Phi\theta - y||_2^2 + \boxed{\lambda||\theta||_2^2}$$

- Try this loss instead of ridge...

$$\text{Total cost} = \underbrace{\text{measure of fit}}_{RSS(\theta)} + \ \lambda \cdot \underbrace{\text{measure of magnitude of coefficients}}_{\lambda \cdot ||\theta||_1}$$

$$\Longrightarrow \ \min \ ||\Phi\theta - y||_2^2 + \boxed{\lambda||\theta||_1}$$

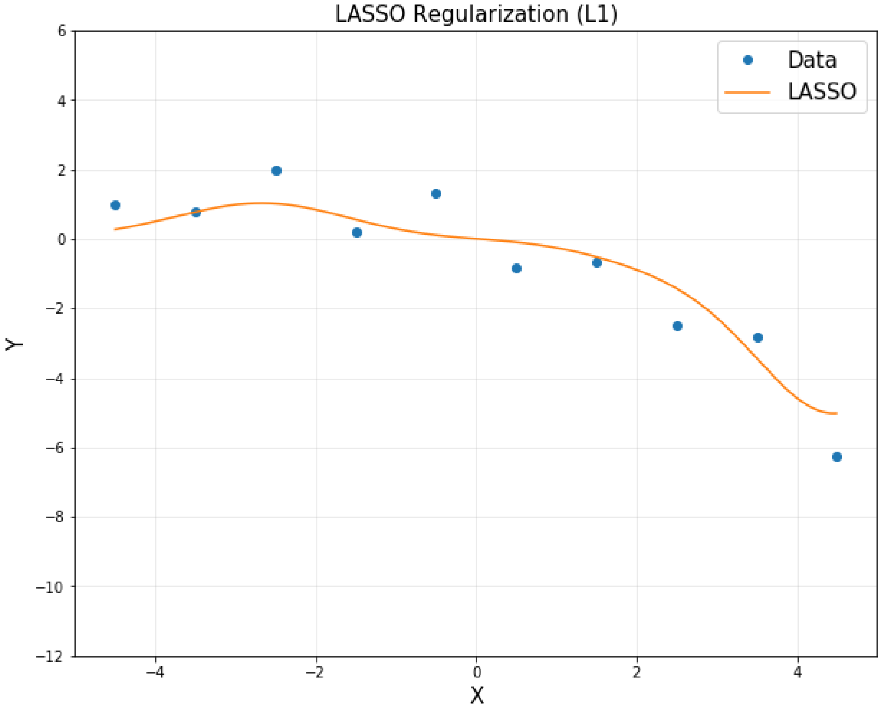
- λ is a tuning parameter = balance of fit and sparsity
- Known as *LASSO*
  - least absolute shrinkage and selection operator

LASSO Regularization

$$\text{Total cost} = \underbrace{\text{measure of fit}}_{RSS(\theta)} + \; \lambda \cdot \underbrace{\text{measure of magnitude of coefficients}}_{\lambda \cdot \|\theta\|_1}$$

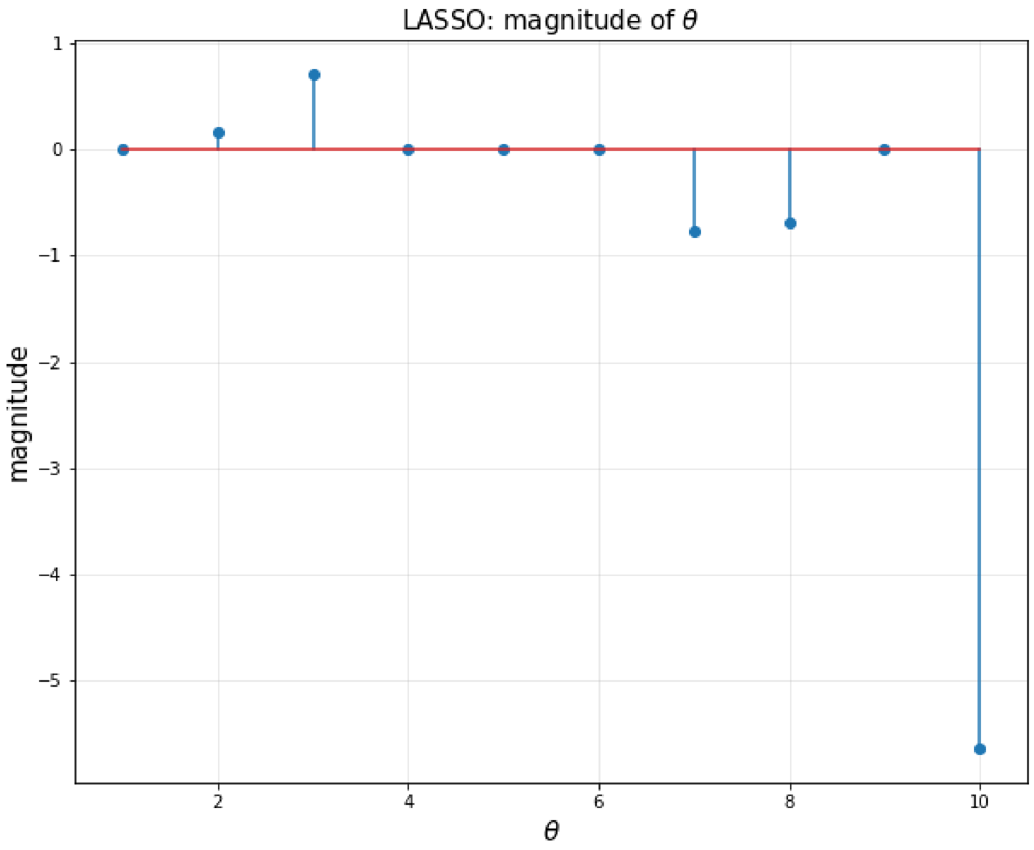
$$\implies \min \; \|\Phi\theta - y\|_2^2 + \lambda\|\theta\|_1$$

