Regression 2

Industrial AI Lab.
Changyun Choi / Iljeok Kim

Linear Regression: Advanced

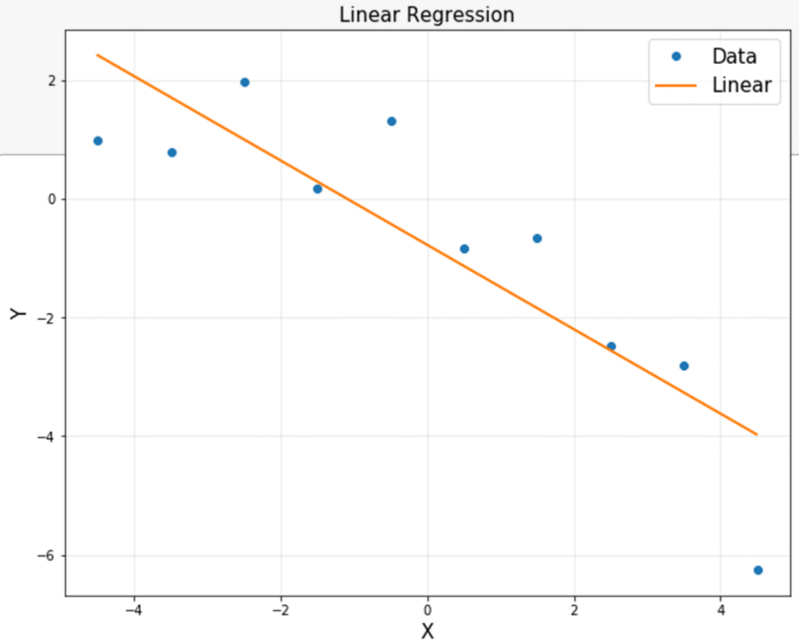
- Overfitting
- Regularization (Ridge and Lasso)

Overfitting: Start with Linear Regression

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# 10 data points
n = 10
x = np.linspace(-4.5, 4.5, 10).reshape(-1, 1)
y = np.array([0.9819, 0.7973, 1.9737, 0.1838, 1.3180, -0.8361, -0.6591, -2.4701, -2.8122, -6.2512]).reshape(-1, 1)

plt.figure(figsize=(10, 8))
plt.plot(x, y, 'o', label = 'Data')
plt.xlabel('X', fontsize = 15)
plt.ylabel('Y', fontsize = 15)
plt.grid(alpha = 0.3)
plt.show()
```

```
A = np.hstack([x**0, x])
A = np.asmatrix(A)

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

```
[[-0.7774    ]
 [-0.71070424]]
```



Recap: Nonlinear Regression

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

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

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

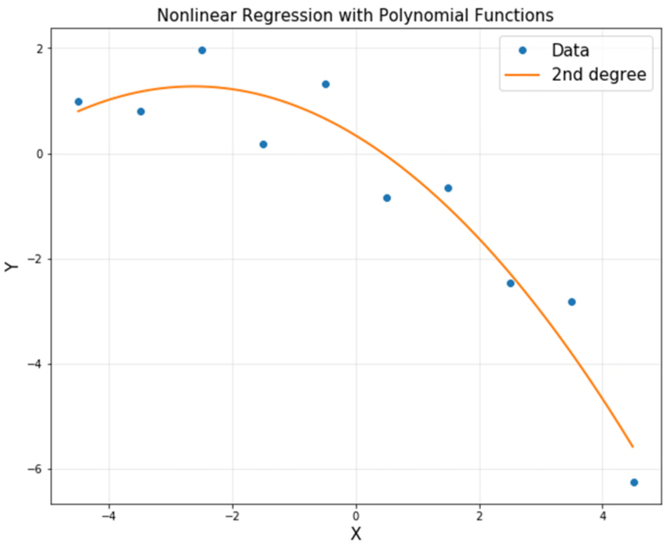
$$\Phi = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & & \\ 1 & x_m & x_m^2 \end{bmatrix} \implies \hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_m \end{bmatrix} = \Phi\theta$$

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

Nonlinear Regression

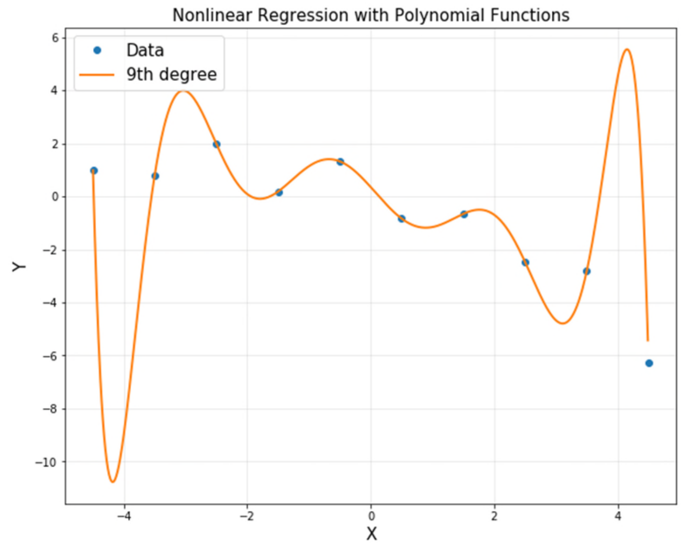
```python
A = np.hstack([x**0, x, x**2])
A = np.asmatrix(A)

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

```
[[ 0.33669062]
 [-0.71070424]
 [-0.13504129]]
```

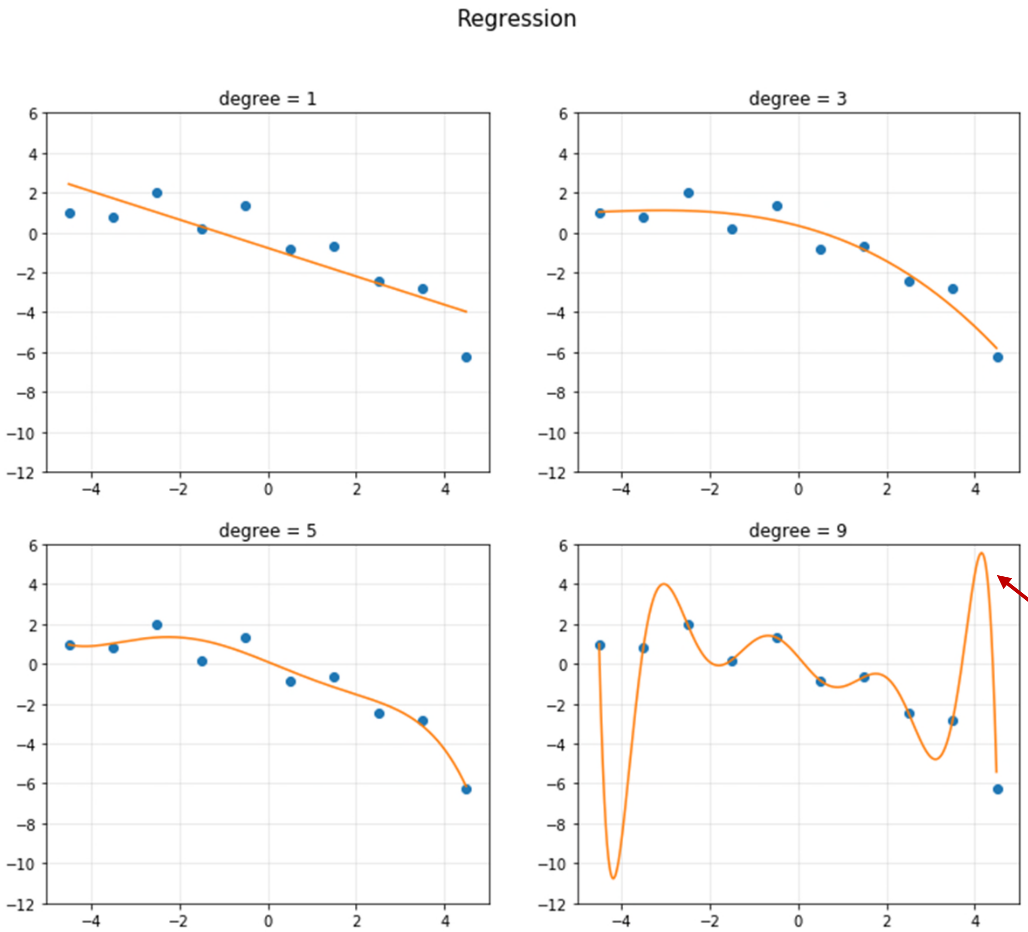



