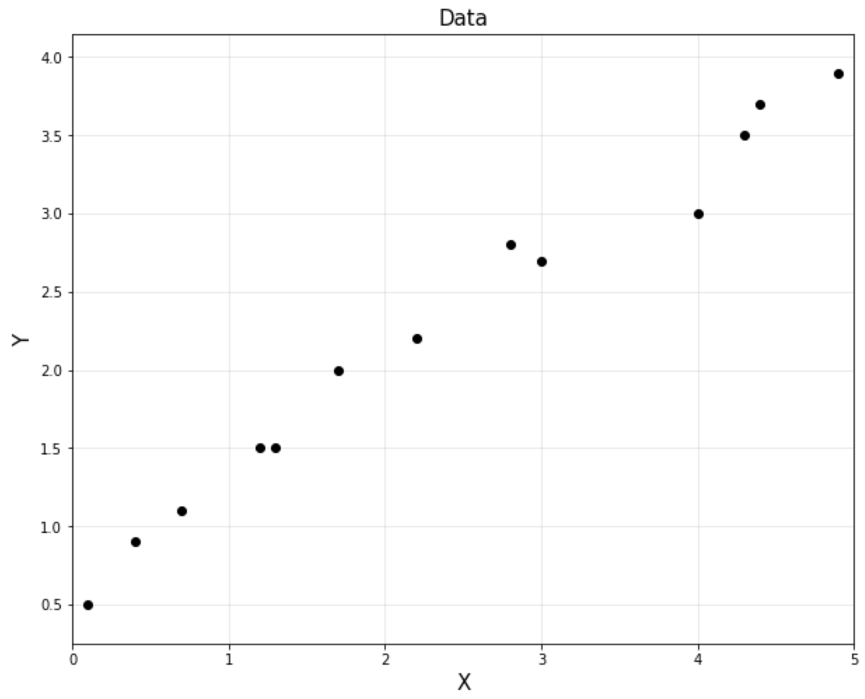
Regression 1

Industrial AI Lab.

Prof. Seungchul Lee

Assumption: Linear Model

$$\hat{y}_i = f(x_i\,;\theta) \text{ in general}$$

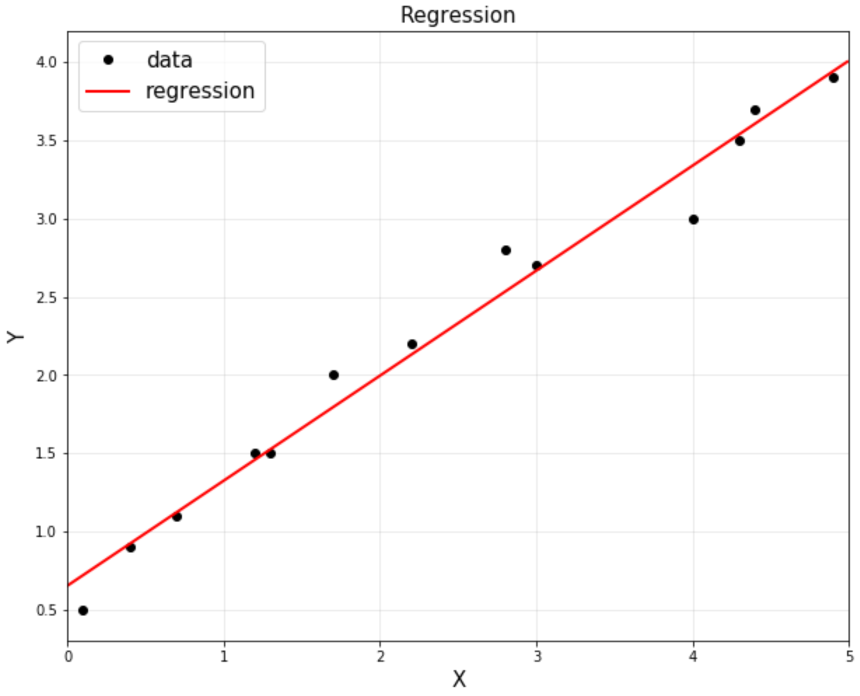


- In many cases, a linear model is used to predict y_i

$$\hat{y}_i = \theta_1 x_i + \theta_2$$

Assumption: Linear Model

$$\hat{y}_i = f(x_i\,;\theta) \ \text{ in general}$$



- In many cases, a linear model is used to predict y_i

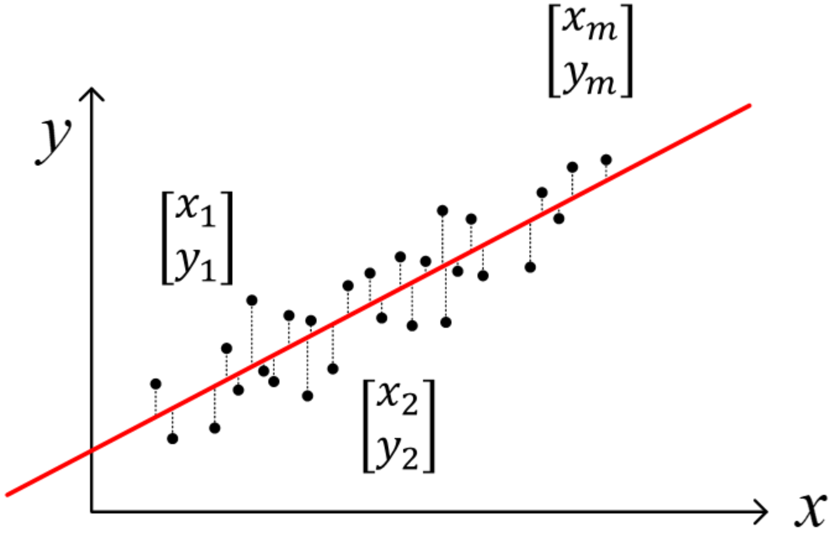
$$\hat{y}_i = \theta_1 x_i + \theta_2$$

Linear Regression

- $\hat{y}_i = f(x_i, \theta)$ in general
- In many cases, a linear model is assumed to predict y_i

Given $\begin{cases} x_i : \text{inputs} \\ y_i : \text{outputs} \end{cases}$, Find θ_0 and θ_1

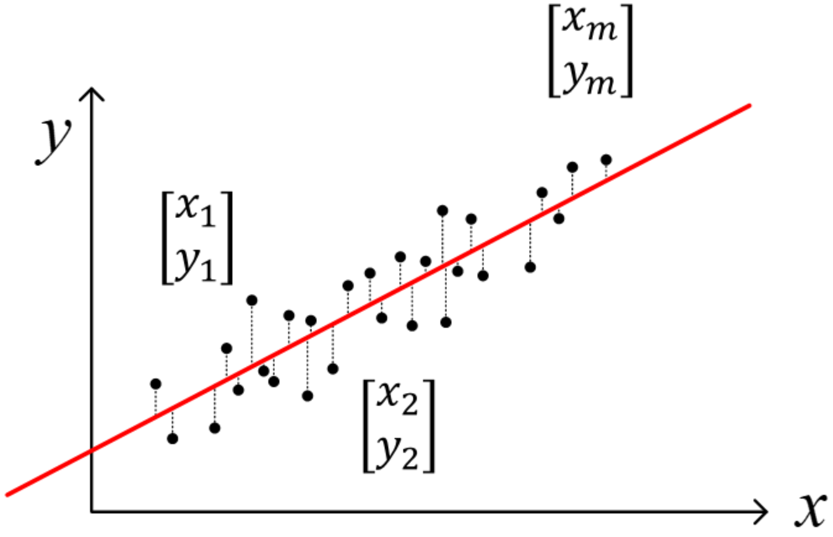
$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \approx \hat{y}_i = \theta_0 + \theta_1 x_i$$



- $\hat{y}_i$: predicted output
- $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$: model parameters

Linear Regression as Optimization

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \approx \hat{y}_i = \theta_0 + \theta_1 x_i$$



- How to find model parameters $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$
- Optimization problem

$$\hat{y}_i = \theta_0 + \theta_1 x_i \qquad \text{such that} \qquad \min_{\theta_0, \theta_1} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$

Re-cast Problem as Least Squares

- For convenience, we define a function that maps inputs to feature vectors, ϕ

$$\hat{y}_i = \theta_0 + x_i\theta_1 = 1 \cdot \theta_0 + x_i\theta_1$$

$$= \begin{bmatrix} 1 & x_i \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ x_i \end{bmatrix}^T \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

$$= \phi^T(x_i)\theta$$

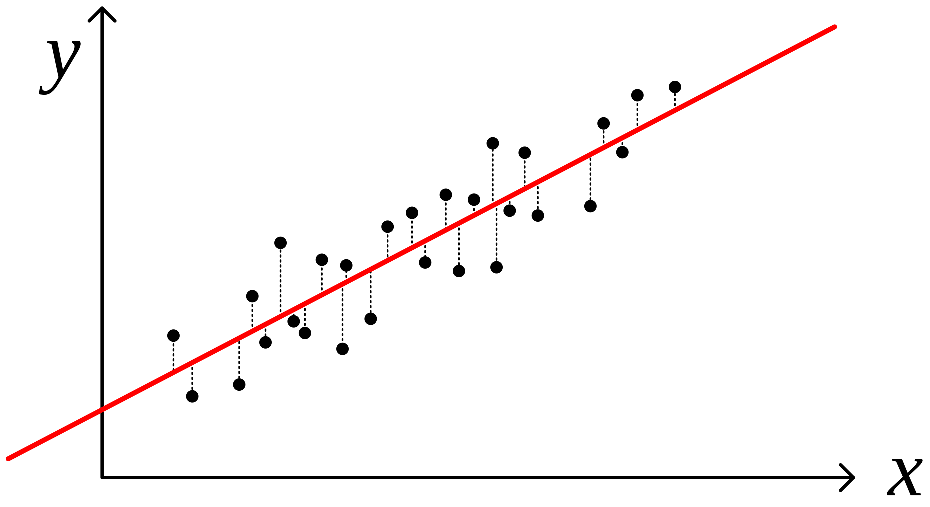
$$\text{feature vector } \phi(x_i) = \begin{bmatrix} 1 \\ x_i \end{bmatrix}$$

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

Optimization

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

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



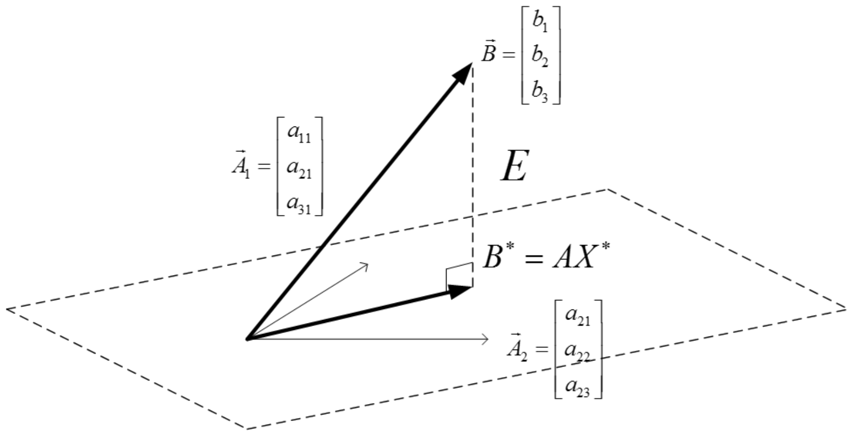
Optimization: Note

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

$$\begin{array}{cccccc} \left[\begin{matrix} 1 \\ 1 \\ \vdots \\ 1 \end{matrix}\right. & \left.\begin{matrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{matrix}\right] & \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} & = & \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \\ \uparrow & \uparrow & \uparrow & & \uparrow \\ \vec{A}_1 & \vec{A}_2 & \vec{x} & & \vec{B} \end{array}$$

over-determined or projection

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



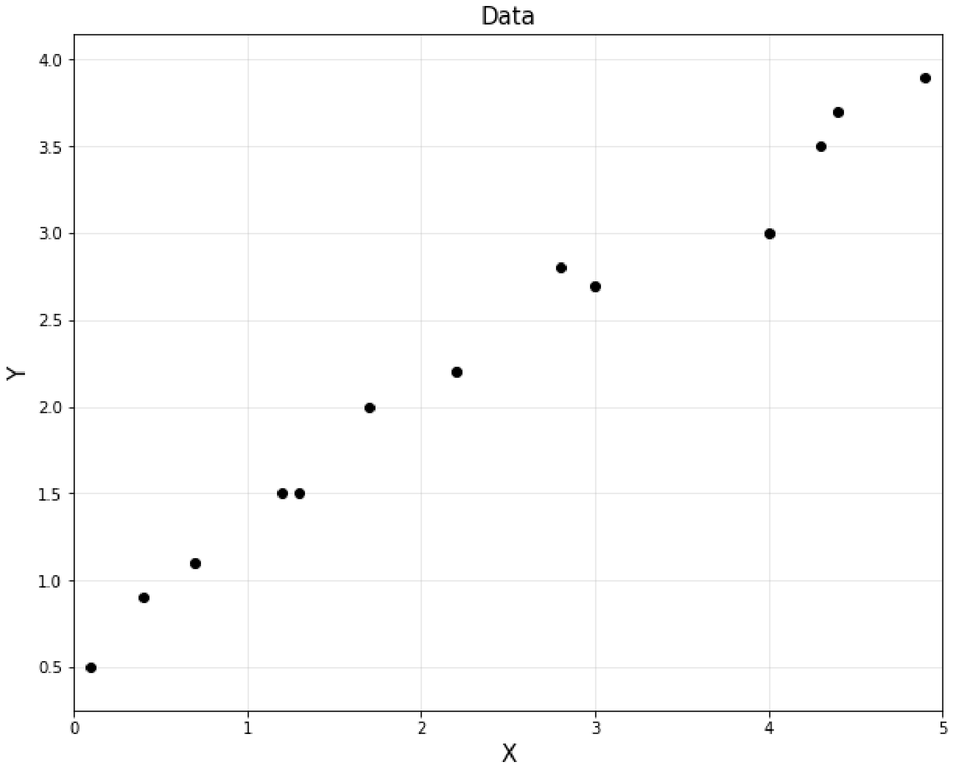
the same principle in a higher dimension

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

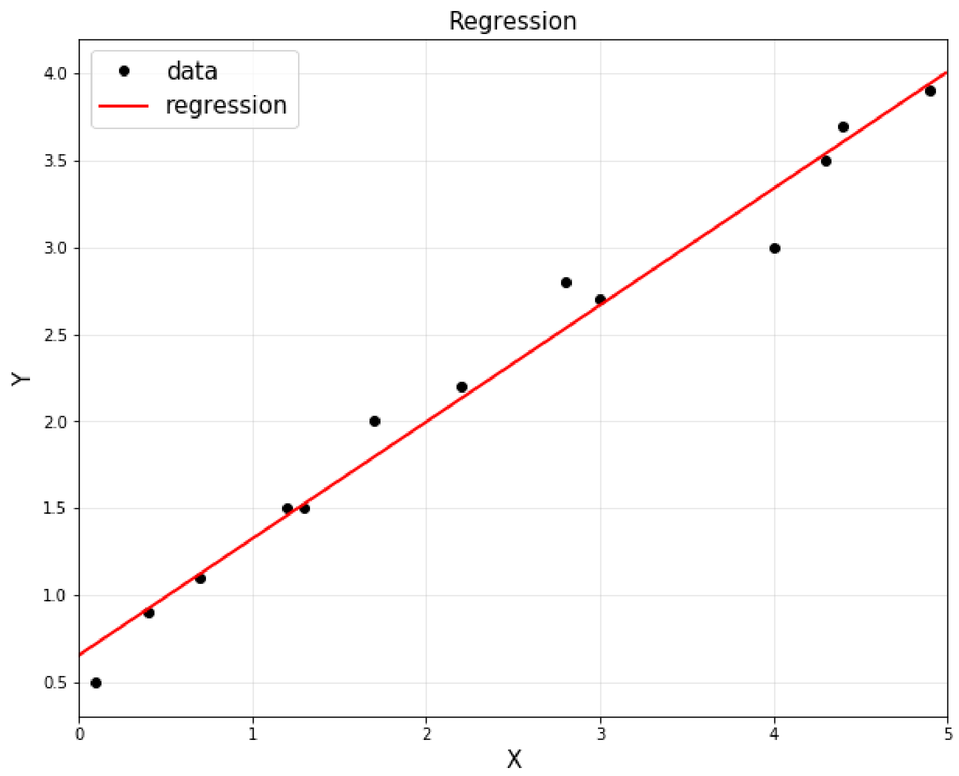
- known as *least square*

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

```
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

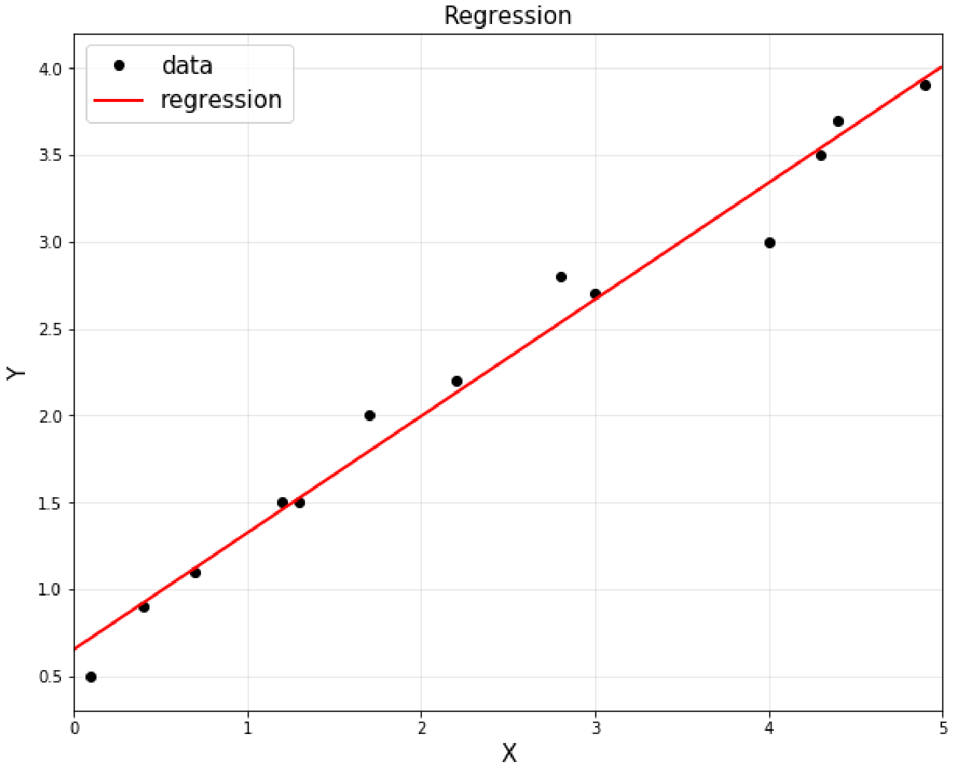
$$\min_{\theta} \|\hat{y} - y\|_2^2 = \min_{\theta} \|A\theta - y\|_2^2$$

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

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

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

```
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)
```



3. Solve using CVXPY Optimization

```
theta2 = cvx.Variable([2, 1])
obj = cvx.Minimize(cvx.norm(A*theta2-y, 2))
cvx.Problem(obj,[]).solve()

print('theta:\n', theta2.value)
```

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

$$\boxed{\min_{\theta} \ \|\hat{y} - y\|_2 = \min_{\theta} \ \|A\theta - y\|_2}$$

3. Solve using CVXPY Optimization