- Approximated function looks similar to that of ridge regression

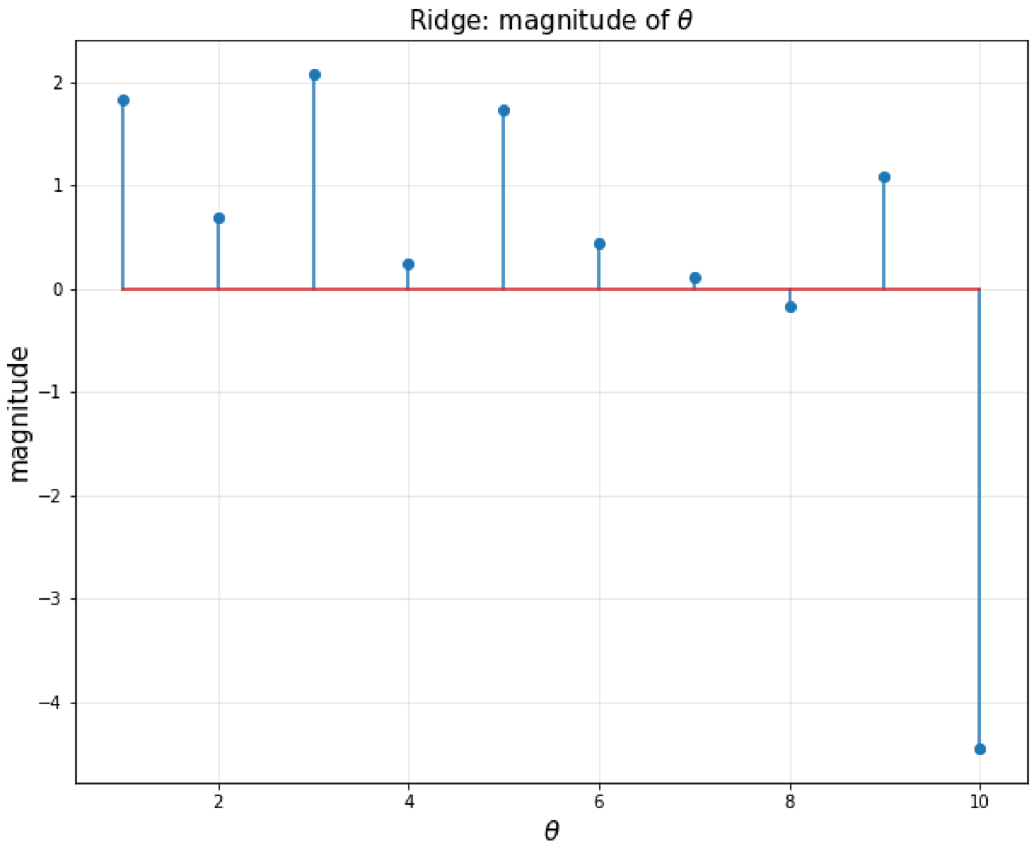


Coefficients θ with LASSO

- Non-zero coefficients indicate 'selected' features



LASSO



Ridge

LASSO vs. Ridge

- Another equivalent forms of optimizations

$$\min \ \|\Phi\theta - y\|_2^2 + \lambda\|\theta\|_1 \qquad\qquad \min \ \|\Phi\theta - y\|_2^2 + \lambda\|\theta\|_2^2$$

$$\Longrightarrow \quad \begin{aligned} \min_{\theta} \quad & \|\Phi\theta - y\|_2^2 \\ \text{subject to} \quad & \|\theta\|_1 \le s_1 \end{aligned} \qquad\qquad \begin{aligned} \min\nolimits_{\theta} \quad & \|\Phi\theta - y\|_2^2 \\ \text{subject to} \quad & \|\theta\|_2 \le s_2 \end{aligned}$$