```python
A = np.hstack([x**i for i in range(10)])
A = np.asmatrix(A)

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

10 input points with degree 9 (or 10)



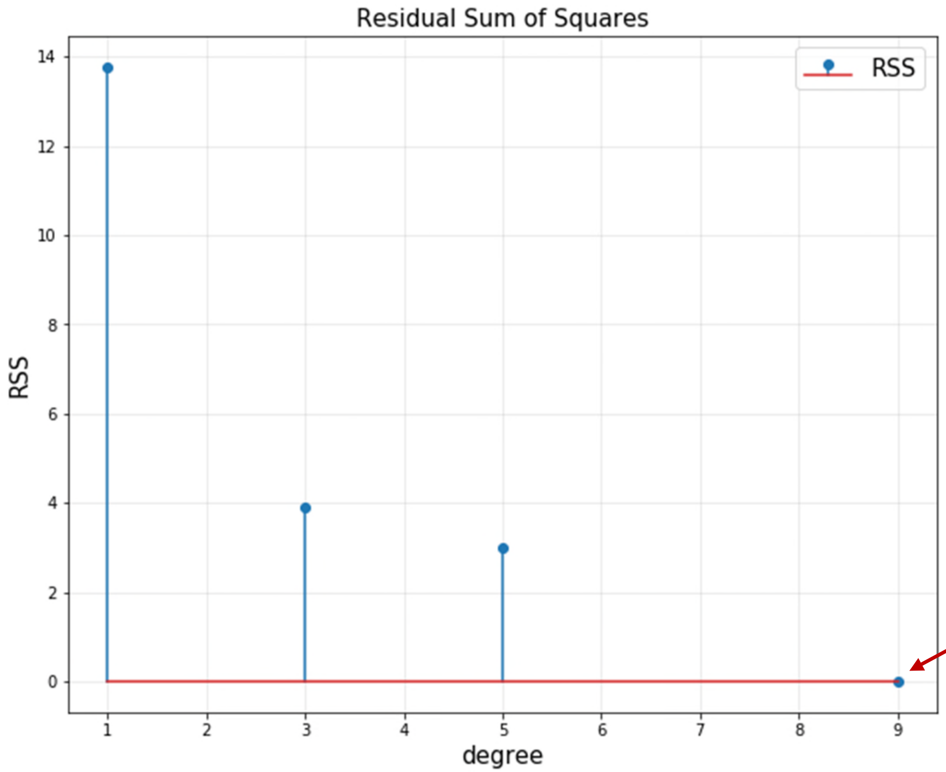
Polynomial Fitting with Different Degrees



Low error on input data points,
but high error nearby

Loss

- Loss: Residual Sum of Squares (RSS)



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

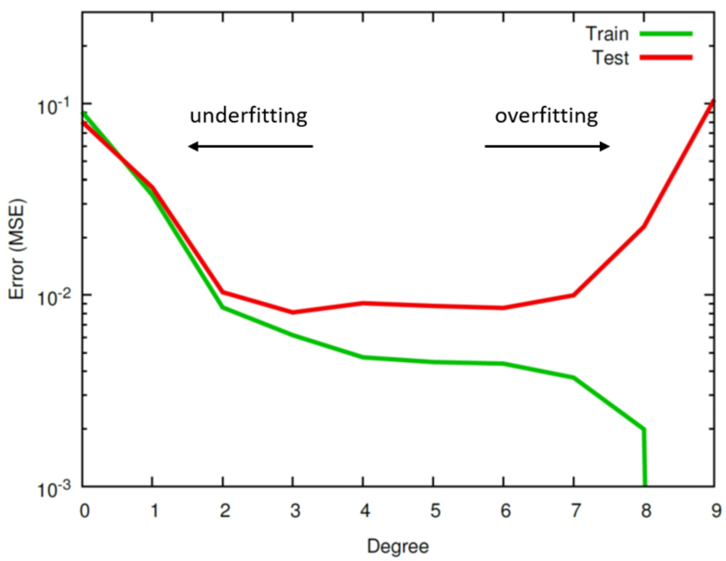
Minimizing loss in training data is often not the best

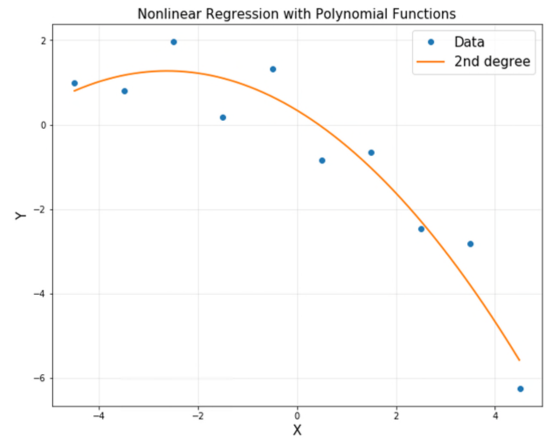
↑

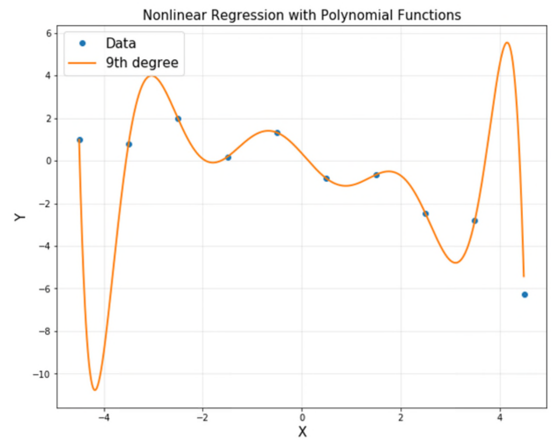
Low error on input data points, but high error nearby