```
theta2 = cvx.Variable([2, 1])
obj = cvx.Minimize(cvx.norm(A*theta2-y, 2))
cvx.Problem(obj,[]).solve()

print('theta:\n', theta2.value)
```

$$\boxed{\min_{\theta} \ \|\hat{y} - y\|_2 = \min_{\theta} \ \|A\theta - y\|_2}$$

- By the way, do we have to use only L_2 norm? No.
  - Let's use L_1 norm

- By the way, do we have to use only L_2 norm? No.
  - Let's use L_1 norm

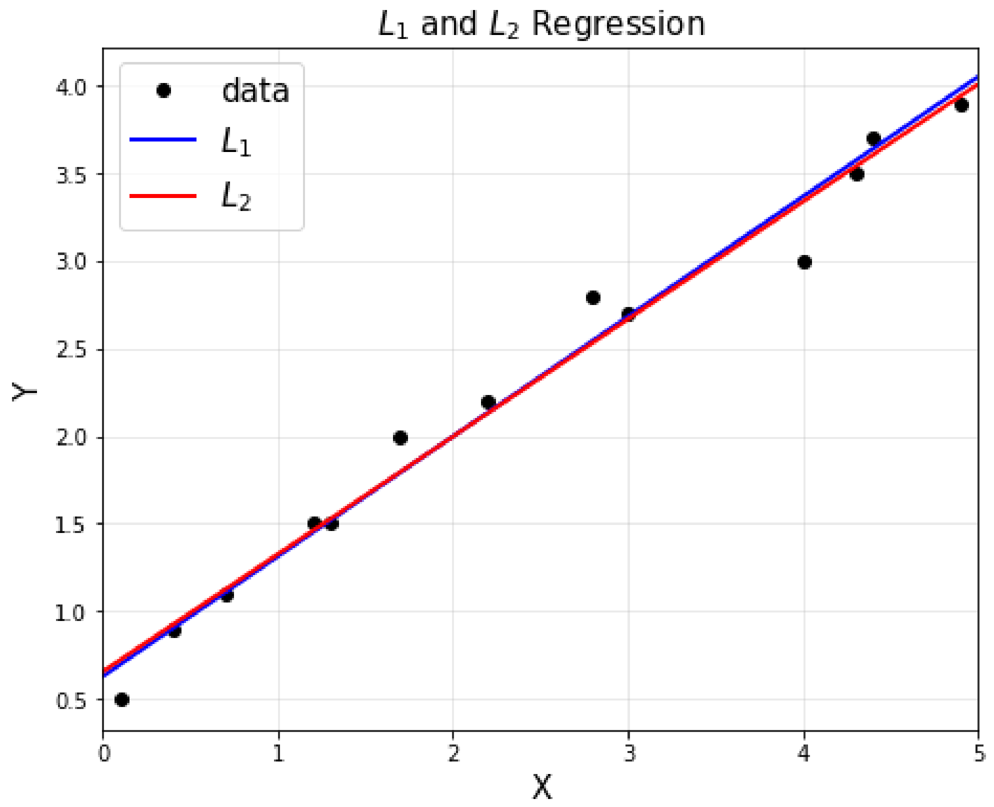
```
theta1 = cvx.Variable([2, 1])
obj = cvx.Minimize(cvx.norm(A*theta1-y, 1))
cvx.Problem(obj).solve()

print('theta:\n', theta1.value)
```

```
theta:
 [[0.628129  ]
 [0.68520147]]
```

$$\boxed{\min_{\theta} \ \|\hat{y} - y\|_1 = \min_{\theta} \ \|A\theta - y\|_1}$$

L_2 Norm vs. L_1 Norm

- L_1 norm also provides a decent linear approximation.



POSTECH

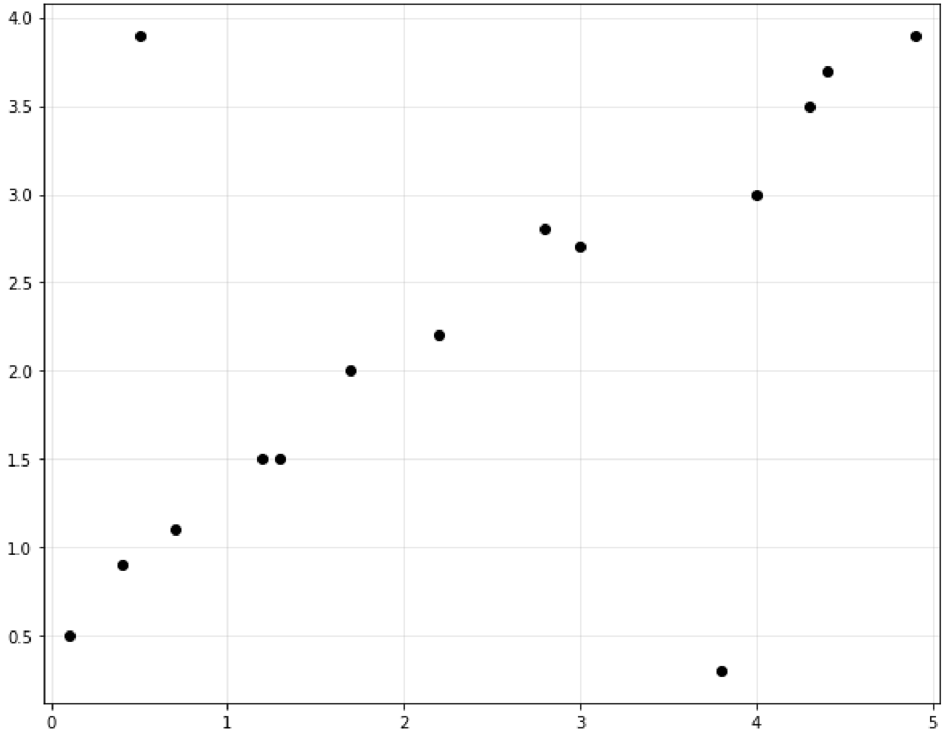
Regression with Outliers

- L_1 norm also provides a decent linear approximation.

- **What if outliers exist?**
  - Fitting with the different norms
  - source:
    - Week 9 of Computational Methods for Data Analysis by Coursera of Univ. of Washington
    - Chapter 17, online book available

Regression with Outliers

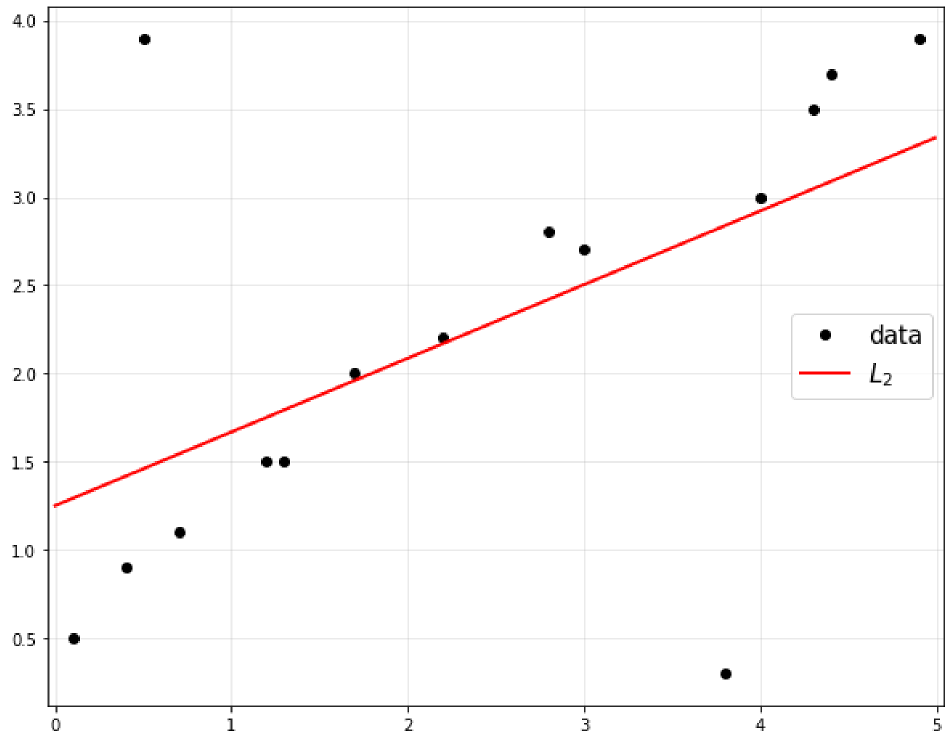
```
# add outliers
x = np.vstack([x, np.array([0.5, 3.8]).reshape(-1, 1)])
y = np.vstack([y, np.array([3.9, 0.3]).reshape(-1, 1)])

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



L_2 Norm vs. L_1 Norm

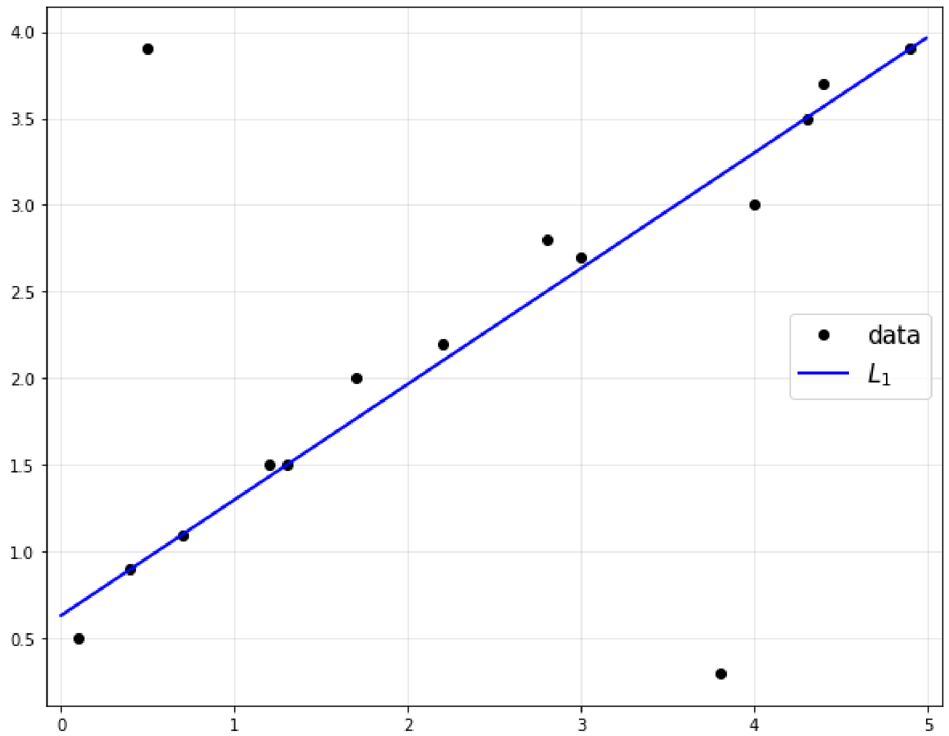
$$\min_{\theta} \ \|A\theta - y\|_2$$

```
theta2 = cvx.Variable([2, 1])
obj2 = cvx.Minimize(cvx.norm(A*theta2-y, 2))
prob2 = cvx.Problem(obj2).solve()
```



$$\min_{\theta} \ \|A\theta - y\|_1$$

```
theta1 = cvx.Variable([2, 1])
obj1 = cvx.Minimize(cvx.norm(A*theta1-y, 1))
prob1 = cvx.Problem(obj1).solve()
```

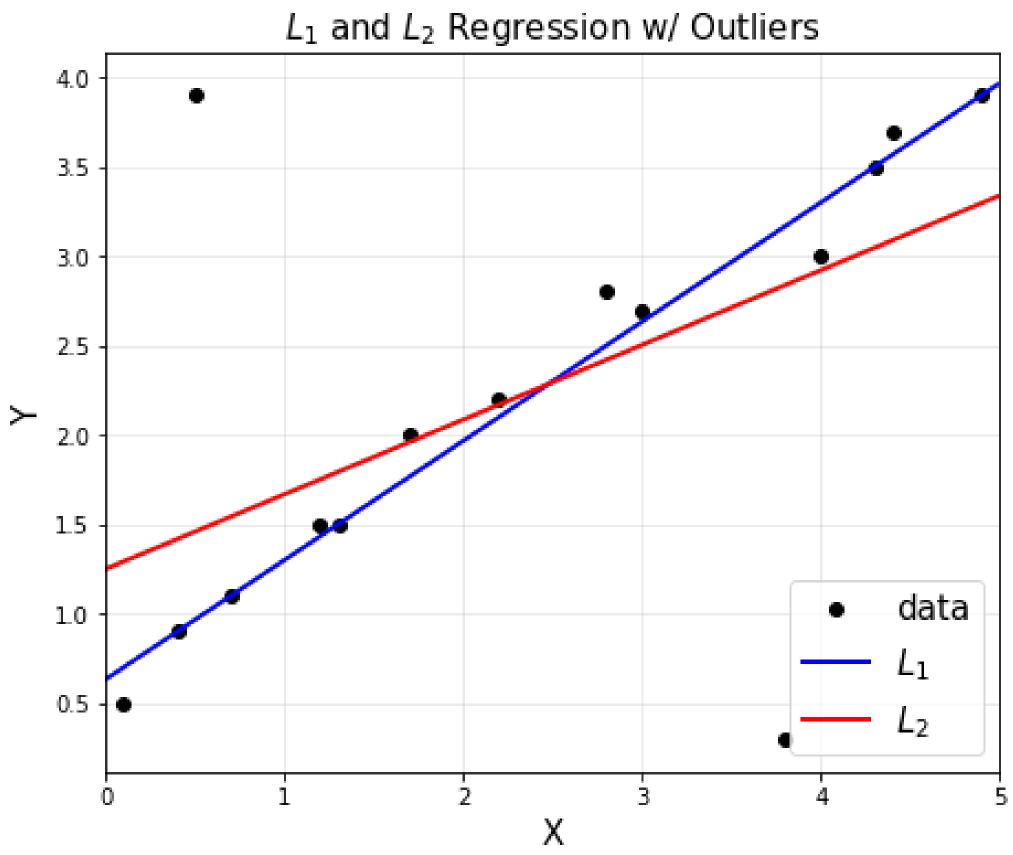


Think About What Makes Different

- It is important to understand what makes them different

$$\min_{\theta} \; \|A\theta - y\|_1$$

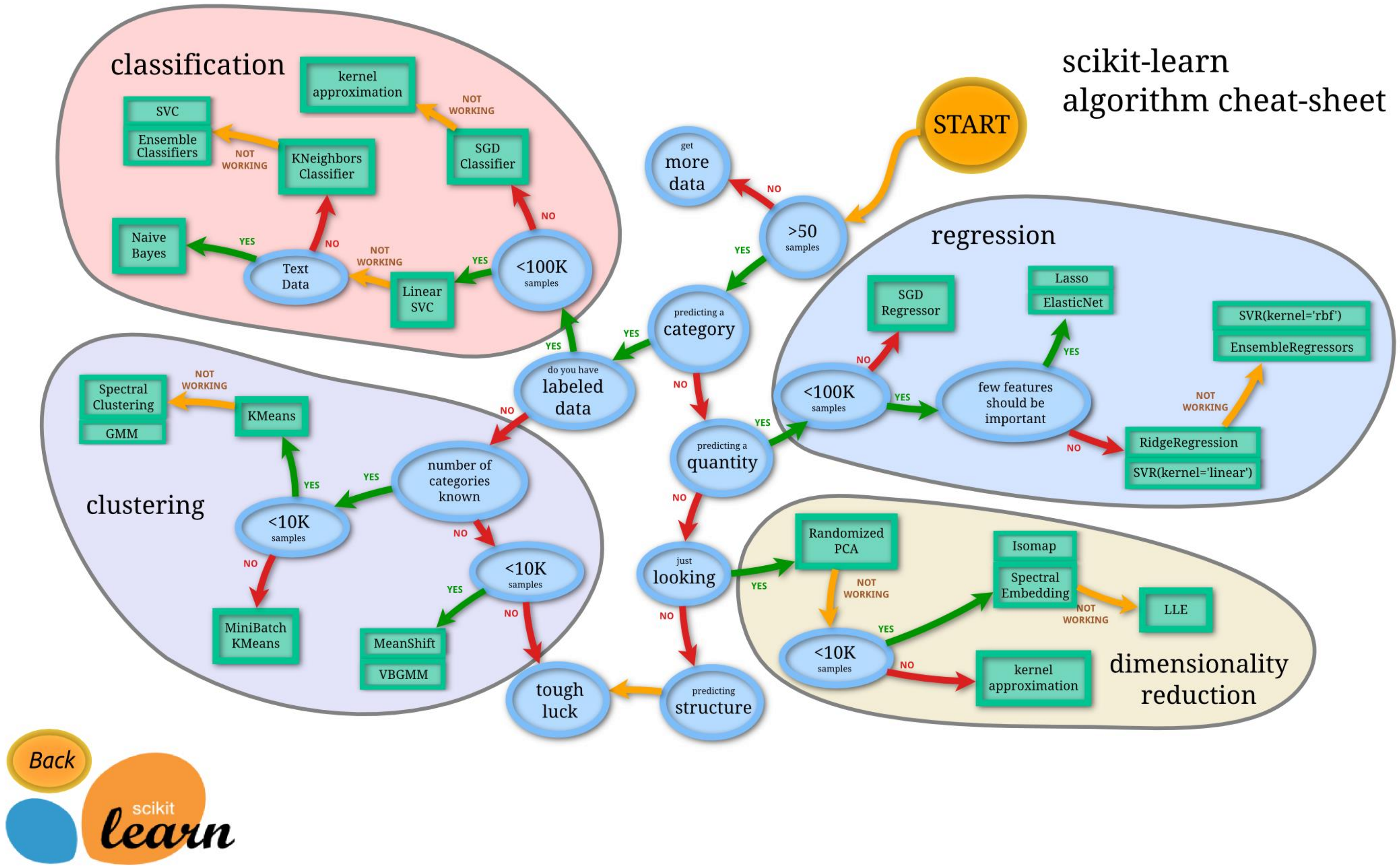
$$\min_{\theta} \; \|A\theta - y\|_2$$



Scikit-Learn

- Machine Learning in Python
- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license
- https://scikit-learn.org/stable/index.html#

Scikit-Learn



Scikit-Learn: Regression

```
from sklearn import linear_model
```

```
reg = linear_model.LinearRegression()
reg.fit(x, y)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

```
reg.coef_
```

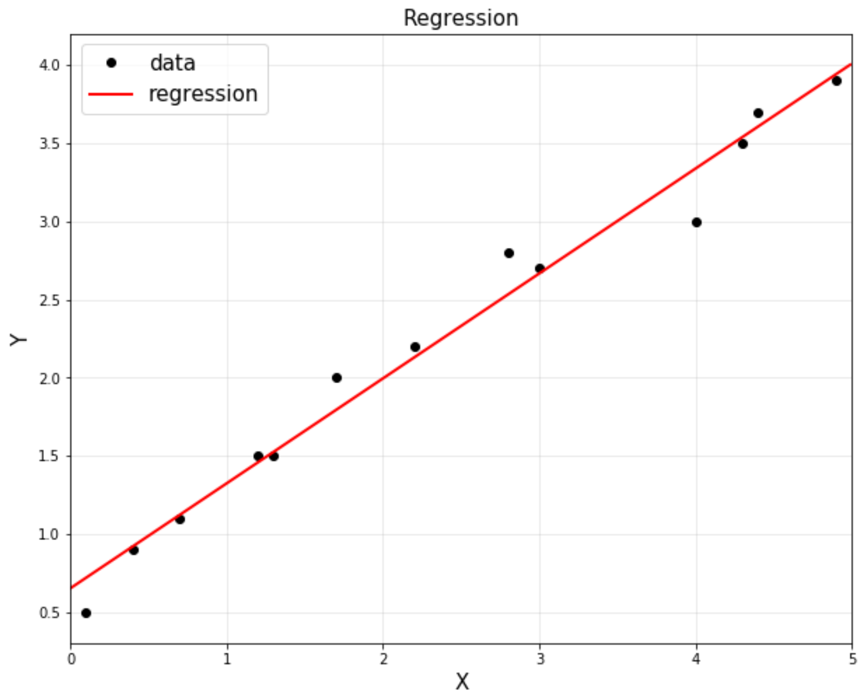
```
array([[0.67129519]])
```

```
reg.intercept_
```

```
array([0.65306531])
```

Scikit-Learn: Regression

```
# 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)
plt.plot(xp, reg.predict(xp), 'r', linewidth=2, label="regression")
plt.legend(fontsize=15)
plt.axis('equal')
plt.grid(alpha=0.3)
plt.xlim([0, 5])
plt.show()
```

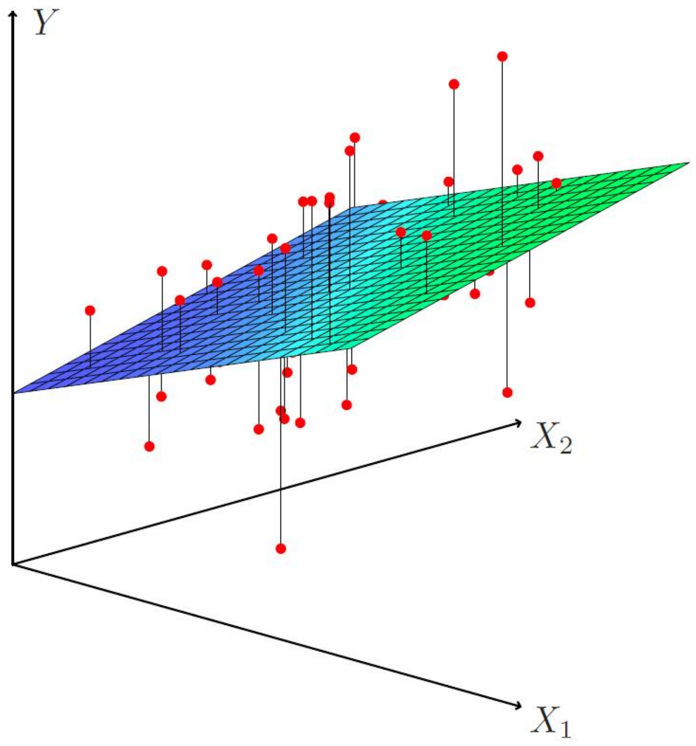


Multivariate Linear Regression

- Linear regression for multivariate data

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

$$\phi\left(x^{(i)}\right) = \begin{bmatrix} 1 \\ x_1^{(i)} \\ x_2^{(i)} \end{bmatrix} \qquad \Longrightarrow \quad \theta^* = (\Phi^T \Phi)^{-1} \Phi^T y$$

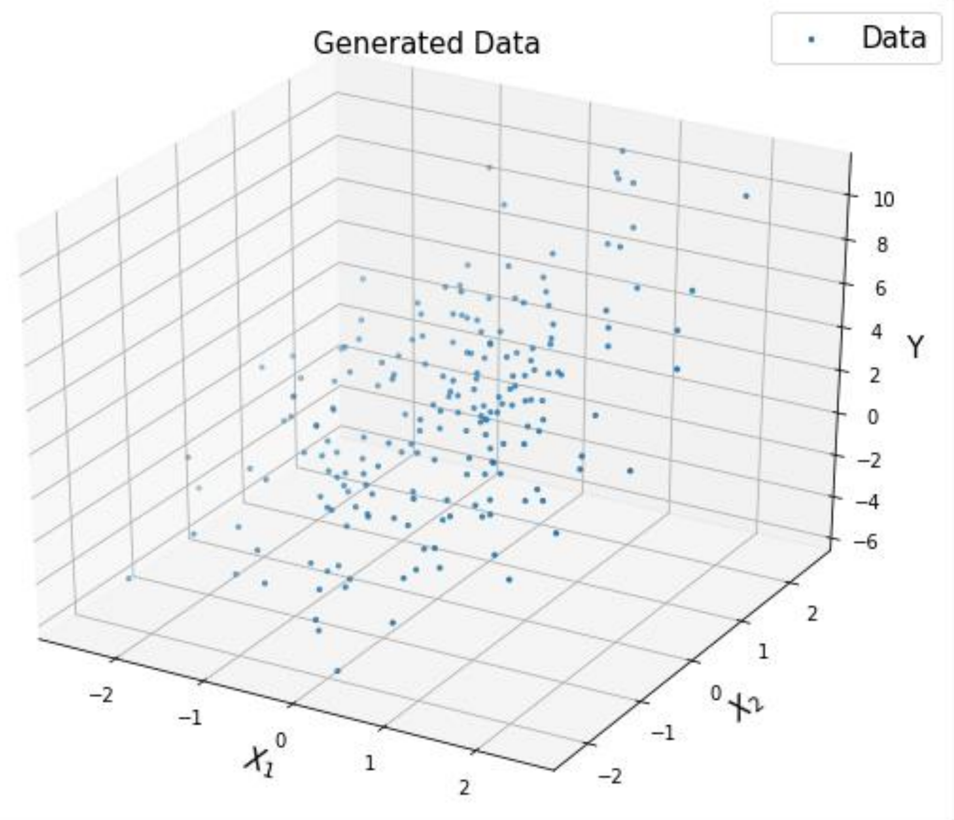


$$\Phi = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \\ \vdots & & \\ 1 & x_1^{(m)} & x_2^{(m)} \end{bmatrix} \quad \Longrightarrow \quad \hat{y} = \begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \vdots \\ \hat{y}^{(m)} \end{bmatrix} = \Phi\theta$$

- Same in matrix representation

Multivariate Linear Regression

```python
# y = theta0 + theta1*x1 + theta2*x2 + noise

n = 200
x1 = np.random.randn(n, 1)
x2 = np.random.randn(n, 1)
noise = 0.5*np.random.randn(n, 1);

y = 2 + 1*x1 + 3*x2 + noise
```

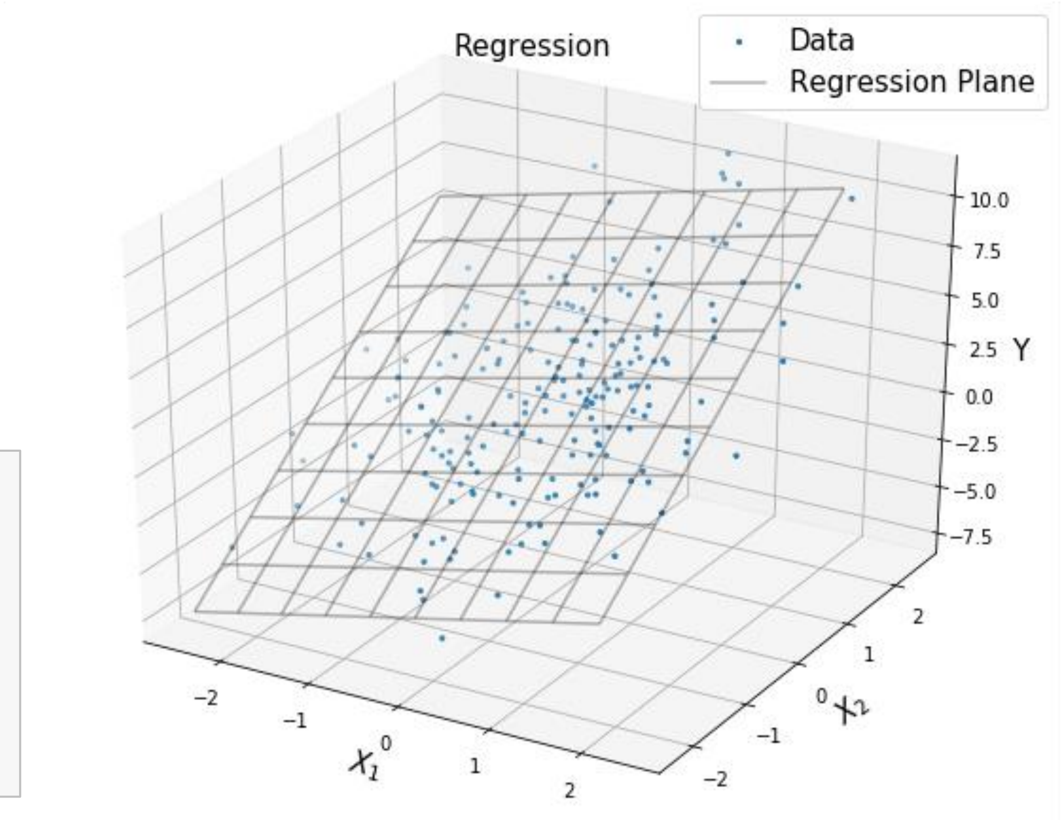


Multivariate Linear Regression

$$\Phi = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \\ \vdots & & \\ 1 & x_1^{(m)} & x_2^{(m)} \end{bmatrix} \quad \Longrightarrow \quad \hat{y} = \begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \vdots \\ \hat{y}^{(m)} \end{bmatrix} = \Phi\theta$$

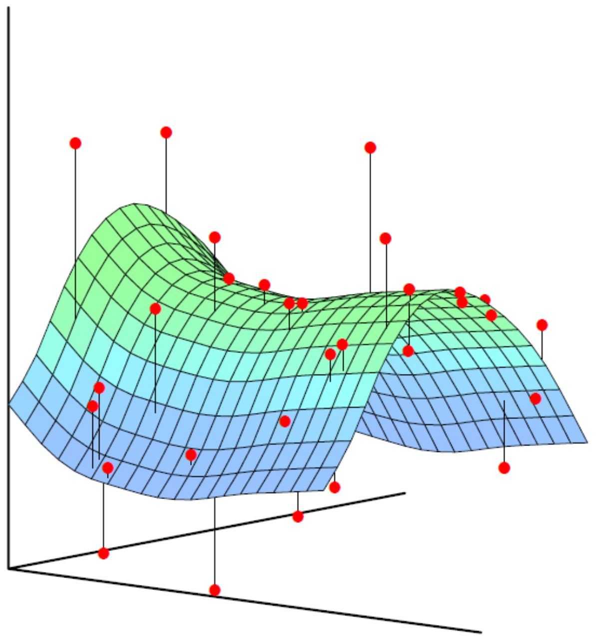
$$\Longrightarrow \quad \theta^* = (\Phi^T \Phi)^{-1} \Phi^T y$$

```
A = np.hstack([np.ones((n, 1)), x1, x2])
A = np.asmatrix(A)
theta = (A.T*A).I*A.T*y

X1, X2 = np.meshgrid(np.arange(np.min(x1), np.max(x1), 0.5),
                     np.arange(np.min(x2), np.max(x2), 0.5))
YP = theta[0,0] + theta[1,0]*X1 + theta[2,0]*X2
```



Nonlinear Regression

- Linear regression for non-linear data



- Same as linear regression, just with non-linear features
- Method 1: constructing explicit feature vectors
  - polynomial features
  - Radial basis function (RBF) features
- Method 2: implicit feature vectors, kernel trick (*optional*)

Nonlinear Regression

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

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

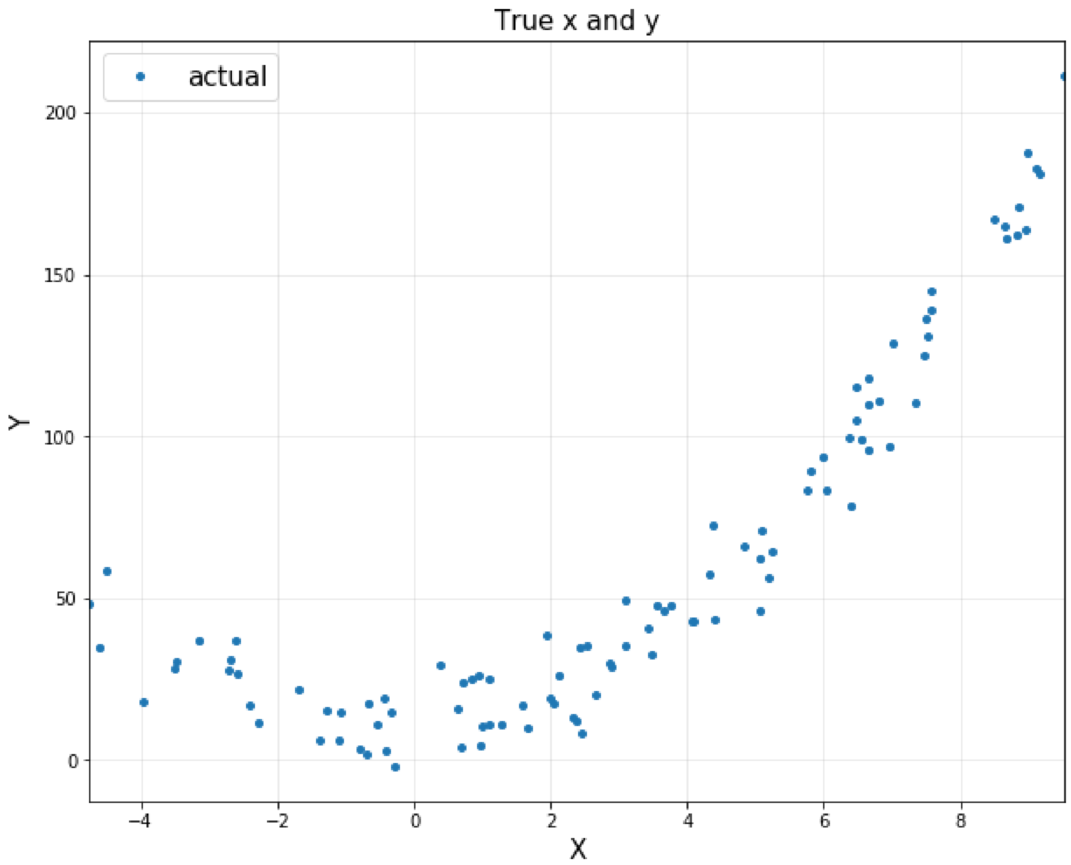
$$\phi(x_i) = \begin{bmatrix} 1 \\ x_i \\ x_i^2 \end{bmatrix}$$

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

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

Polynomial Regression

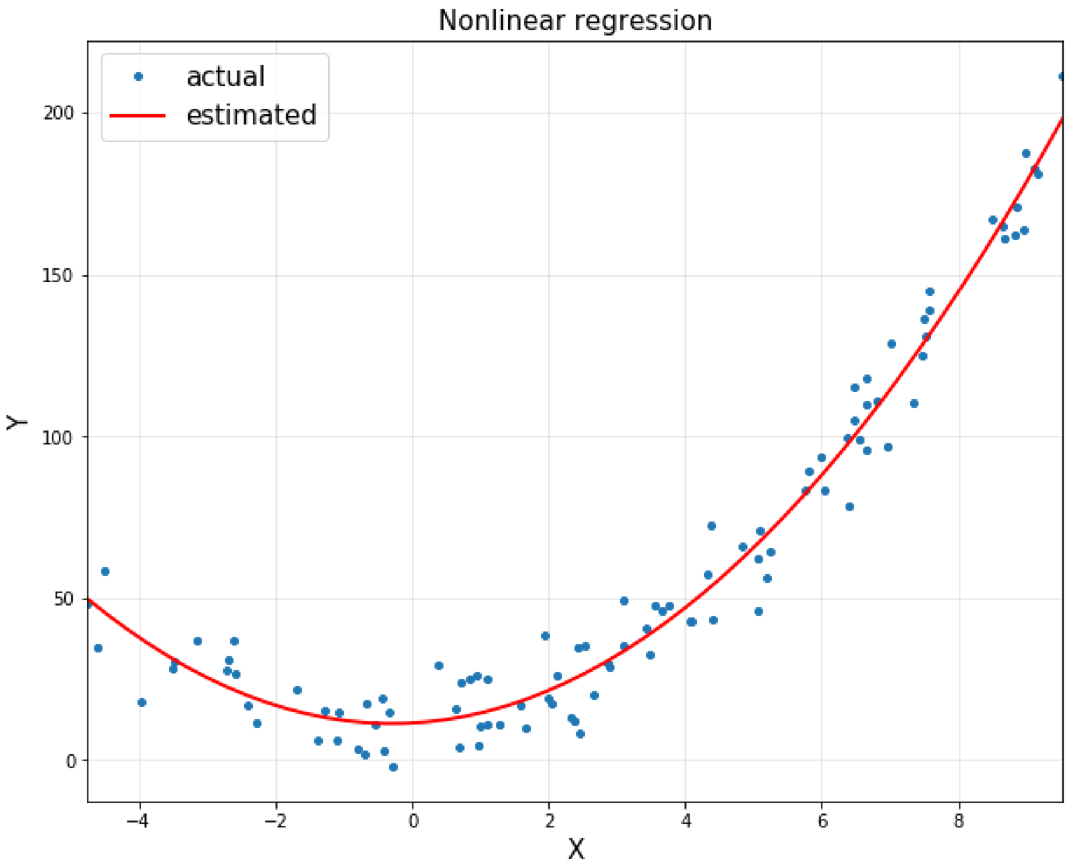
```
# y = theta0 + theta1*x + theta2*x^2 + noise

n = 100
x = -5 + 15*np.random.rand(n, 1)
noise = 10*np.random.randn(n, 1)

y = 10 + 1*x + 2*x**2 + noise
```



Polynomial Regression

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

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

theta = (A.T*A).I*A.T*y
print('theta:\n', theta)
```



Summary: Linear Regression

- Though linear regression may seem limited, it is very powerful, since the input features can themselves include non-linear features of data

- Linear regression on non-linear features of data

- For least-squares loss, optimal parameters still are

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

Regression 2

Industrial AI Lab.