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



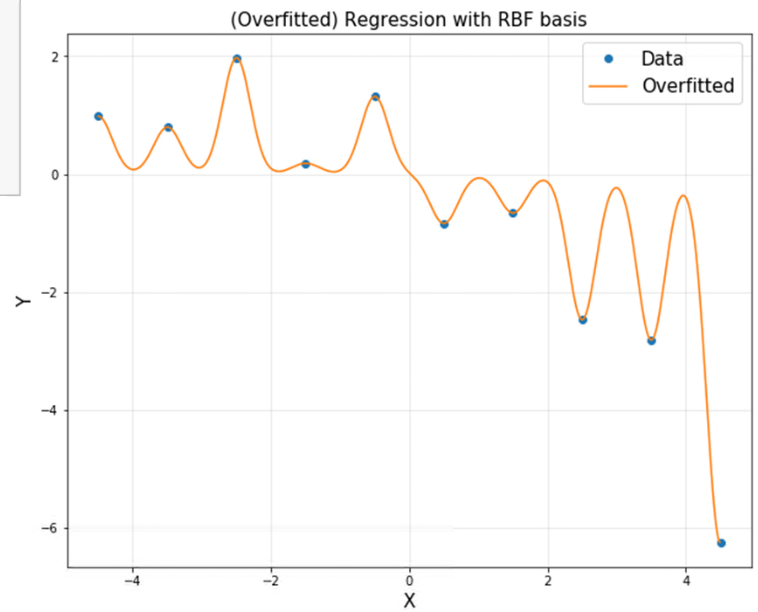




Linear Regression with RBF

```
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
```

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

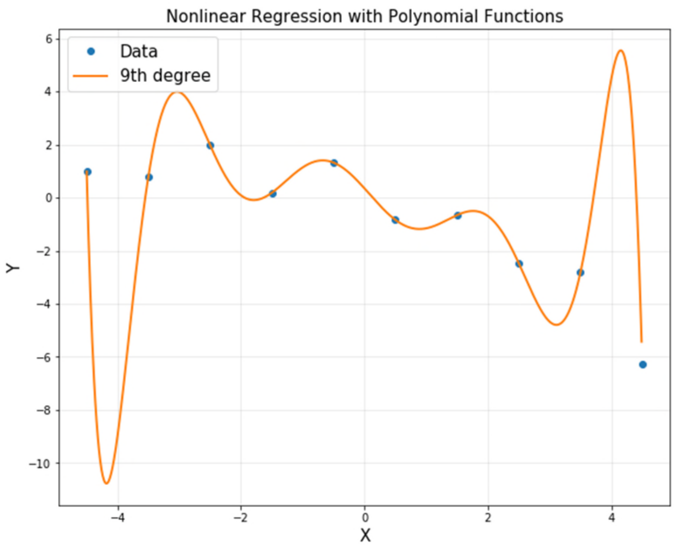


- With many features, our prediction function becomes very expensive
- Can lead to overfitting

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



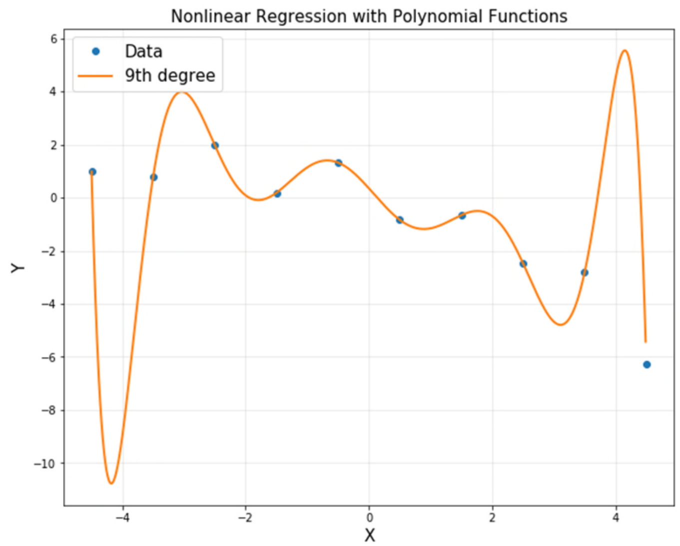
Generalization Error

- 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 expressive function (e.g., lower degree polynomial, fewer RBF centers, larger RBF bandwidth)
  - Keep the magnitude of the parameter small
  - Regularization: penalize large parameters θ

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

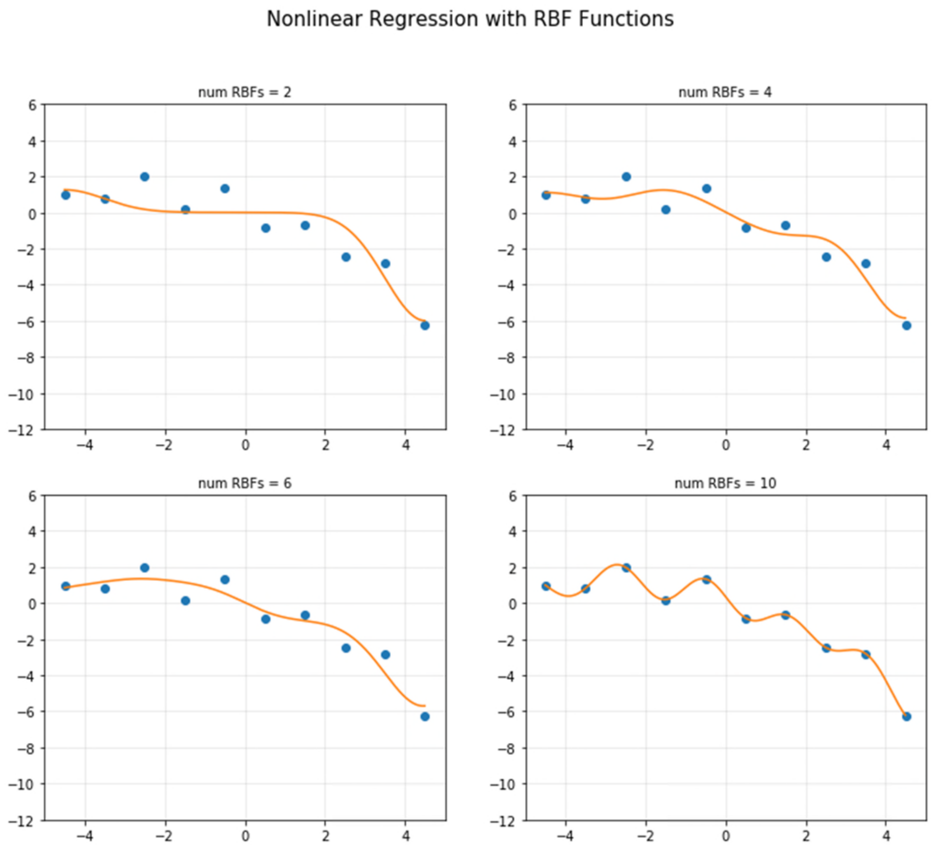
  - λ: regularization parameter, trades off between low loss and small values of θ

With Less Basis Functions: Fewer RBF Centers

```
d = [2, 4, 6, 10]
sigma = 1

plt.figure(figsize=(12, 10))

for k in range(4):
    u = np.linspace(-4.5, 4.5, d[k])

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

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

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

    plt.subplot(2, 2, k+1)
    plt.plot(x, y, 'o')
    plt.plot(xp, yp)
    plt.axis([-5, 5, -12, 6])
    plt.title('num RBFs = {}'.format(d[k]), fontsize = 10)
    plt.grid(alpha = 0.3)
```

With Less Basis Functions: Fewer RBF Centers

- Least-squares fits for different numbers of RBFs



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 (Shrinkage Methods)

- 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

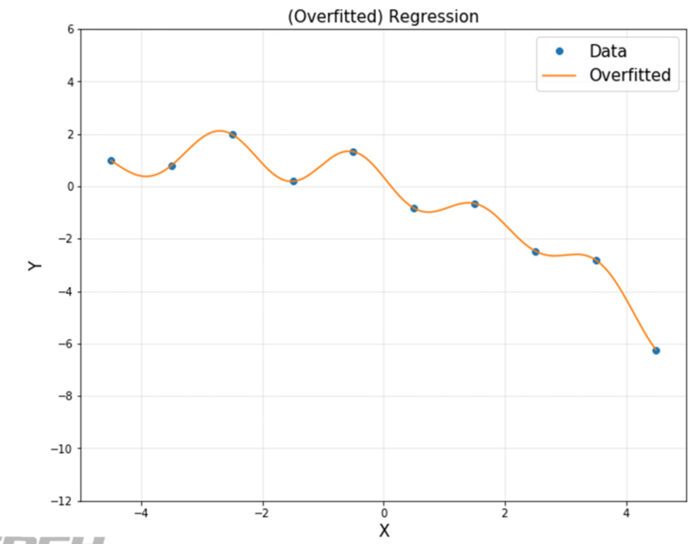
- the tuning parameter λ serves to control the relative impact of these two terms on the regression coefficient estimates

- known as a *ridge regression*

RBF: Start from Rich Representation

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

sigma = 1

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 = cvx.Variable([d, 1])
obj = cvx.Minimize(cvx.sum_squares(A*theta-y))
prob = cvx.Problem(obj).solve()

yp = rbfbasis*theta.value
```

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

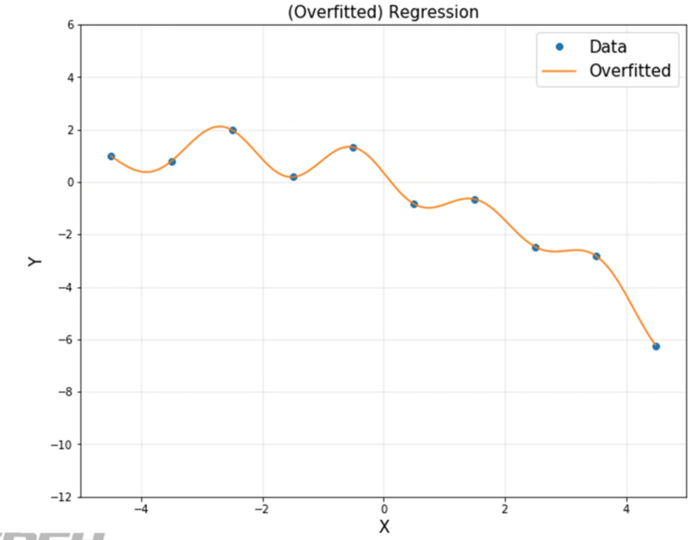


RBF with Regularization

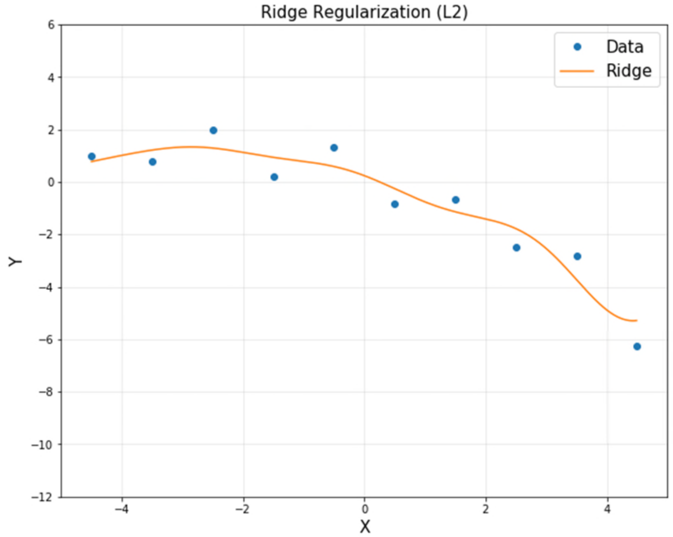
- Start from rich representation. Then, regularize coefficients θ

```
# ridge regression

lamb = 0.1
theta = cvx.Variable([d, 1])
obj = cvx.Minimize(cvx.sum_squares(A*theta - y) + lamb*cvx.sum_squares(theta))
prob = cvx.Problem(obj).solve()

yp = rbfbasis*theta.value
```

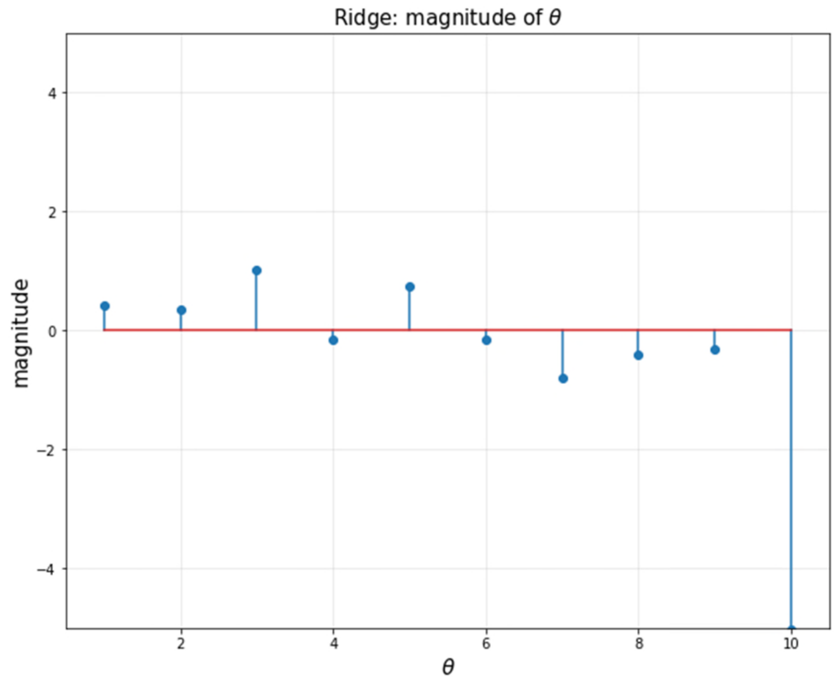
$$\min \|\Phi\theta - y\|_2^2 + \lambda\|\theta\|_2^2$$

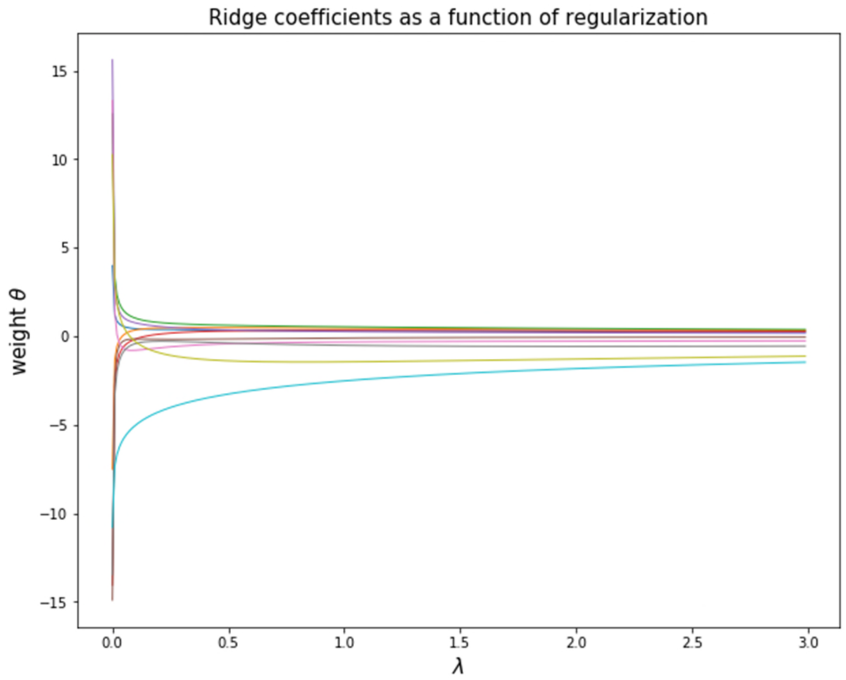


$$\Longrightarrow$$



Coefficients θ





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...

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

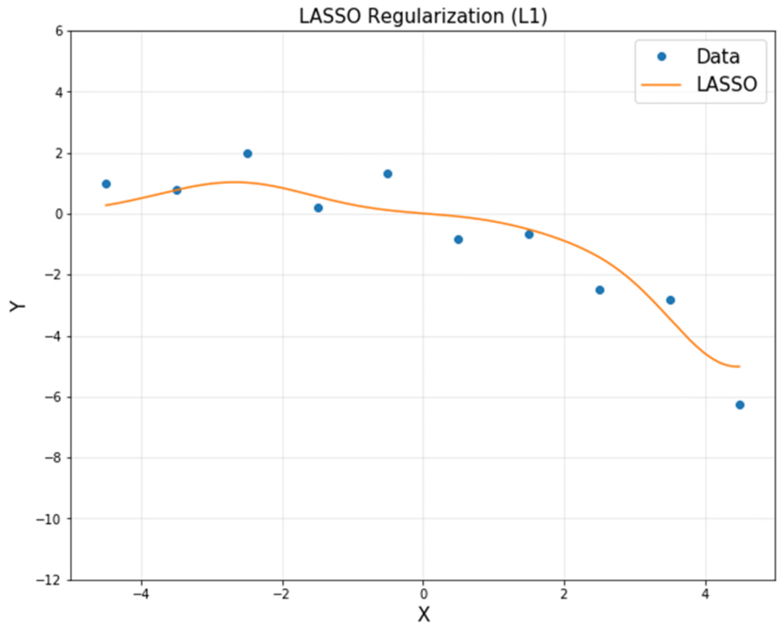
$$\Longrightarrow \min \|\Phi\theta - y\|_2^2 + \boxed{\lambda\|\theta\|_1}$$

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

RBF with LASSO