Prof. Seungchul Lee

Linear Regression: Advanced

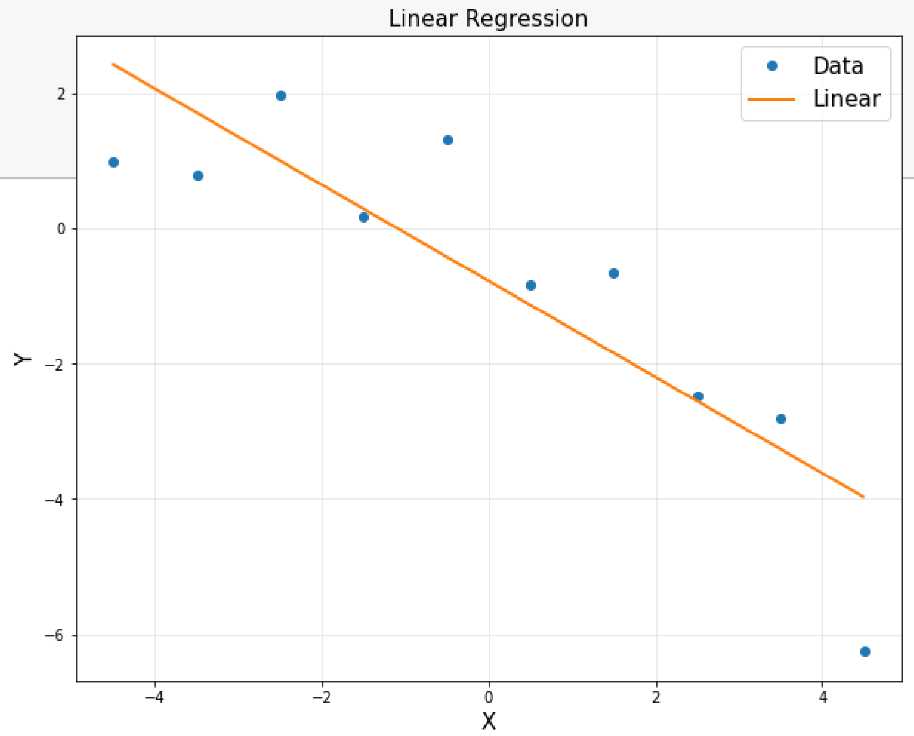
- Overfitting
- Linear Basis Function Models
- Regularization (Ridge and Lasso)
- Evaluation

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)

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

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

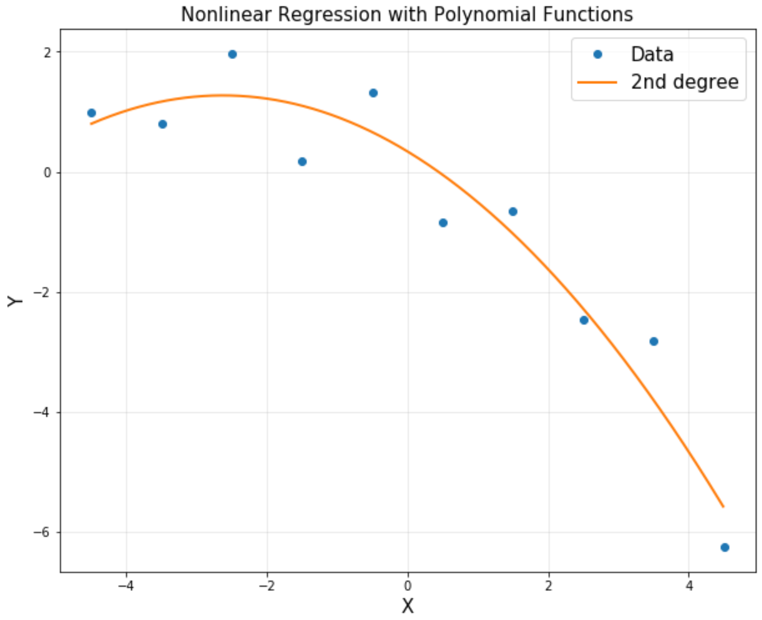
$$\Phi = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & & \\ 1 & x_m & x_m^2 \end{bmatrix} \quad \implies \quad \hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_m \end{bmatrix} = \Phi\theta$$

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

Nonlinear Regression

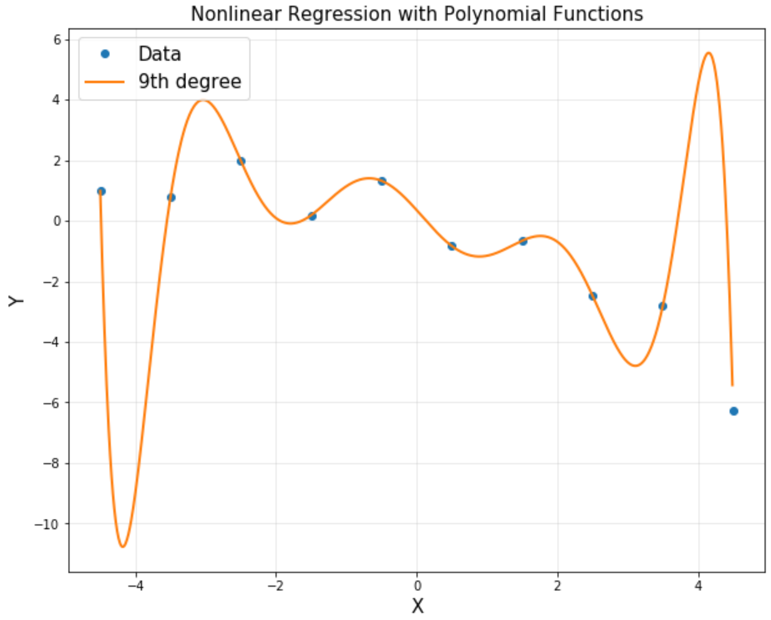
```python
A = np.hstack([x**0, x, x**2])
A = np.asmatrix(A)

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

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

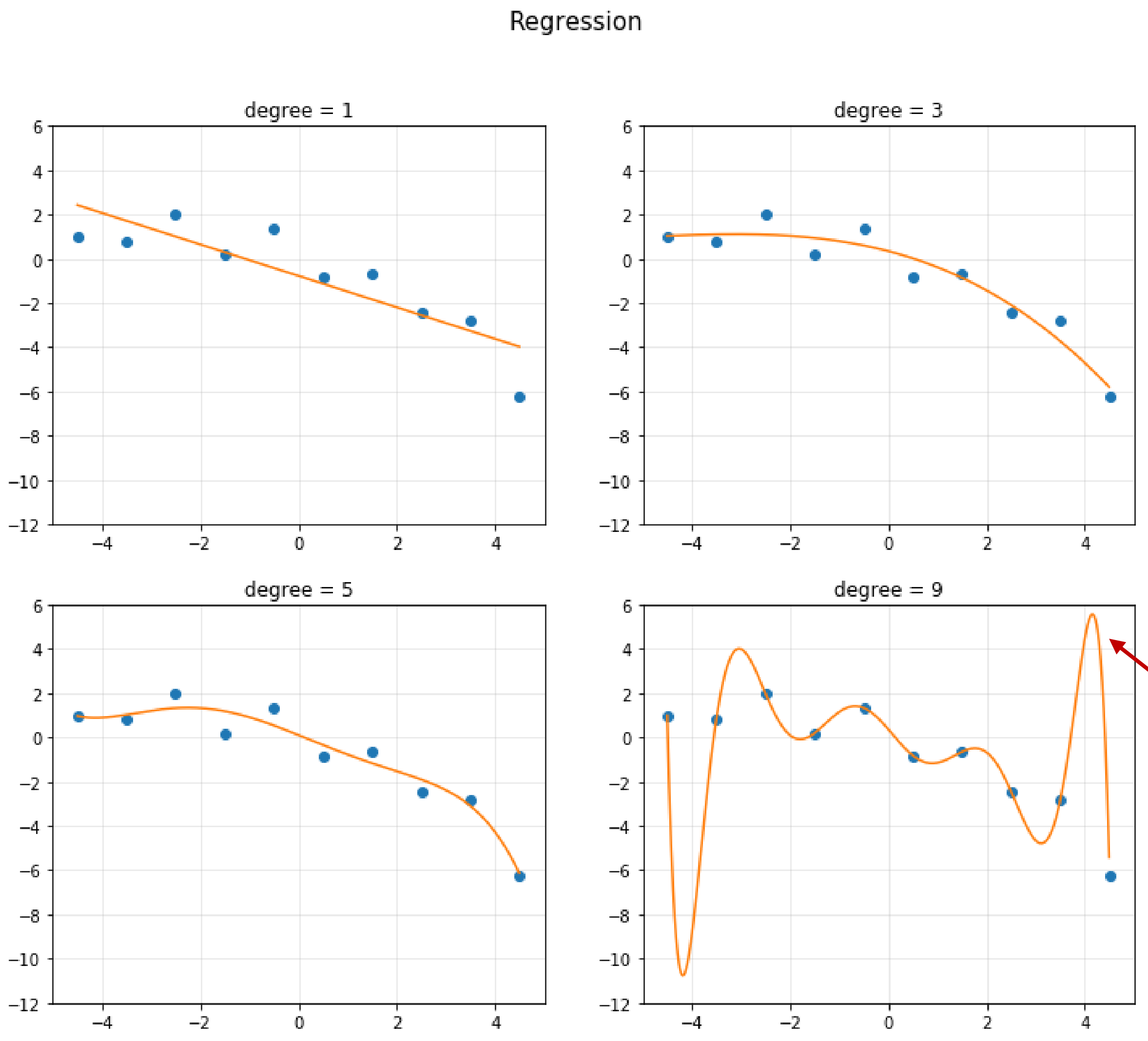



```python
A = np.hstack([x**i for i in range(10)])
A = np.asmatrix(A)

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

10 input points with degree 9 (or 10)



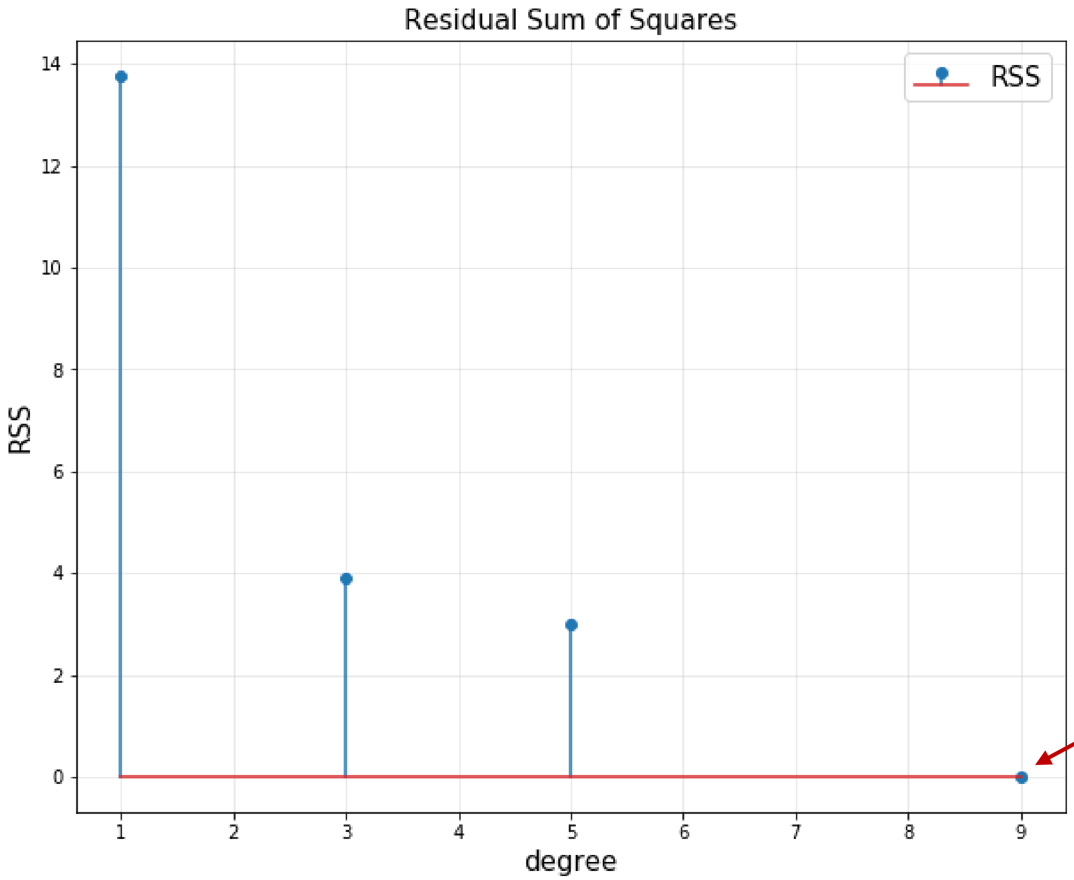
Polynomial Fitting with Different Degrees



Low error on input data points, but high error nearby

Loss

- Loss: Residual Sum of Squares (RSS)



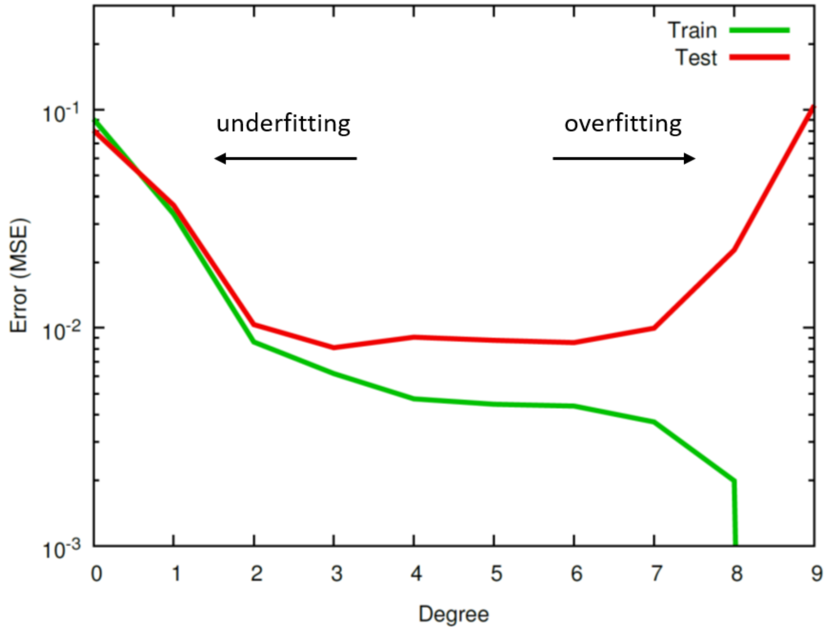
$$\min_{\theta} \|\hat{y} - y\|_2^2$$

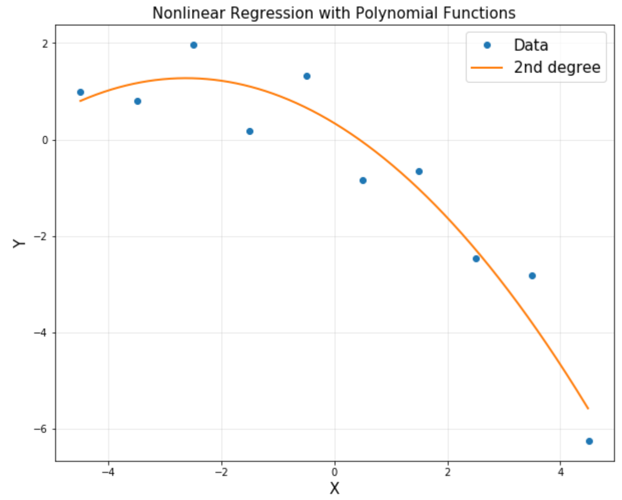
Minimizing loss in training data is often not the best

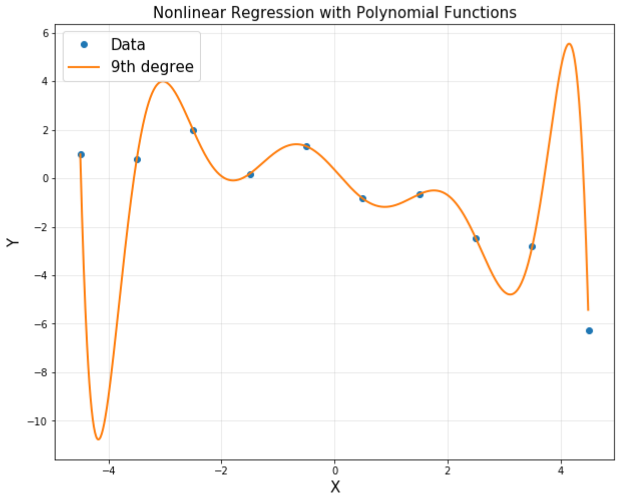
Low error on input data points, but high error nearby

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







Function Approximation:
Linear Basis Function Model

Function Approximation

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

Recap: Nonlinear Regression

- 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

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

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

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

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

Construct Explicit Feature Vectors

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

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

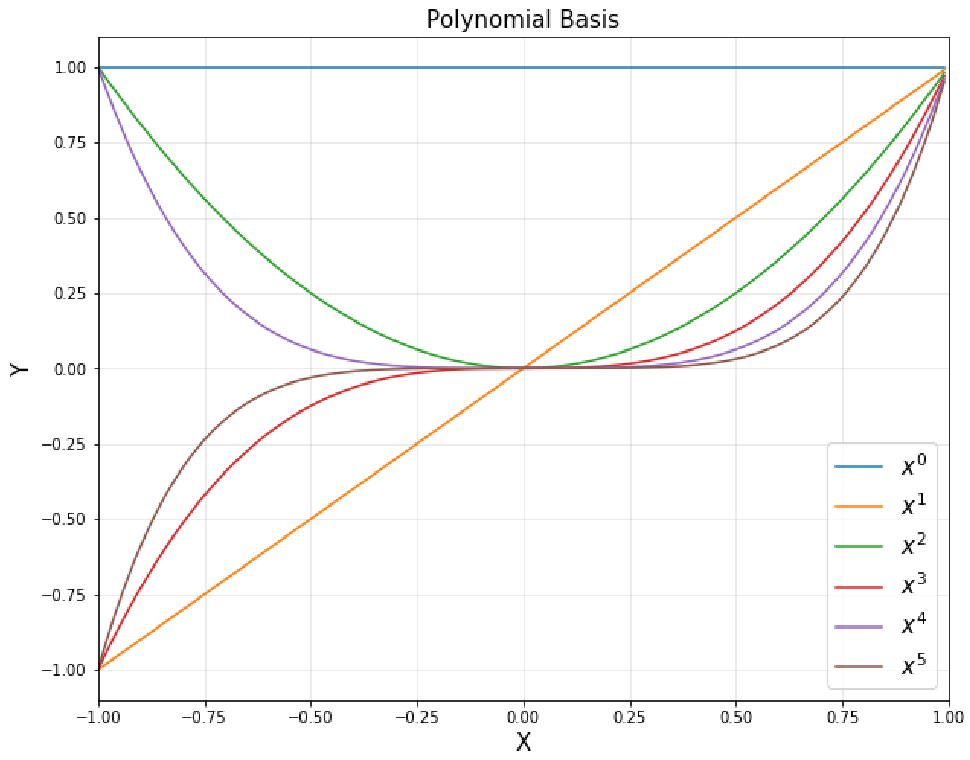
$$\Phi = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & & \\ 1 & x_m & x_m^2 \end{bmatrix} \quad \Longrightarrow \quad \hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_m \end{bmatrix} = \Phi\theta$$

Polynomial Basis

1) Polynomial functions

$$b_i(x) = x^i, \quad i = 0, \cdots, d$$

```
xp = np.arange(-1, 1, 0.01).reshape(-1, 1)
polybasis = np.hstack([xp**i for i in range(6)])

plt.figure(figsize=(10, 8))

for i in range(6):
    plt.plot(xp, polybasis[:,i], label = '$x^{}$'.format(i))
```

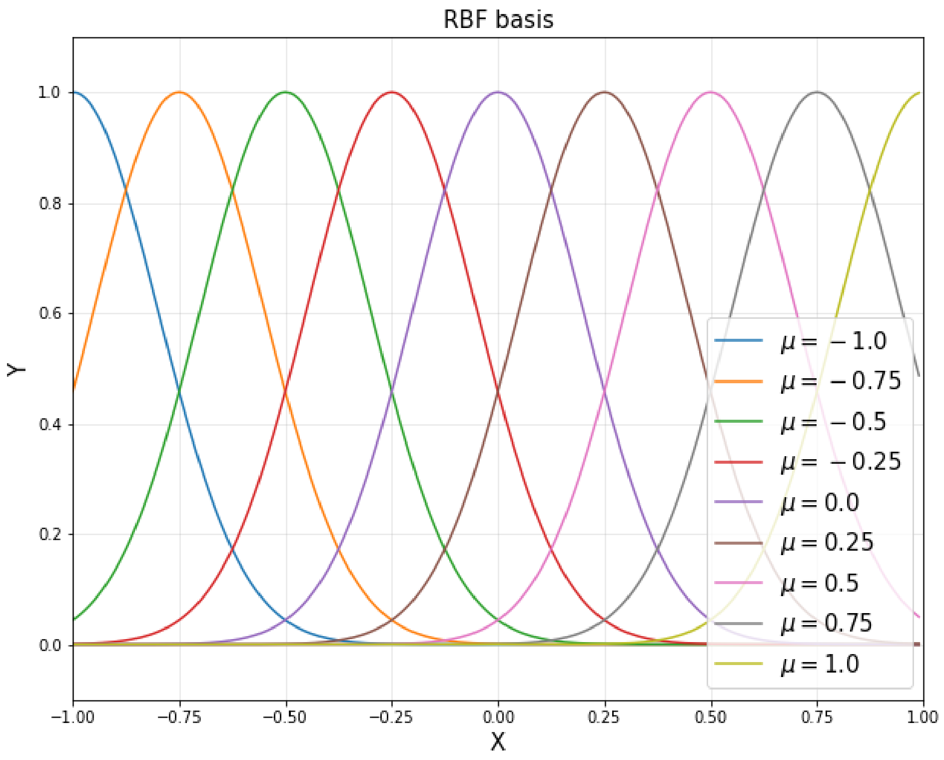


RBF Basis

2) Radial Basis Functions (RBF) with bandwidth σ and k RBF centers $\mu_i \in \mathbb{R}^n, \qquad i = 1, 2, \cdots, k$

$$b_i(x) = \exp\left(-\frac{\|x - \mu_i\|^2}{2\sigma^2}\right)$$

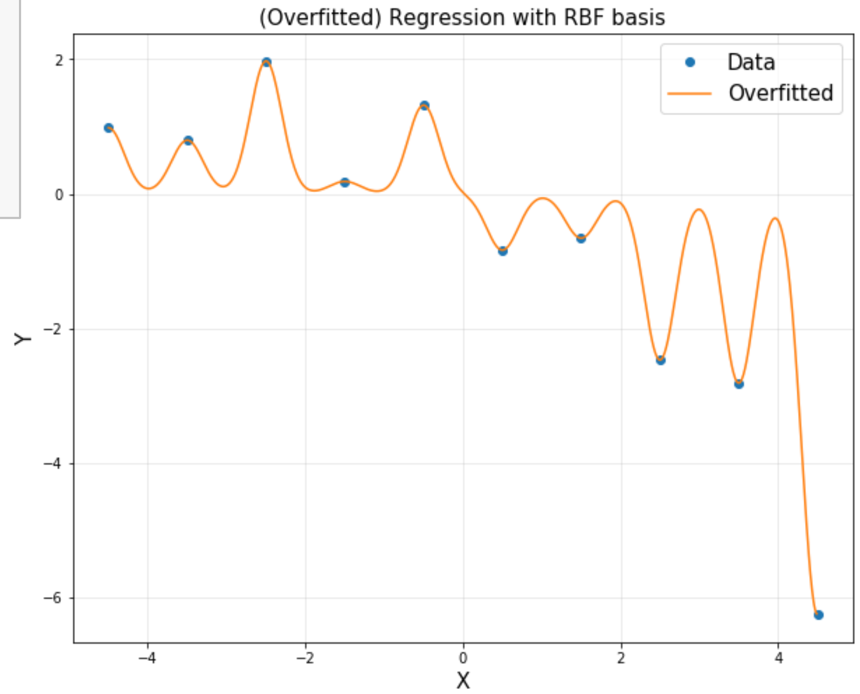
```
d = 9

u = np.linspace(-1, 1, d)
sigma = 0.2

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

plt.figure(figsize=(10, 8))

for i in range(d):
    plt.plot(xp, rbfbasis[:,i], label='$\mu = {}$'.format(u[i]))
```



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
```

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

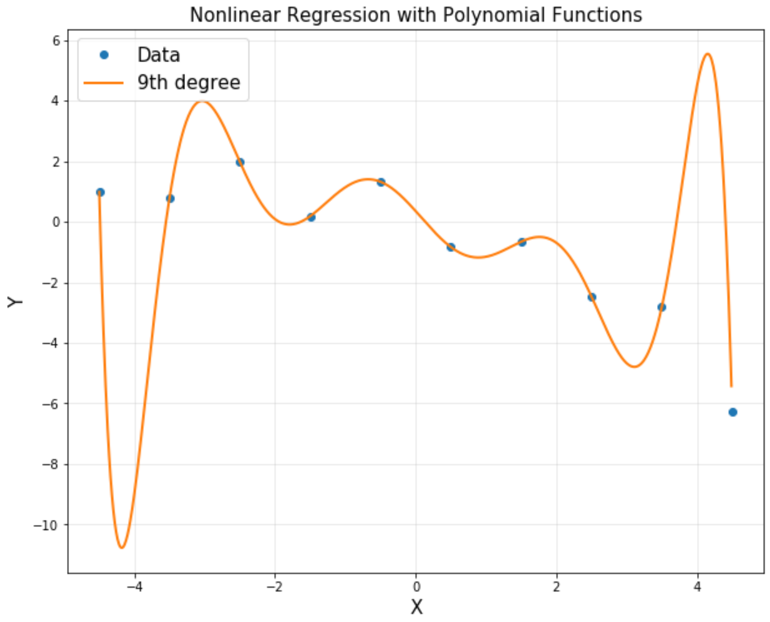


- With many features, our prediction function becomes very expensive
- Can lead to overfitting

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



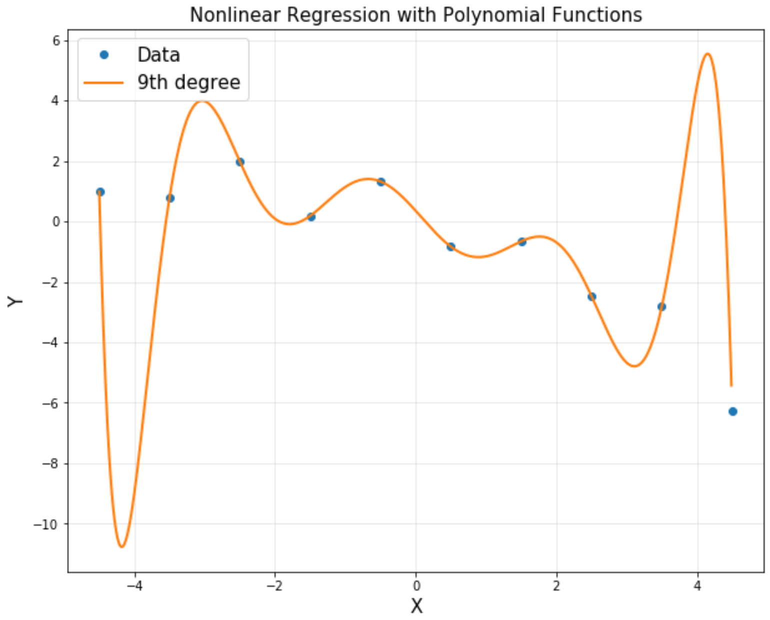
Generalization Error

- 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

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

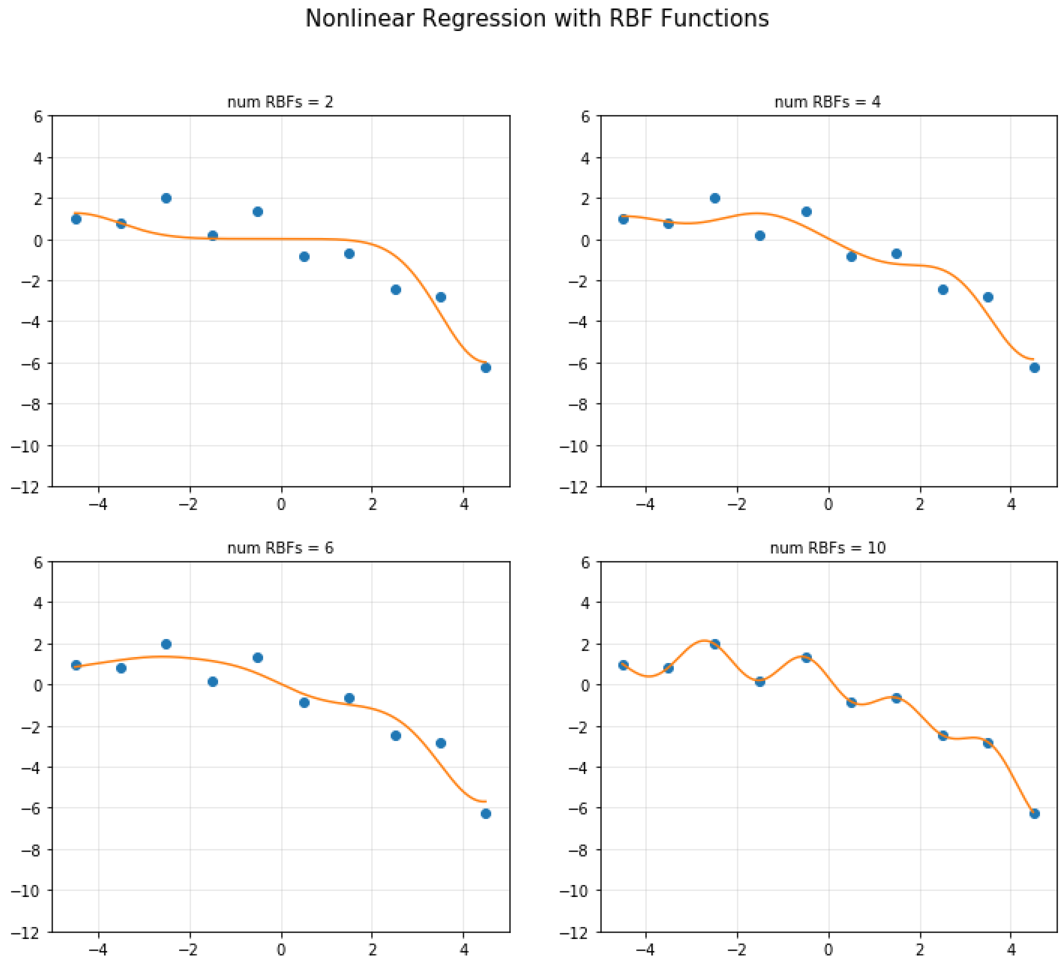
  - λ: regularization parameter, trades off between low loss and small values of θ

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

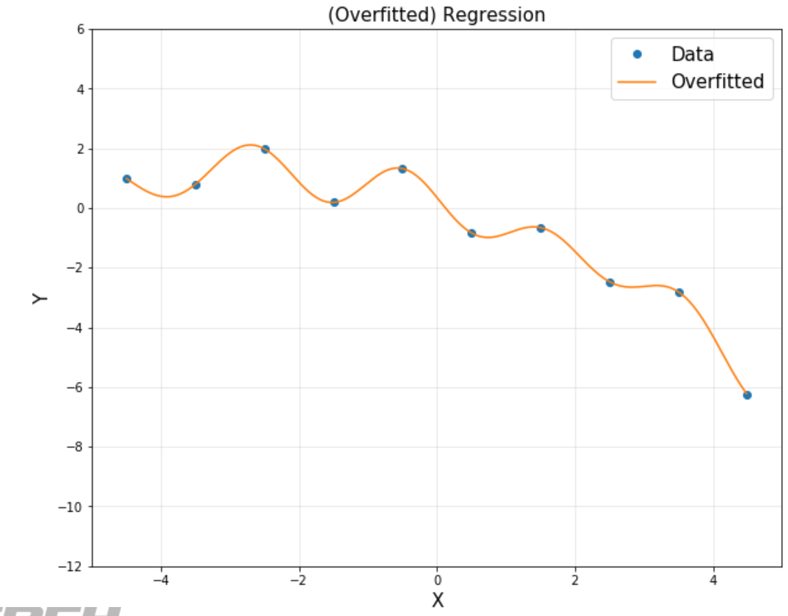
- the tuning parameter λ serves to control the relative impact of these two terms on the regression coefficient estimates

- 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
```

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

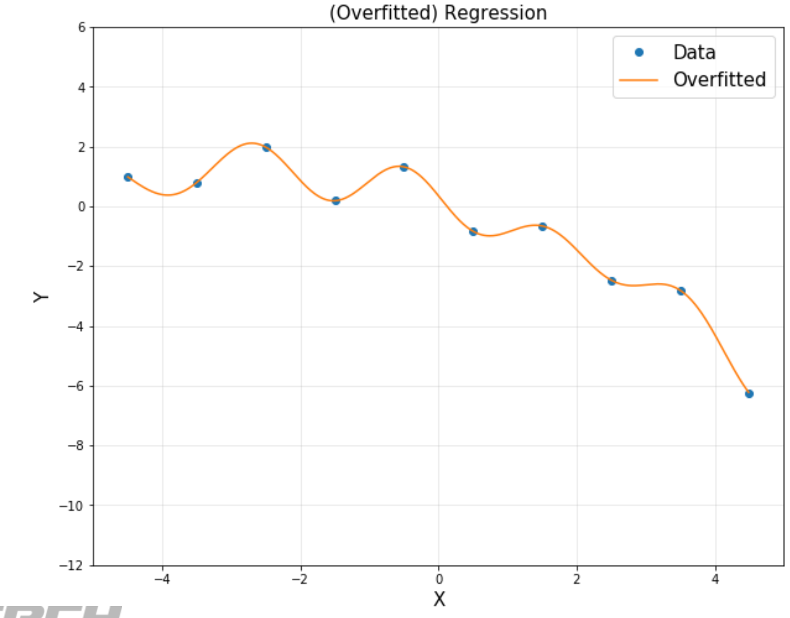


RBF with Regularization

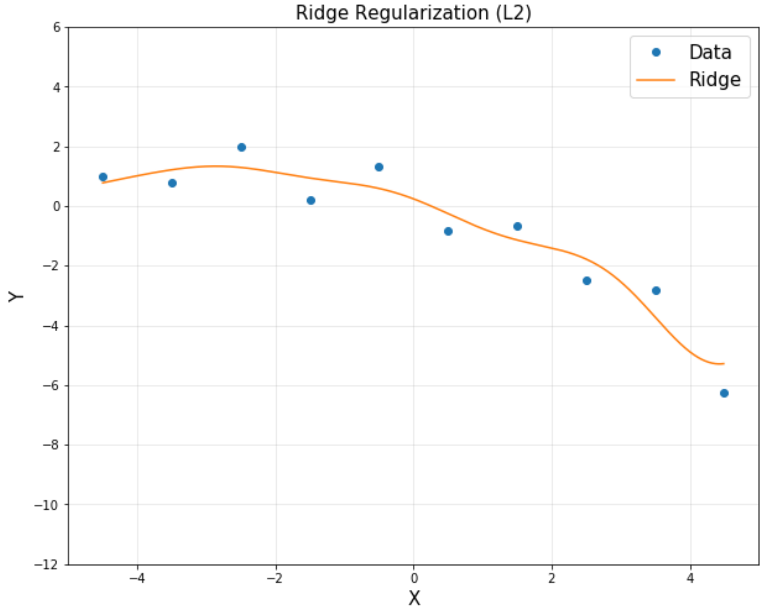
- Start from rich representation. Then, regularize coefficients θ

```python
# 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
```

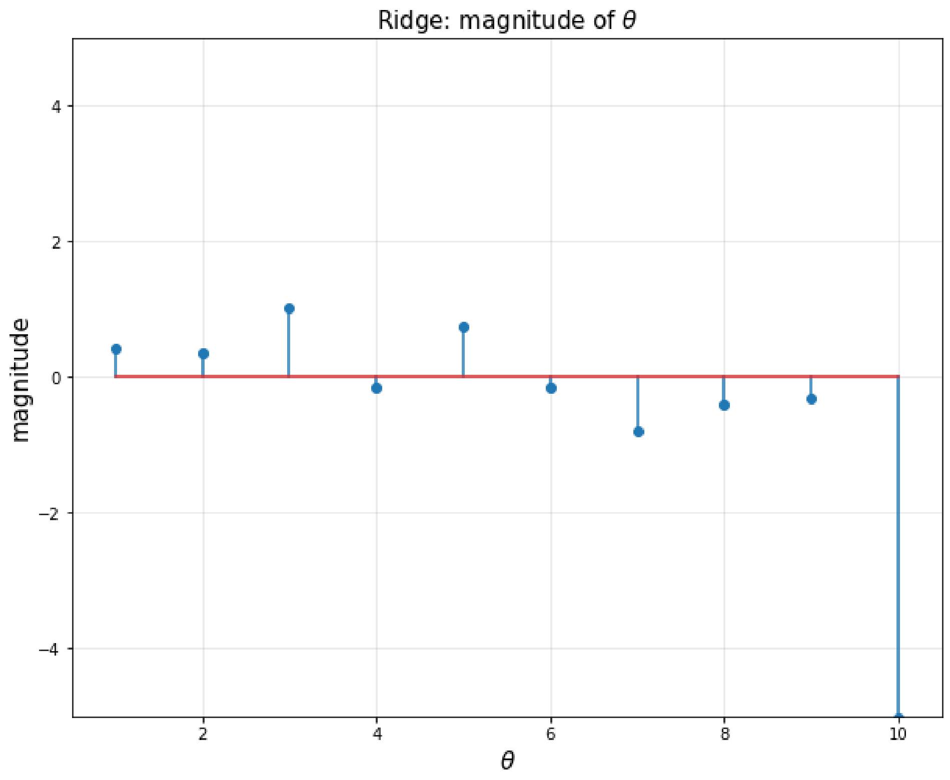
$$\min \ \|\Phi\theta - y\|_2^2 + \lambda\|\theta\|_2^2$$

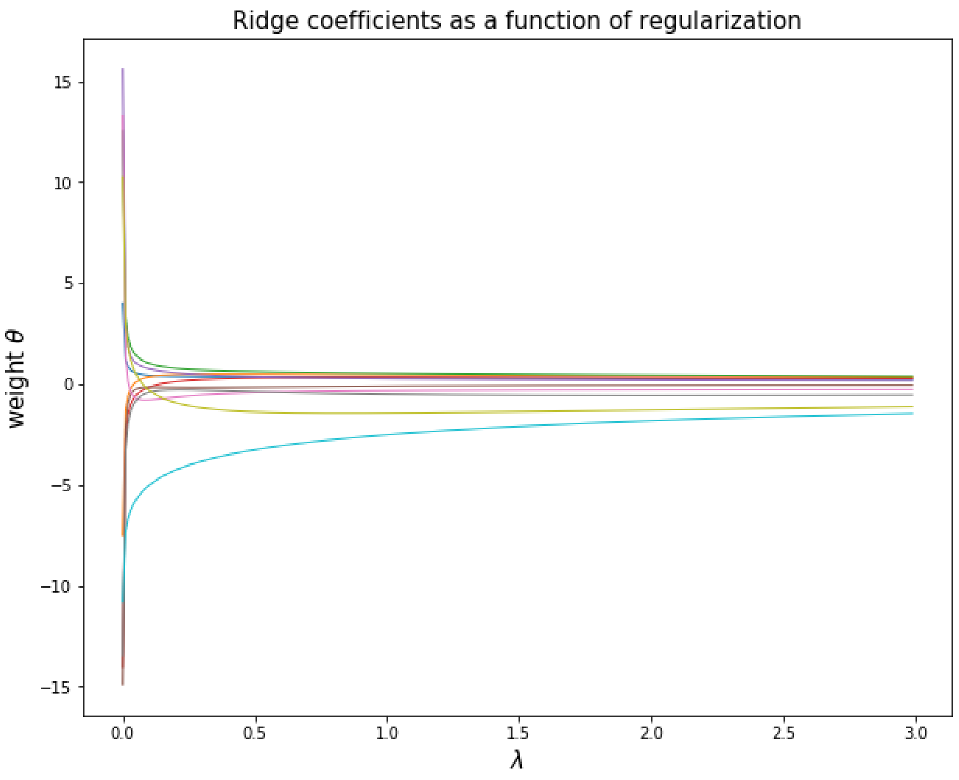


$\Longrightarrow$



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...