```
# LASSO regression

lamb = 2
theta = cvx.Variable([d, 1])
obj = cvx.Minimize(cvx.sum_squares(A*theta - y) + lamb*cvx.norm(theta, 1))
prob = cvx.Problem(obj).solve()

yp = rbfbasis*theta.value
```

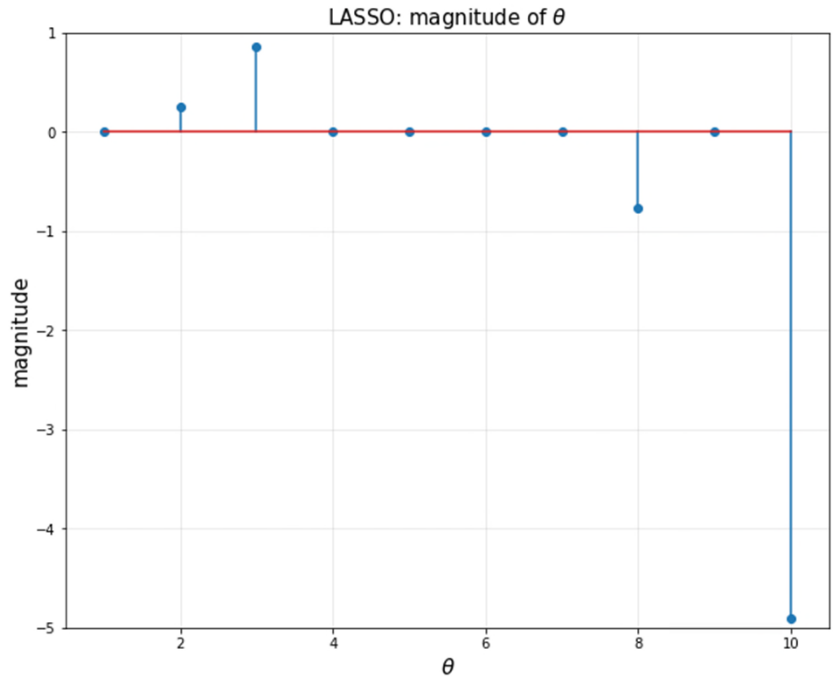
$$\min \|\Phi\theta - y\|_2^2 + \lambda\|\theta\|_1$$



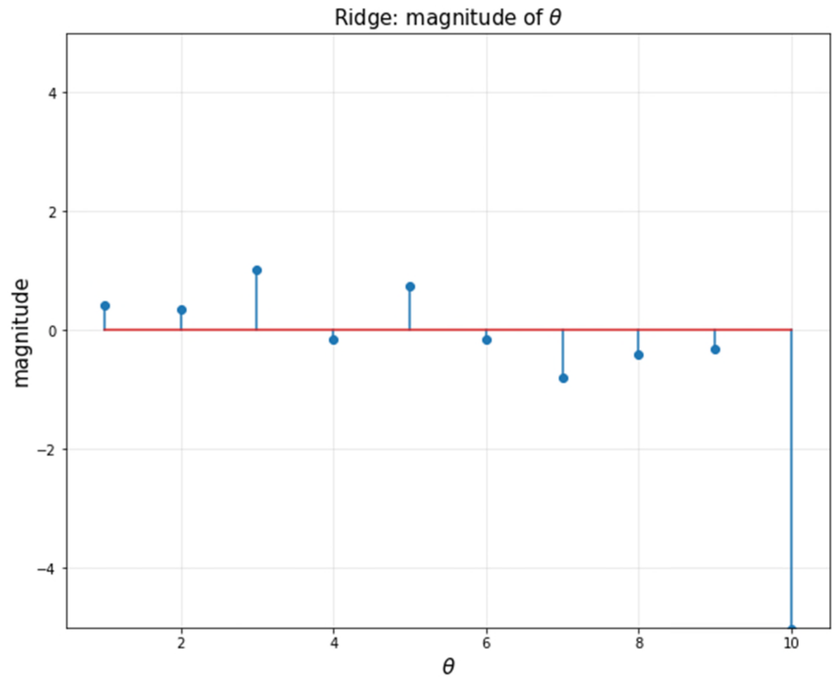
- Approximated function looks similar to that of ridge regression

Coefficients θ with LASSO

- Non-zero coefficients indicate ‘selected’ features



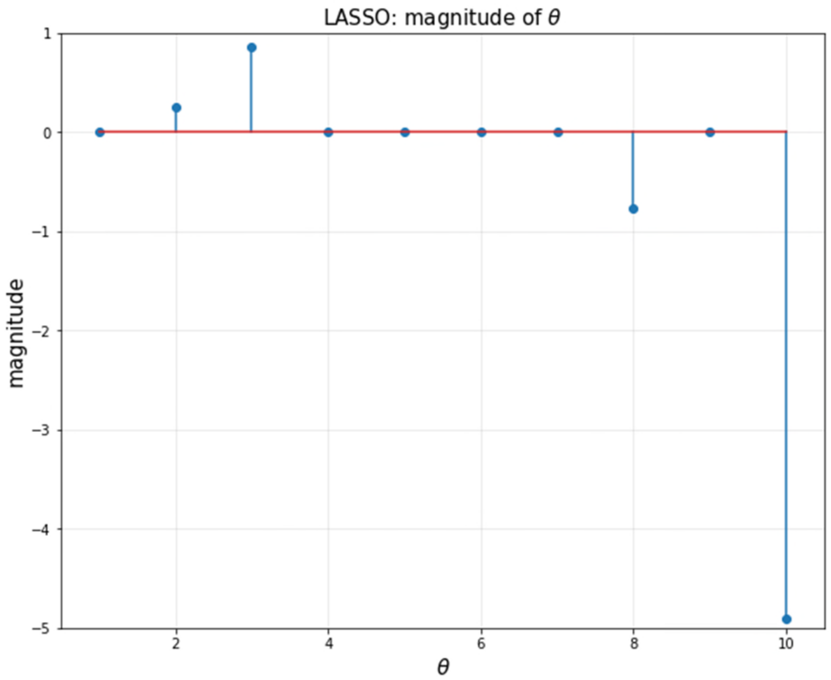
LASSO

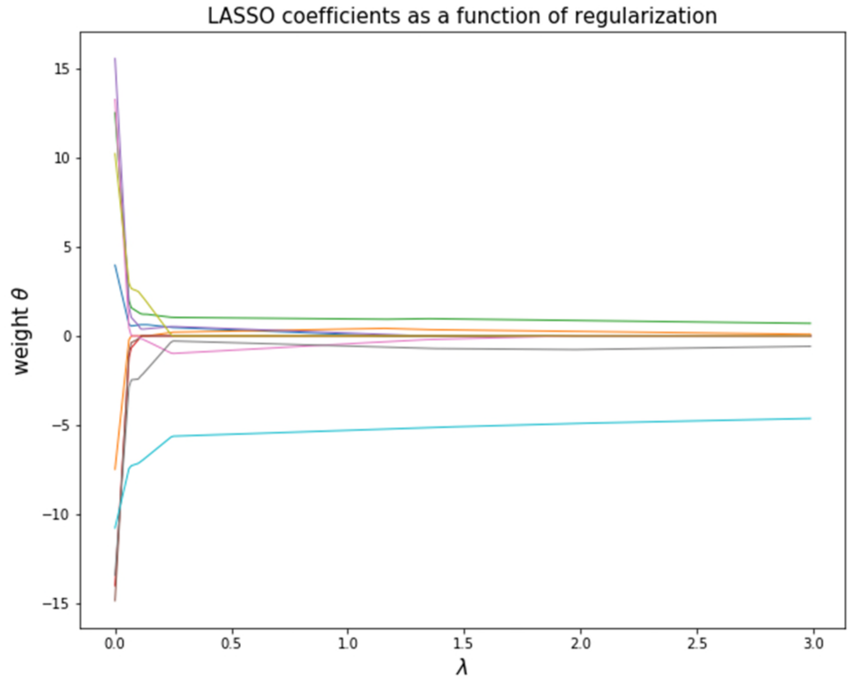


Ridge

Coefficients θ with LASSO

- Non-zero coefficients indicate 'selected' features



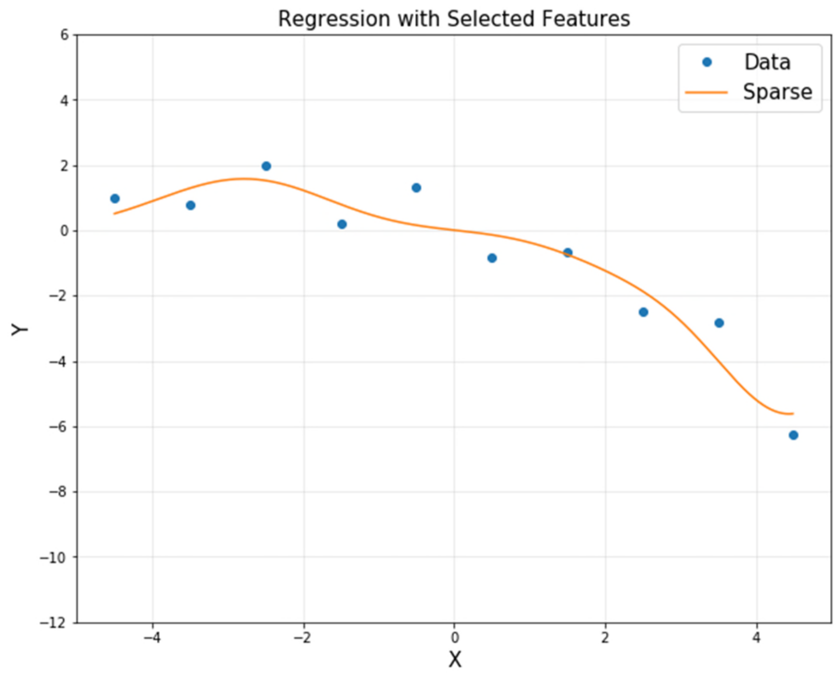


Sparsity for Feature Selection using Lasso

- Least squares with a penalty on the L_1 norm of the parameters
- Start with full model (all possible features)
- 'Shrink' some coefficients exactly to 0
  - *i.e.*, knock out certain features
  - The L_1 penalty has the effect of forcing some of the coefficient estimates to be exactly equal to zero
- Non-zero coefficients indicate 'selected' features

Regression with Selected Features

```
# reduced order model
# we will use only theta 2, 3, 8, 10

d = 4
u = np.array([-3.5, -2.5, 2.5, 4.5])
sigma = 1

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

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

theta = cvx.Variable([d, 1])
obj = cvx.Minimize(cvx.norm(A*theta-y, 2))
prob = cvx.Problem(obj).solve()