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

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

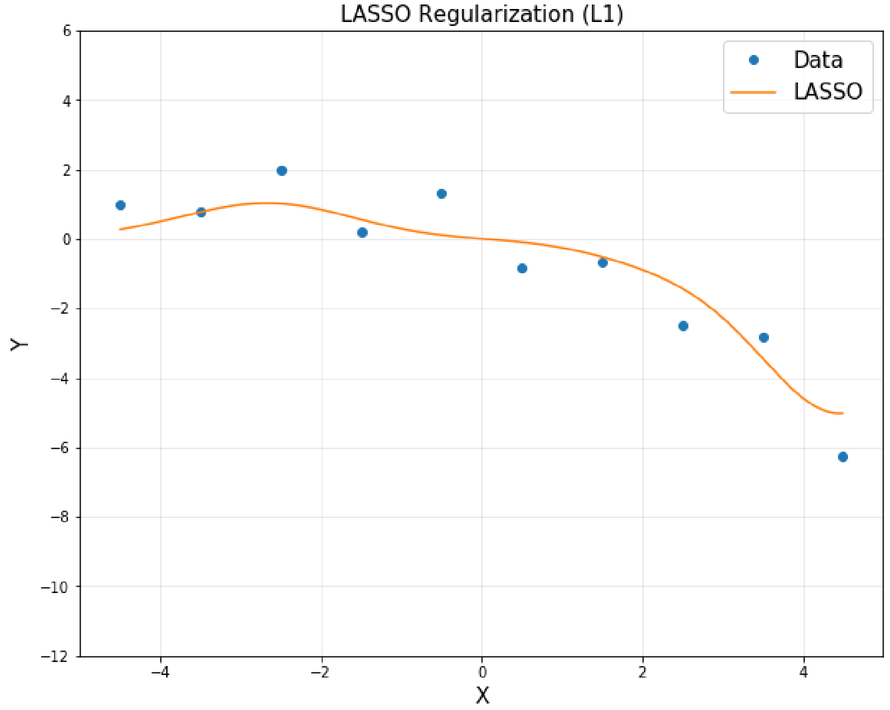
- λ is a tuning parameter = balance of fit and sparsity
- Known as *LASSO*
  - least absolute shrinkage and selection operator