yp = rbfbasis*theta.value
```



LASSO vs. Ridge

- Another equivalent forms of optimizations

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

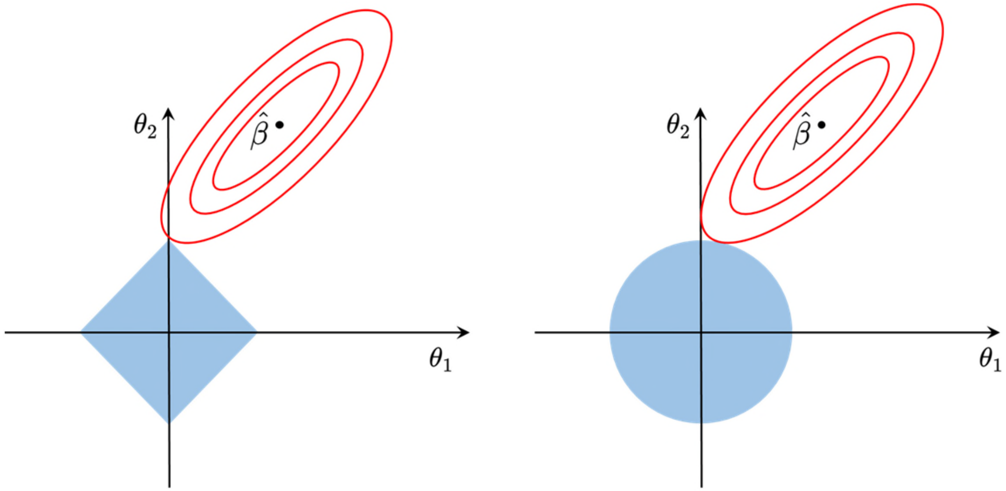
$$\Longrightarrow \qquad \begin{array}{rl} \min\limits_{\theta} & \|\Phi\theta - y\|_2^2 \\ \text{subject to} & \|\theta\|_1 \leq s_1 \end{array} \qquad\qquad \begin{array}{rl} \min_{\theta} & \|\Phi\theta - y\|_2^2 \\ \text{subject to} & \|\theta\|_2 \leq s_2 \end{array}$$

LASSO vs. Ridge

- Another equivalent forms of optimizations

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

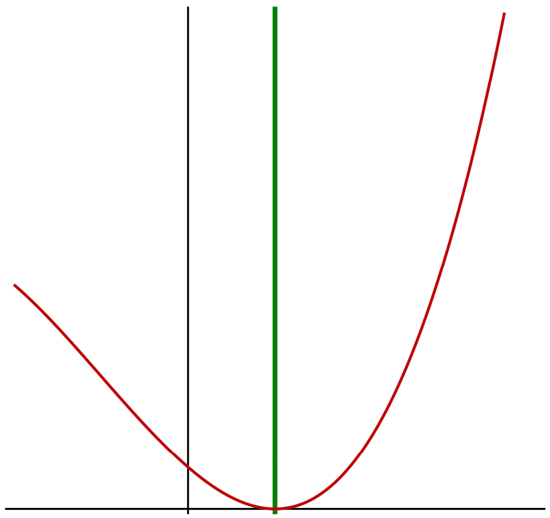
$$\Longrightarrow \quad \begin{array}{rl} \min\limits_{\theta} & \|\Phi\theta - y\|_2^2 \\ \text{subject to} & \|\theta\|_1 \le s_1 \end{array} \qquad\qquad \begin{array}{rl} \min_{\theta} & \|\Phi\theta - y\|_2^2 \\ \text{subject to} & \|\theta\|_2 \le s_2 \end{array}$$



L2 Regularizers: Simple Example

Increasing the λ parameter moves the optimal closer to 0, and away from the optimal for the loss alone.

Since the derivative of $\|x\|_2^2$ is zero at zero, the optimal will never move there if it was not already there.

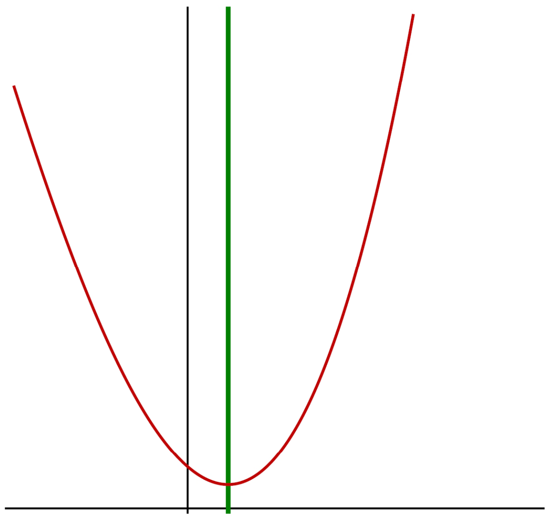


$(x-1)^2 + \frac{1}{6}(x-1)^3$

L2 Regularizers: Simple Example

Increasing the λ parameter moves the optimal closer to 0, and away from the optimal for the loss alone.

Since the derivative of $\|x\|_2^2$ is zero at zero, the optimal will never move there if it was not already there.

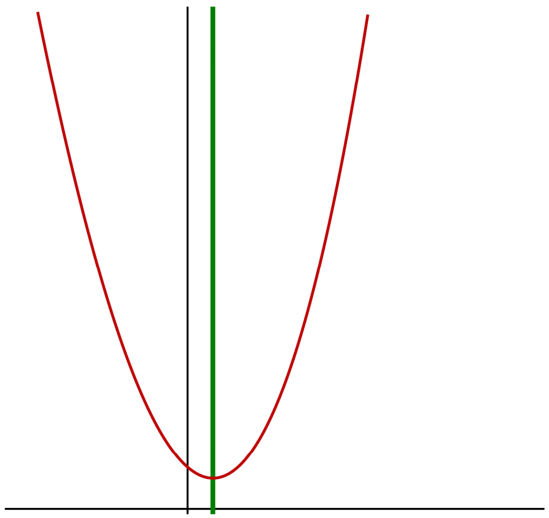


$(x-1)^2 + \frac{1}{6}(x-1)^3 + x^2$

L2 Regularizers: Simple Example

Increasing the λ parameter moves the optimal closer to 0, and away from the optimal for the loss alone.

Since the derivative of $\|x\|_2^2$ is zero at zero, the optimal will never move there if it was not already there.

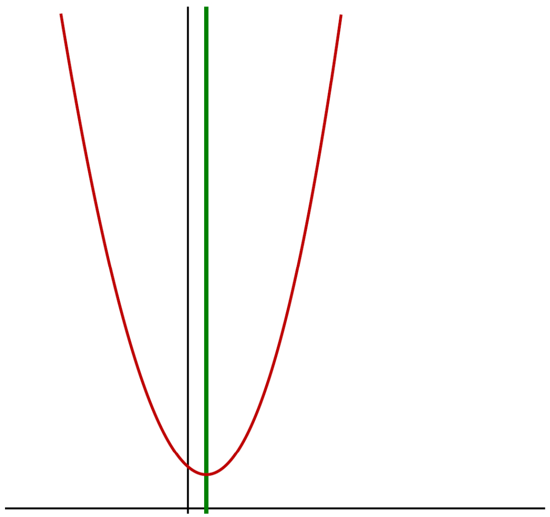


$(x-1)^2 + \frac{1}{6}(x-1)^3 + 2x^2$

L2 Regularizers: Simple Example

Increasing the λ parameter moves the optimal closer to 0, and away from the optimal for the loss alone.

Since the derivative of $\|x\|_2^2$ is zero at zero, the optimal will never move there if it was not already there.

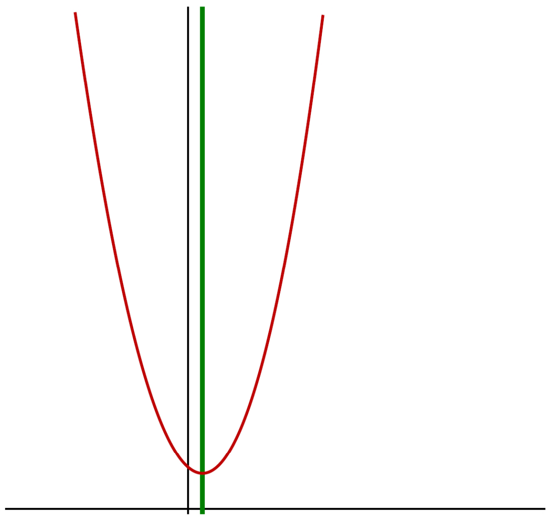


$(x-1)^2 + \frac{1}{6}(x-1)^3 + 3x^2$

L2 Regularizers: Simple Example

Increasing the λ parameter moves the optimal closer to 0, and away from the optimal for the loss alone.

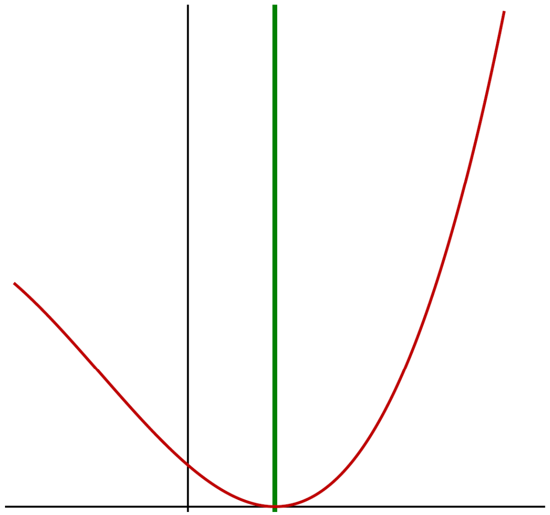
Since the derivative of $\|x\|_2^2$ is zero at zero, the optimal will never move there if it was not already there.



$(x-1)^2 + \frac{1}{6}(x-1)^3 + 4x^2$

L1 Regularizers: Simple Example

Increasing the λ parameter moves the optimal closer to 0, and away from the optimal for the loss without penalty.

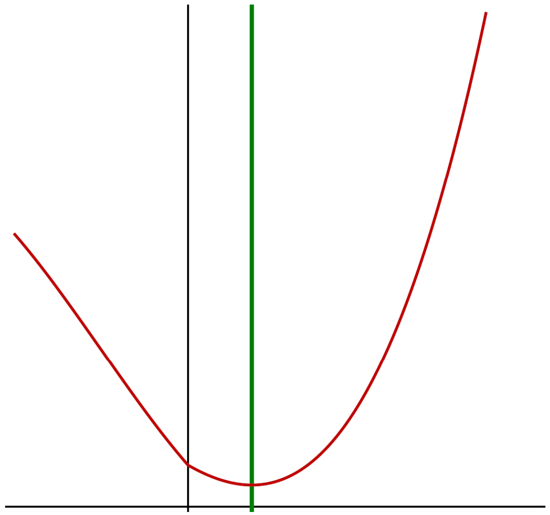


$$(x-1)^2 + \frac{1}{6}(x-1)^3$$

POSTECH

L1 Regularizers: Simple Example

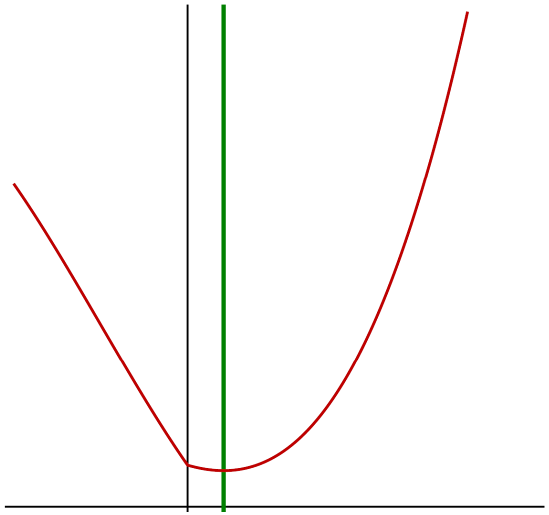
Increasing the λ parameter moves the optimal closer to 0, and away from the optimal for the loss without penalty.



$$(x-1)^2 + \frac{1}{6}(x-1)^3 + \frac{1}{2}|x|$$

L1 Regularizers: Simple Example

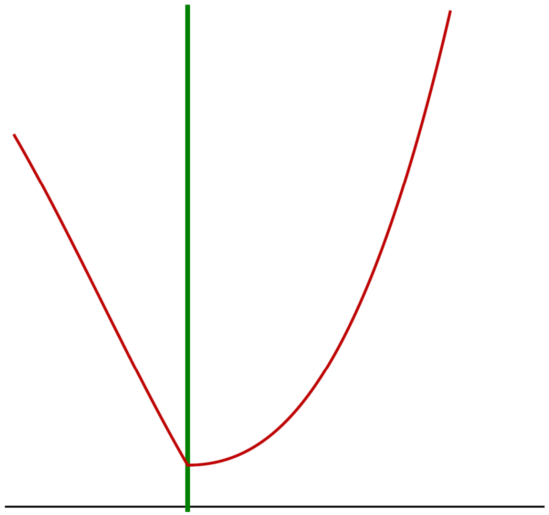
Increasing the λ parameter moves the optimal closer to 0, and away from the optimal for the loss without penalty.



$$(x-1)^2 + \frac{1}{6}(x-1)^3 + |x|$$

L1 Regularizers: Simple Example

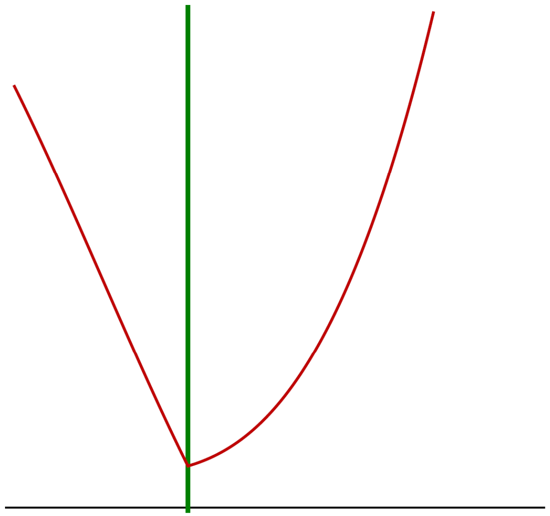
Increasing the λ parameter moves the optimal closer to 0, and away from the optimal for the loss without penalty.



$$(x-1)^2 + \frac{1}{6}(x-1)^3 + \frac{3}{2}|x|$$

L1 Regularizers: Simple Example

Increasing the λ parameter moves the optimal closer to 0, and away from the optimal for the loss without penalty.

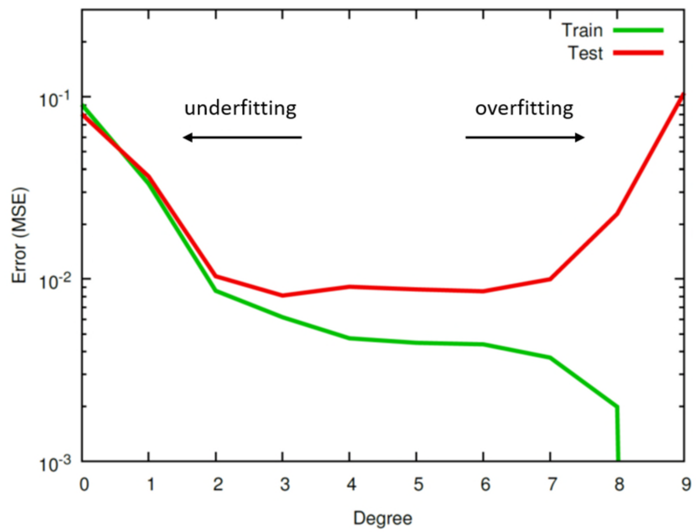


$$(x-1)^2 + \frac{1}{6}(x-1)^3 + 2|x|$$

Evaluation

- Adding more features will always decrease the loss
- How do we determine when an algorithm achieves "good" performance?

- A better criterion:
  - Training set (e.g., 70 %)
  - Testing set (e.g., 30 %)



- Performance on testing set called *generalization* performance