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
```

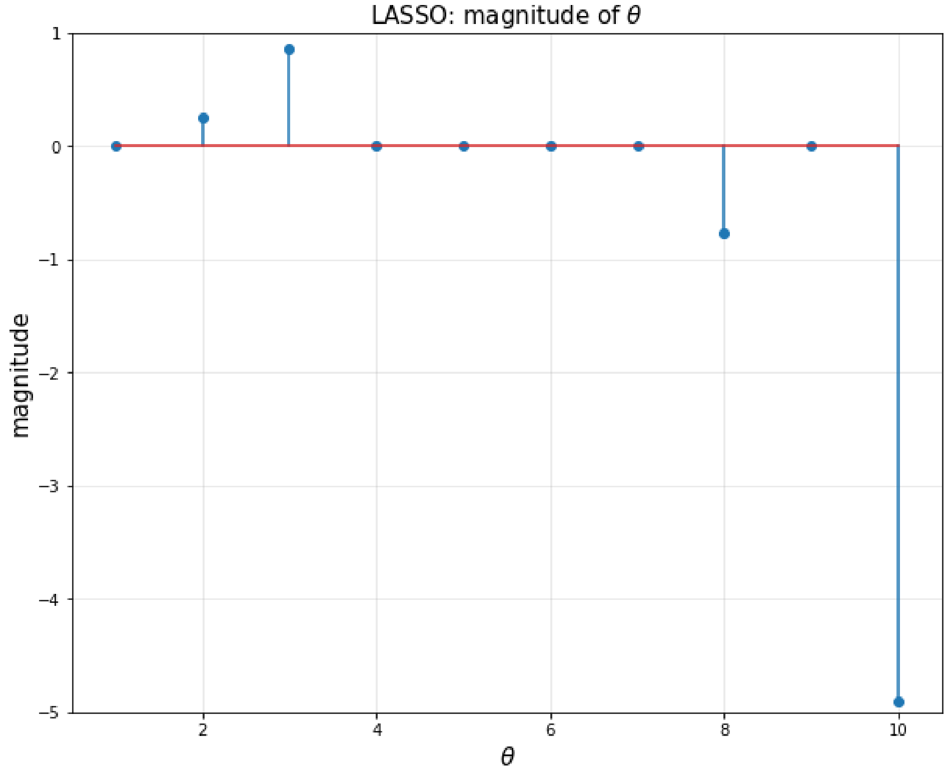
$$\min \; \|\Phi\theta - y\|_2^2 + \lambda\|\theta\|_1$$

- Approximated function looks similar to that of ridge regression

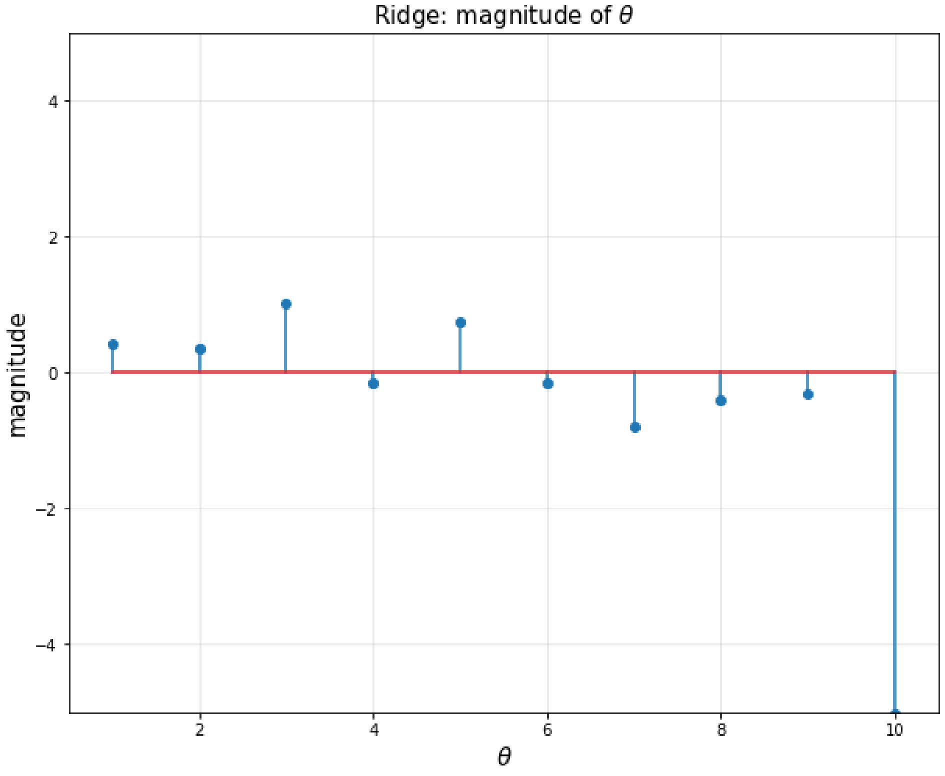


Coefficients θ with LASSO

- Non-zero coefficients indicate 'selected' features



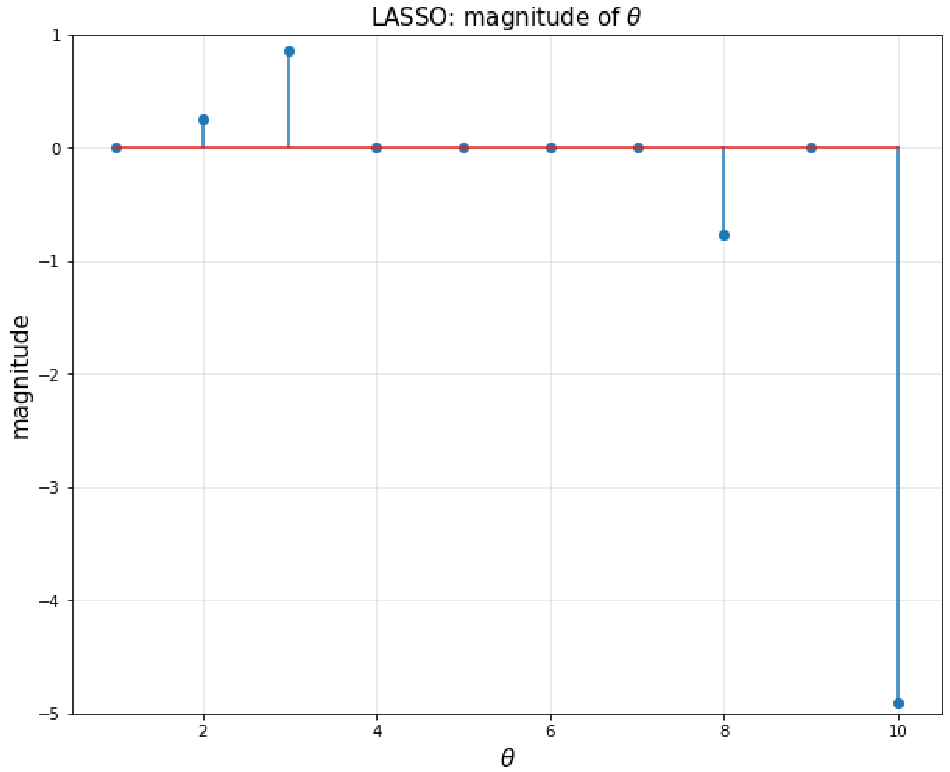
LASSO

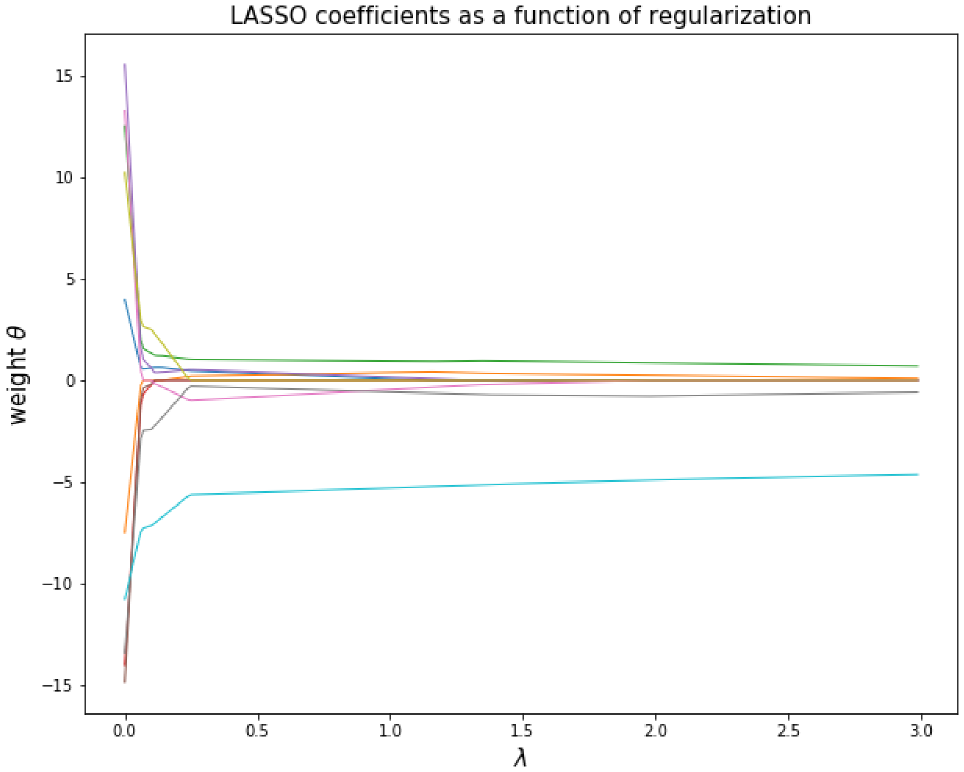


Ridge

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

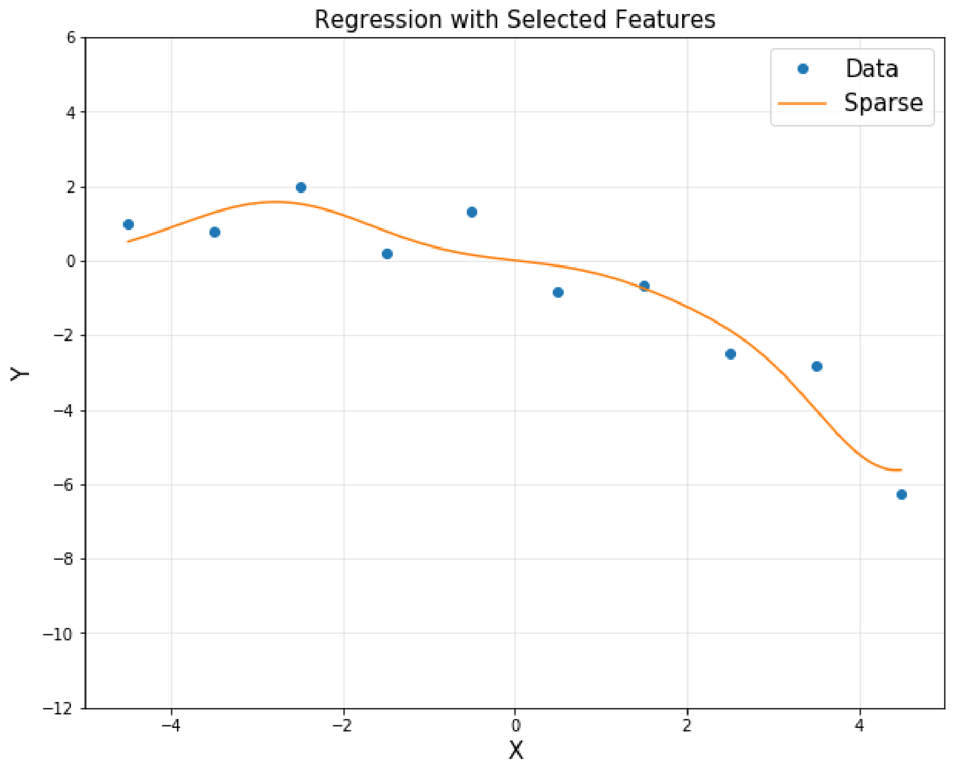
- Start with full model (all possible features)

- ‘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

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

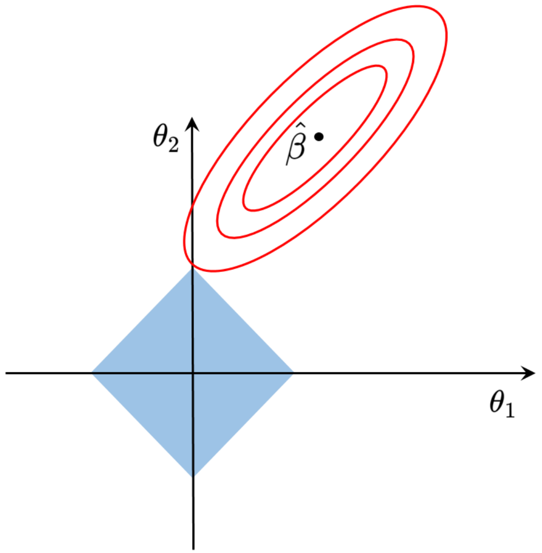
$$\Longrightarrow \quad \begin{aligned} \min_{\theta} \quad & \|\Phi\theta - y\|_2^2 \\ \text{subject to} \quad & \|\theta\|_1 \le s_1 \end{aligned} \qquad\qquad \begin{aligned} \min_{\theta} \quad & \|\Phi\theta - y\|_2^2 \\ \text{subject to} \quad & \|\theta\|_2 \le s_2 \end{aligned}$$

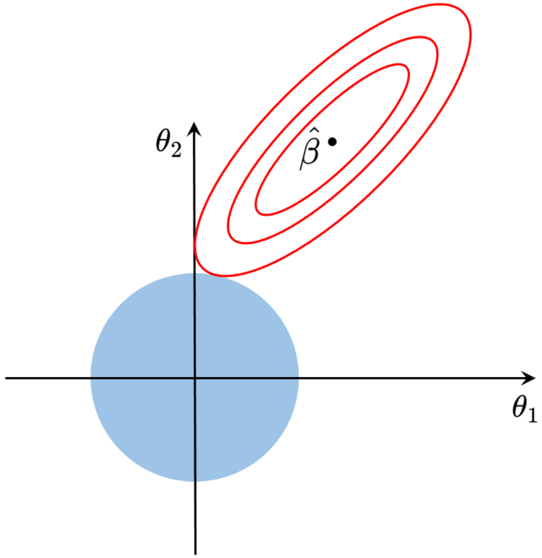
LASSO vs. Ridge

- Another equivalent forms of optimizations

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

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

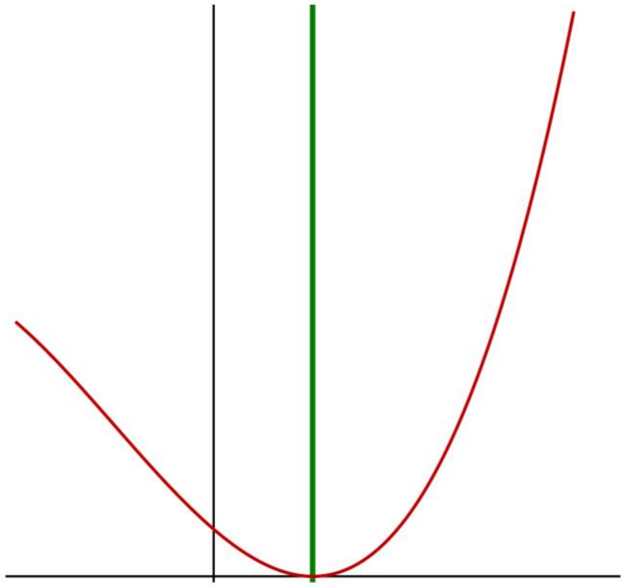




L2 Regularizers: Simple Example

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

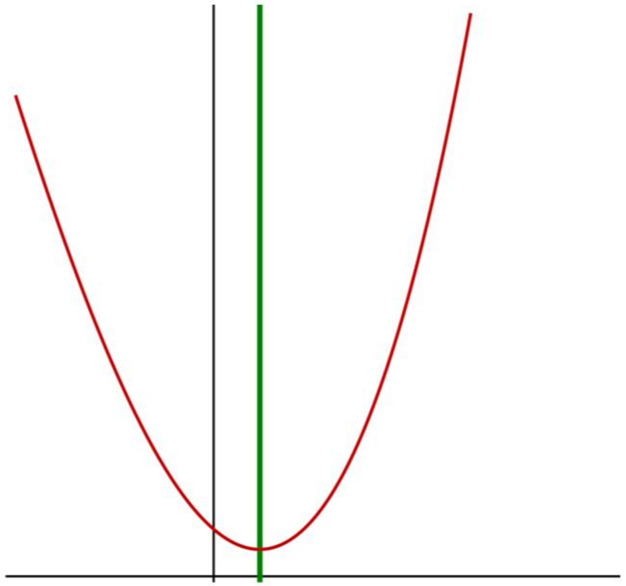
Since the derivative of $\|x\|_2^2$ is zero at zero, the optimal will never move there if it was not already there.



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

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

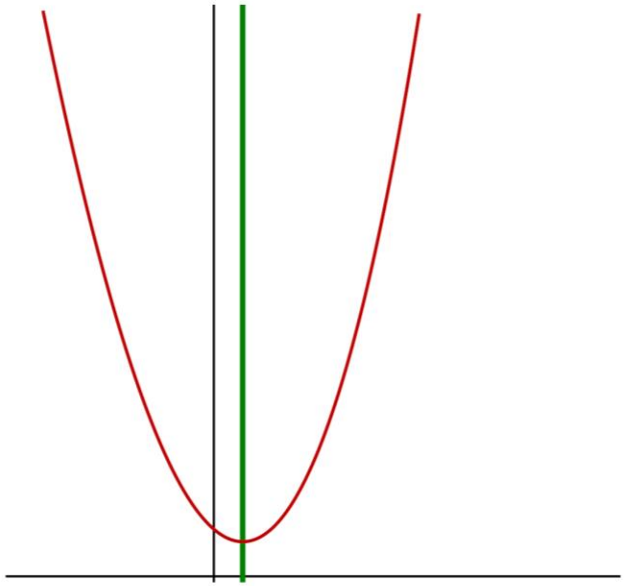
Since the derivative of $\|x\|_2^2$ is zero at zero, the optimal will never move there if it was not already there.



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

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

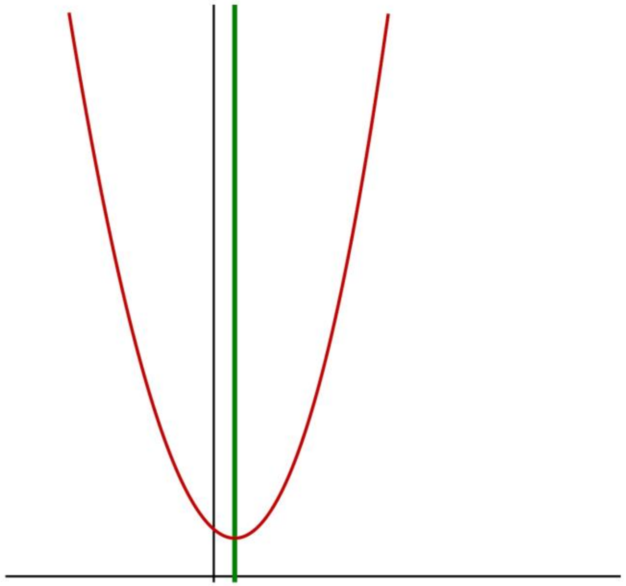
Since the derivative of $\|x\|_2^2$ is zero at zero, the optimal will never move there if it was not already there.



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

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

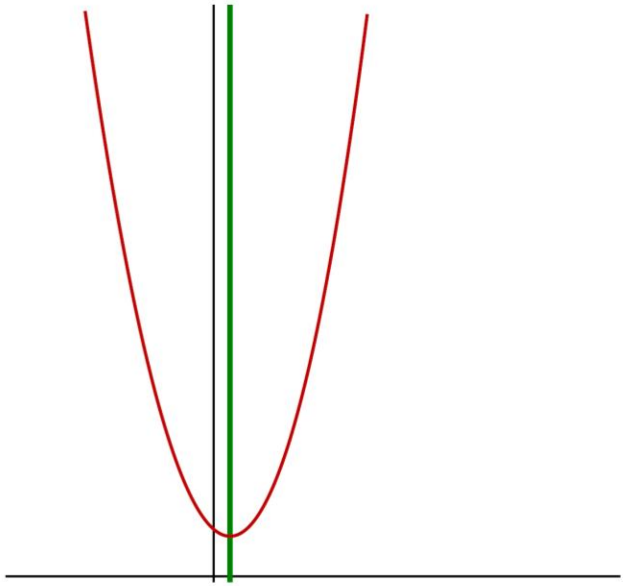
Since the derivative of $\|x\|_2^2$ is zero at zero, the optimal will never move there if it was not already there.



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

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

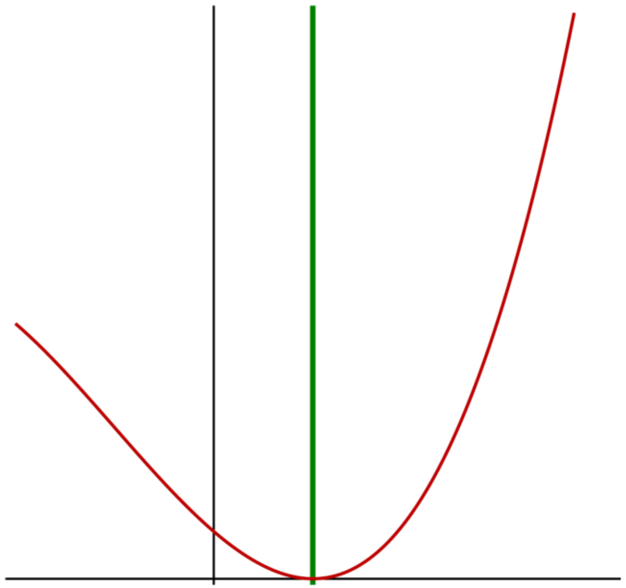
Since the derivative of $\|x\|_2^2$ is zero at zero, the optimal will never move there if it was not already there.



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

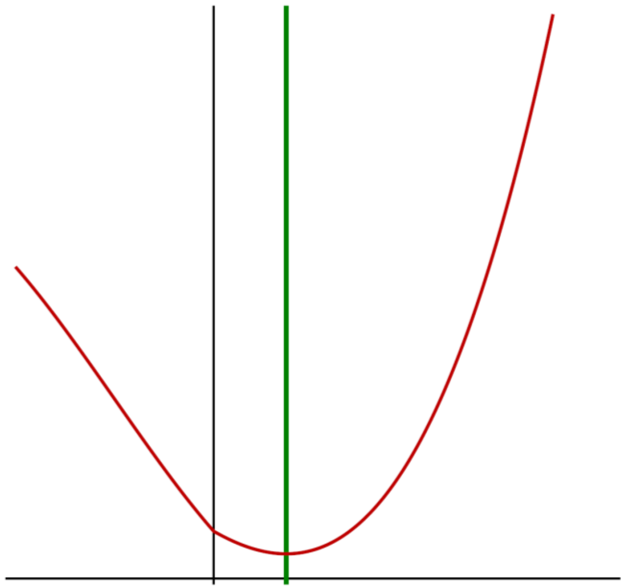
L1 Regularizers: Simple Example

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



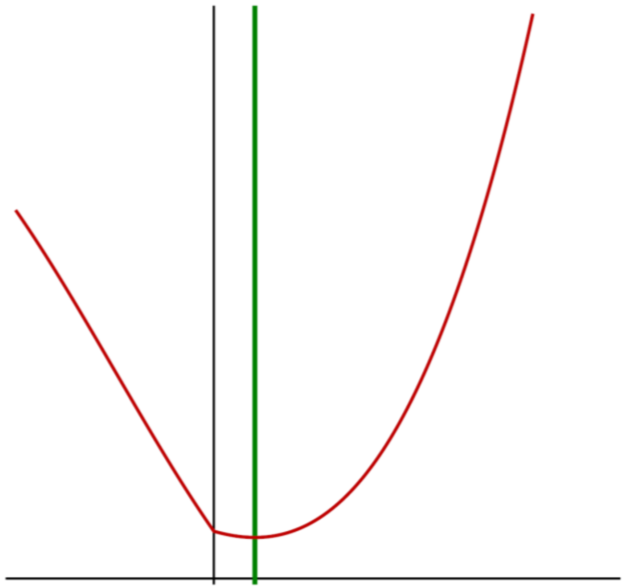
$$(x-1)^2 + \frac{1}{6}(x-1)^3$$

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



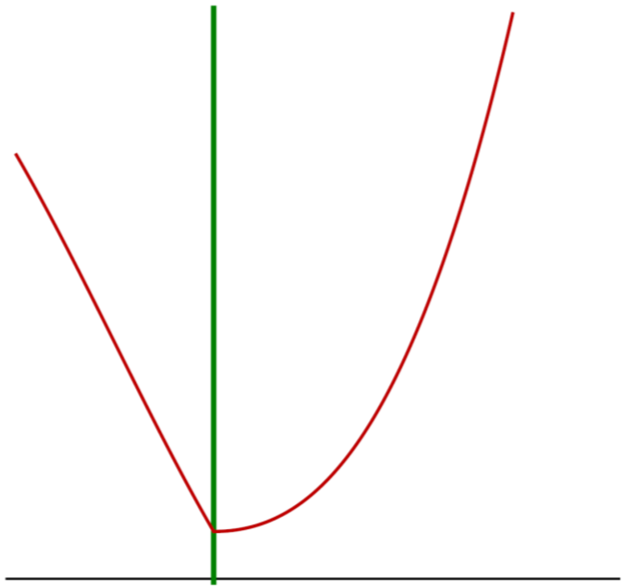
$$(x-1)^2 + \frac{1}{6}(x-1)^3 + \frac{1}{2}|x|$$

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



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

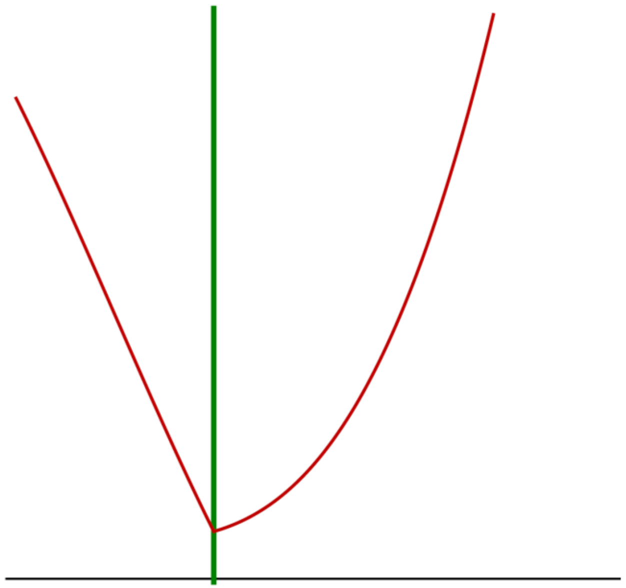
Increasing the λ parameter moves the optimal closer to 0, and away from the optimal for the loss without penalty.



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

POSTECH

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

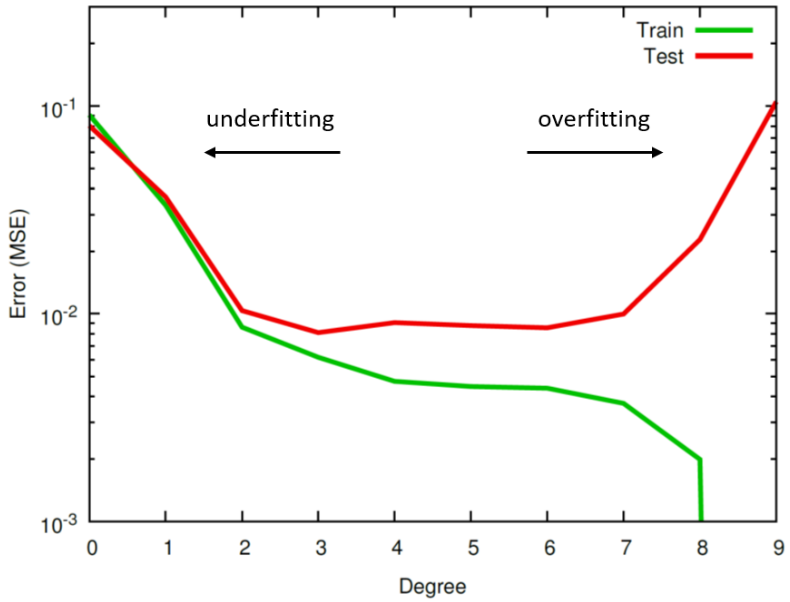


$$(x-1)^2 + \frac{1}{6}(x-1)^3 + 2|x|$$

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

Regression 3

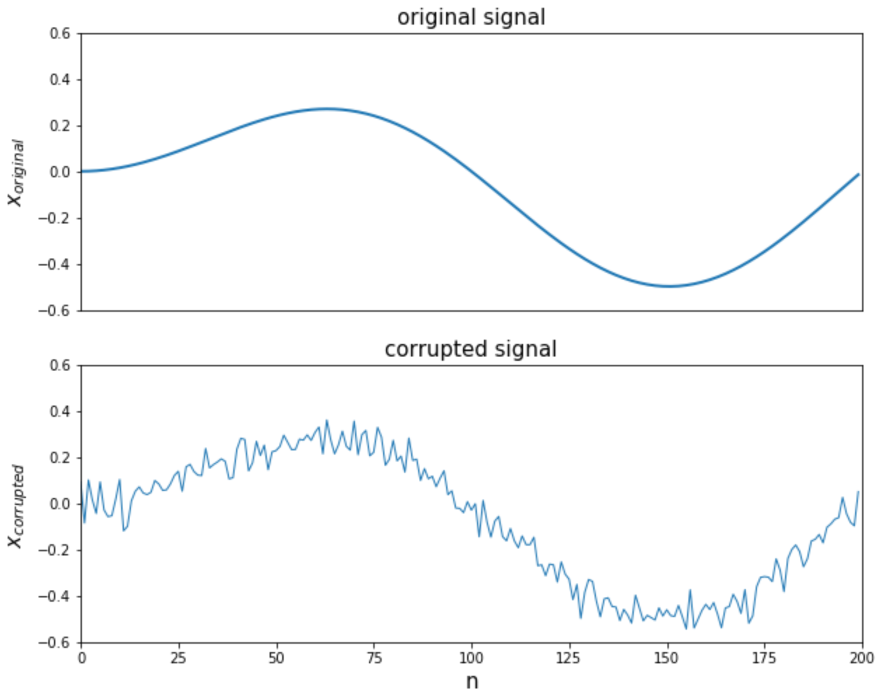
Industrial AI Lab.
Prof. Seungchul Lee

Linear Regression Examples

- De-noising
- Total Variation

De-noising Signal

- We start with a signal represented by a vector $x \in \mathbb{R}^n$
  - x_i corresponds to the value of some function of time, evaluated (or sampled) at evenly spaced points.
- Suppose x is corrupted by some small, rapidly varying noise ε,
  - i.e. $x_{cor} = x + \varepsilon$



Transform it to an Optimization Problem

- Transform de-noising in time into an optimization problem
- It is usually assumed that the signal does not vary too rapidly, which means that usually, we have $x_i \approx x_{i+1}$

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad \min_X \left\{ \underbrace{\|(X - X_{cor})\|_2^2}_{\text{how much } x \text{ deviates from } x_{cor}} + \mu \underbrace{\sum_{k=1}^{n-1} (x_{k+1} - x_k)^2}_{\text{penalize rapid changes of } X} \right\}$$

- μ
  – to adjust the relative weight of the first and second terms
  – to controls the "smoothness" of $\hat{x}$

Source:
- Boyd & Vandenberghe's book "Convex Optimization"
- http://cvxr.com/cvx/examples/ (Figures 6.8-6.10: Quadratic smoothing)
- Week 4 of Linear and Integer Programming by Coursera of Univ. of Colorado

Transform it to an Optimization Problem

$$\min_X \left\{ \underbrace{\|(X - X_{cor})\|_2^2}_{\text{how much } x \text{ deviates from } x_{cor}} + \mu \underbrace{\sum_{k=1}^{n-1} (x_{k+1} - x_k)^2}_{\text{penalize rapid changes of } X} \right\}$$

1) $X - X_{cor} = I_n X - X_{cor}$

2) $\sum (x_{k+1} - x_k)^2$

$$\Longrightarrow \left\{ \begin{array}{l} (x_2 - x_1) - 0 = [\,-1, \quad 1, \quad 0, \quad \cdots \quad 0\,] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} - 0 \\ (x_3 - x_2) - 0 = [\,0, \quad -1, \quad 1, \quad \cdots \quad 0\,] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} - 0 \\ \qquad \vdots \\ \Longrightarrow \left\| \underbrace{\begin{bmatrix} -1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -1 & 1 \end{bmatrix}}_{D} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}}_{X} \underbrace{-}_{-} \underbrace{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{0} \right\|_2^2 \end{array} \right.$$

Least-Square Problems

$$\min_X \left\{ \underbrace{\|(X - X_{cor})\|_2^2}_{\text{how much } x \text{ deviates from } x_{cor}} + \mu \underbrace{\sum_{k=1}^{n-1} (x_{k+1} - x_k)^2}_{\text{penalize rapid changes of } X} \right\}$$

$$\left\| I_n X - X_{cor} \right\|_2^2 + \mu \left\| DX - 0 \right\|_2^2 = \left\| Ax - b \right\|_2^2$$

$$= \left\| \begin{bmatrix} I_n \\ \sqrt{\mu} D \end{bmatrix} X - \begin{bmatrix} X_{cor} \\ 0 \end{bmatrix} \right\|_2^2 \qquad \text{where } A = \begin{bmatrix} I_n \\ \sqrt{\mu} D \end{bmatrix}, \quad b = \begin{bmatrix} X_{cor} \\ 0 \end{bmatrix}$$

- Then, plug A, b into Python to numerically solve
- Note: de-noising is generally conducted by a low pass filter in the frequency domain

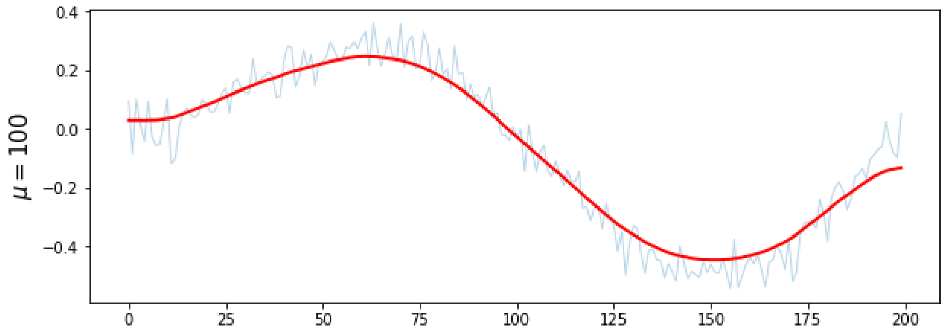
Coded in Python

$$\left\| \begin{bmatrix} I_n \\ \sqrt{\mu} D \end{bmatrix} X - \begin{bmatrix} X_{cor} \\ 0 \end{bmatrix} \right\|_2^2$$

$$\text{where } A = \begin{bmatrix} I_n \\ \sqrt{\mu} D \end{bmatrix}, \quad b = \begin{bmatrix} X_{cor} \\ 0 \end{bmatrix}$$

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

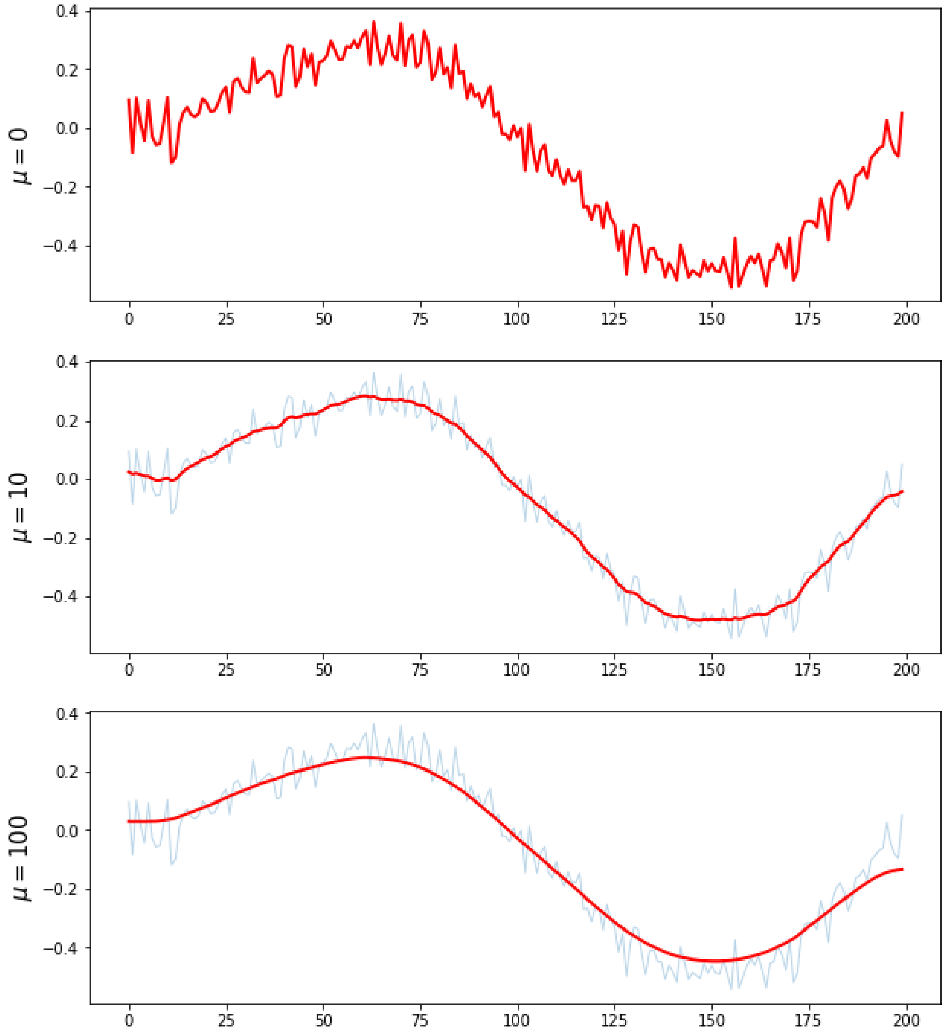
```
mu = 100

D = np.zeros([n-1, n])
D[:,0:n-1] -= np.eye(n-1)
D[:,1:n] += np.eye(n-1)
A = np.vstack([np.eye(n), np.sqrt(mu)*D])

b = np.vstack([x_cor, np.zeros([n-1,1])])

A = np.asmatrix(A)
b = np.asmatrix(b)

x_reconst = (A.T*A).I*A.T*b
```



See How μ Affects Smoothing Results

$$\min_{X} \left\{ \underbrace{\|(X - X_{cor})\|_2^2}_{\text{how much } x \text{ deviates from } x_{cor}} + \mu \underbrace{\sum_{k=1}^{n-1} (x_{k+1} - x_k)^2}_{\text{penalize rapid changes of } X} \right\}$$

```python
mu = [0, 10, 100];

for i in range(len(mu)):
    A = np.vstack([np.eye(n), np.sqrt(mu[i])*D])
    b = np.vstack([x_cor, np.zeros([n-1,1])])

    A = np.asmatrix(A)
    b = np.asmatrix(b)

    x_reconst = (A.T*A).I*A.T*b

    plt.subplot(3,1,i+1)
    plt.plot(t, x_cor, '-', linewidth = 1, alpha = 0.3)
    plt.plot(t, x_reconst, 'r', linewidth = 2)
    plt.ylabel('$\mu = {}$'.format(mu[i]), fontsize = 15)

plt.show()
```

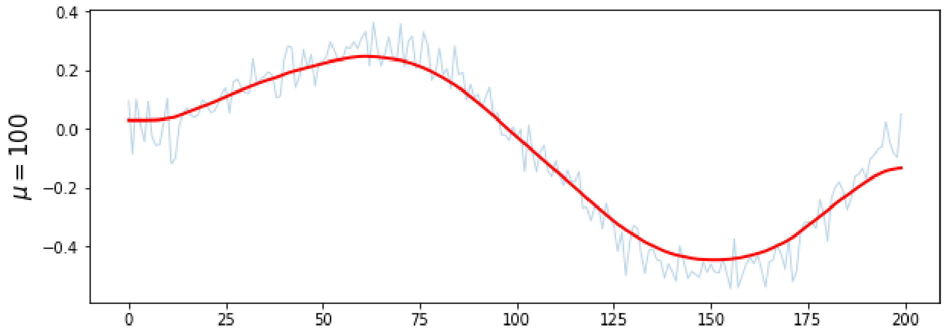


CVXPY Implementation

$$\min \; \left\{ \|x - x_{cor}\|_2^2 + \mu \|Dx\|_2^2 \right\}$$

$$\min_X \left\{ \underbrace{\|(X - X_{cor})\|_2^2}_{\text{how much } x \text{ deviates from } x_{cor}} + \mu \underbrace{\sum_{k=1}^{n-1} (x_{k+1} - x_k)^2}_{\text{penalize rapid changes of } X} \right\}$$

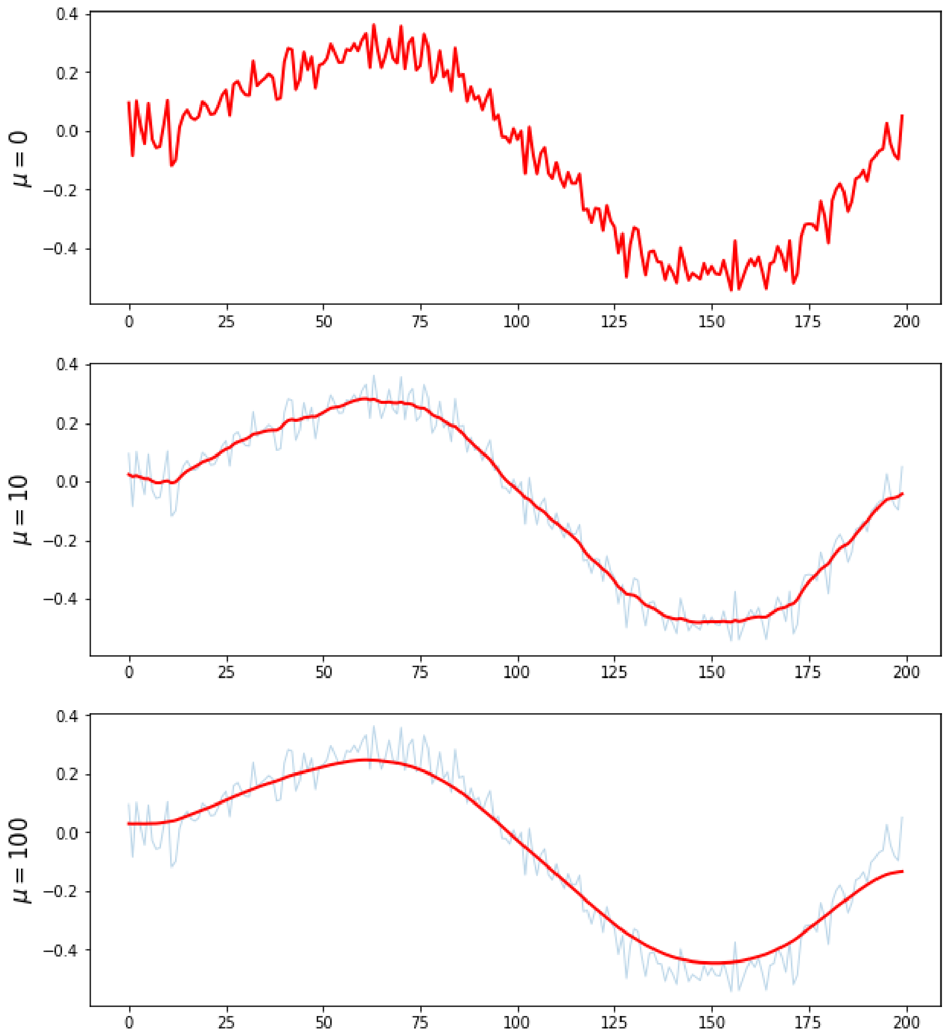
```
mu = 100

x_reconst = cvx.Variable([n,1])
#obj = cvx.Minimize(cvx.sum_squares(x_reconst-x_cor) + mu*cvx.sum_squares(x_reconst[1:n]-x_reconst[0:n-1]))
obj = cvx.Minimize(cvx.sum_squares(x_reconst-x_cor) + mu*cvx.sum_squares(D*x_reconst))

prob = cvx.Problem(obj).solve()
```



CVXPY: See How μ Affects Smoothing Results

$$\min \; \left\{ \|x - x_{cor}\|_2^2 + \mu \|Dx\|_2^2 \right\}$$

```
mu = [0, 10, 100]

for i in range(len(mu)):
    x_reconst = cvx.Variable([n,1])
    obj = cvx.Minimize(cvx.sum_squares(x_reconst - x_cor) +
                           mu[i]*cvx.sum_squares(D*x_reconst))
    prob = cvx.Problem(obj).solve()

    plt.subplot(3,1,i+1)
    plt.plot(t,x_cor,'-', linewidth = 1, alpha = 0.3)
    plt.plot(t,x_reconst.value, 'r', linewidth = 2)
    plt.ylabel('$\mu = {}$'.format(int(mu[i])), fontsize = 15)

plt.show()
```

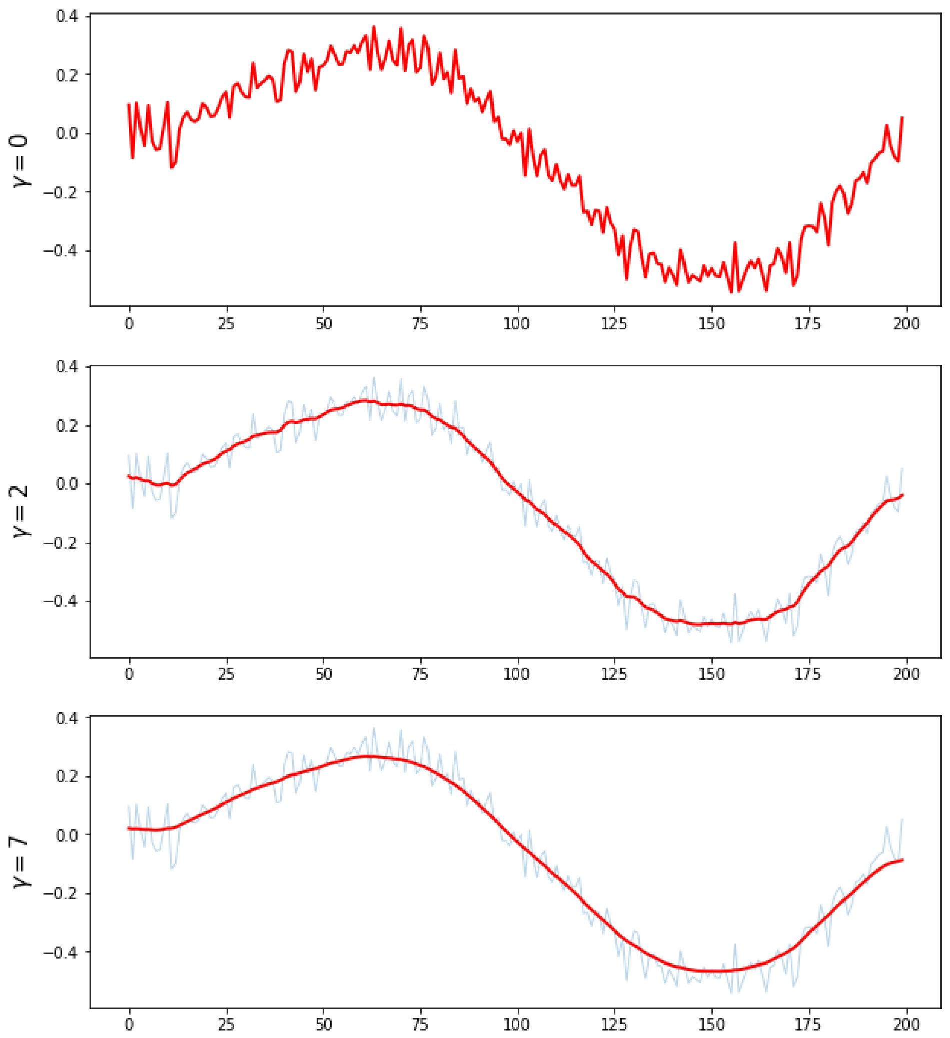


L_2 Norm

$$\min \; \{\|x - x_{cor}\|_2^2 + \mu\|Dx\|_2^2\}$$

$$\min \; \{\|x - x_{cor}\|_2 + \gamma\|Dx\|_2\}$$

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

gammas = [0, 2, 7]

for i in range(len(gammas)):
    x_reconst = cvx.Variable([n,1])
    obj = cvx.Minimize(cvx.norm(x_reconst-x_cor, 2) + gammas[i]*(cvx.norm(D*x_reconst, 2)))
    prob = cvx.Problem(obj).solve()

    plt.subplot(3,1,i+1)
    plt.plot(t,x_cor,'-', linewidth = 1, alpha = 0.3)
    plt.plot(t,x_reconst.value, 'r', linewidth = 2)
    plt.ylabel('$ \gamma = {}$'.format(gammas[i]), fontsize = 15)

plt.show()
```



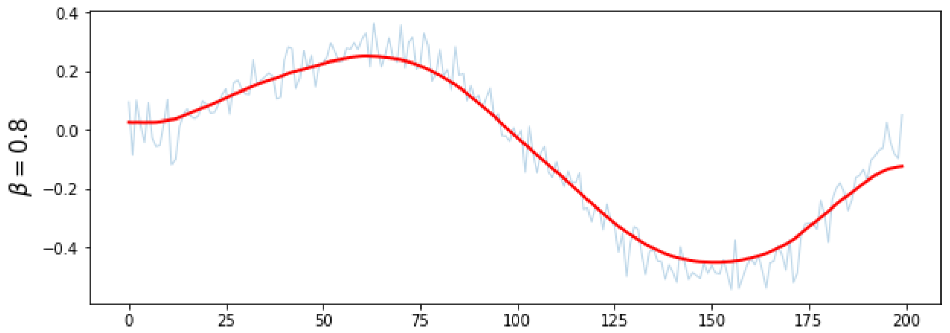
L_2 Norm with a Constraint

$$\min \ \{\|x - x_{cor}\|_2^2 + \mu\|Dx\|_2^2\}$$

$$\min \ \{\|x - x_{cor}\|_2 + \gamma\|Dx\|_2\}$$

$$\begin{aligned} &\min \ \|Dx\|_2 \\ s.t. \quad &\|x - x_{cor}\|_2 < \beta \end{aligned}$$

```
beta = 0.8

x_reconst = cvx.Variable([n,1])
obj = cvx.Minimize(cvx.norm(D*x_reconst, 2))
const = [cvx.norm(x_reconst-x_cor, 2) <= beta]
prob = cvx.Problem(obj, const).solve()

plt.figure(figsize=(10, 4))
plt.plot(t,x_cor,'-', linewidth = 1, alpha = 0.3)
plt.plot(t,x_reconst.value, 'r', linewidth = 2)
plt.ylabel(r'$\beta = {}$'.format(beta), fontsize = 15)
plt.show()
```

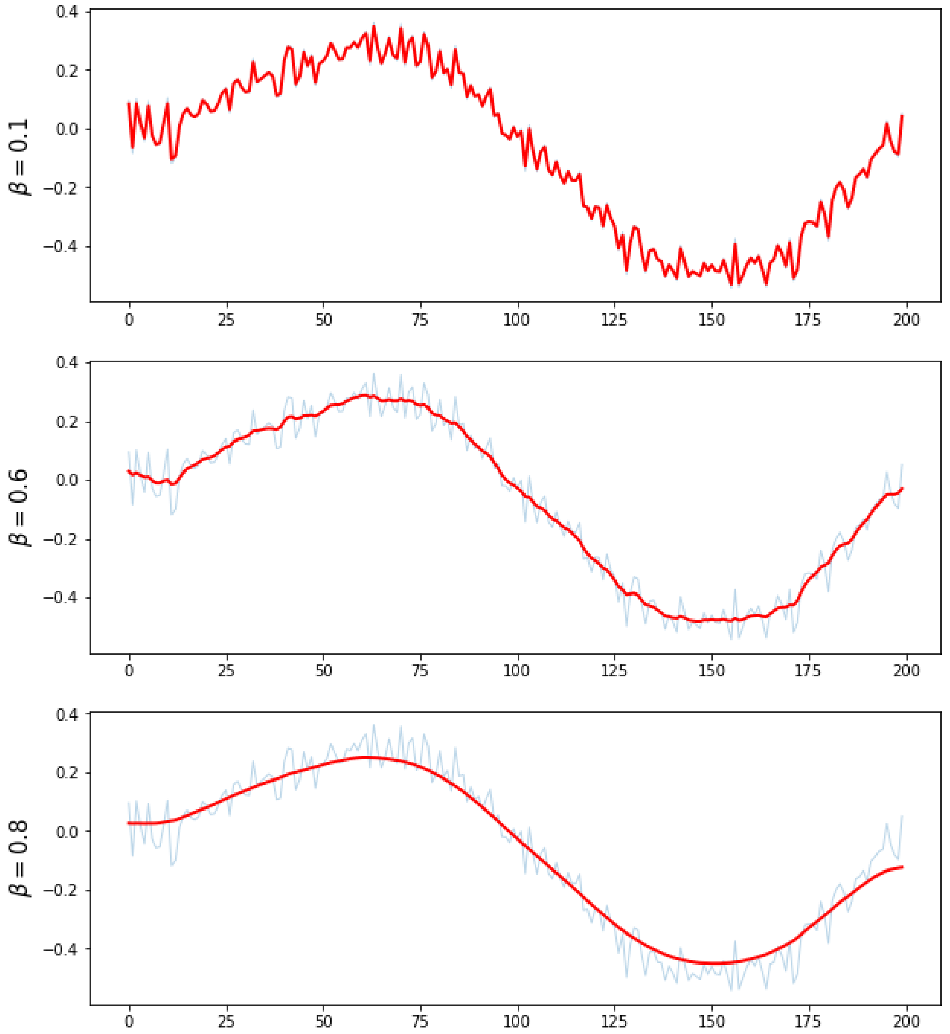


L_2 Norm with a Constraint

$$\min \; \{\|x - x_{cor}\|_2^2 + \mu\|Dx\|_2^2\}$$

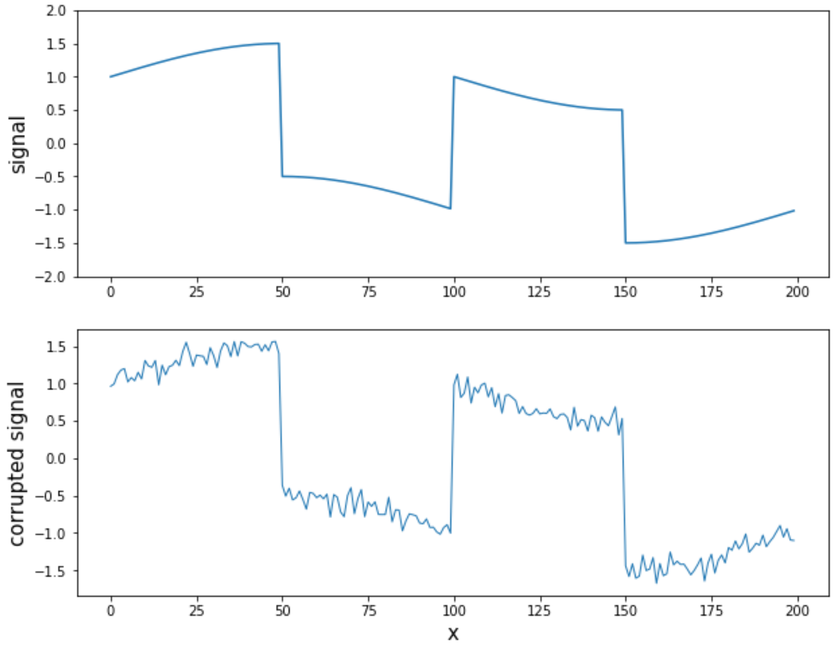
$$\min \; \{\|x - x_{cor}\|_2 + \gamma\|Dx\|_2\}$$

$$\begin{aligned} &\min \; \|Dx\|_2 \\ s.t. \quad &\|x - x_{cor}\|_2 < \beta \end{aligned}$$



Signal with Sharp Transition + Noise

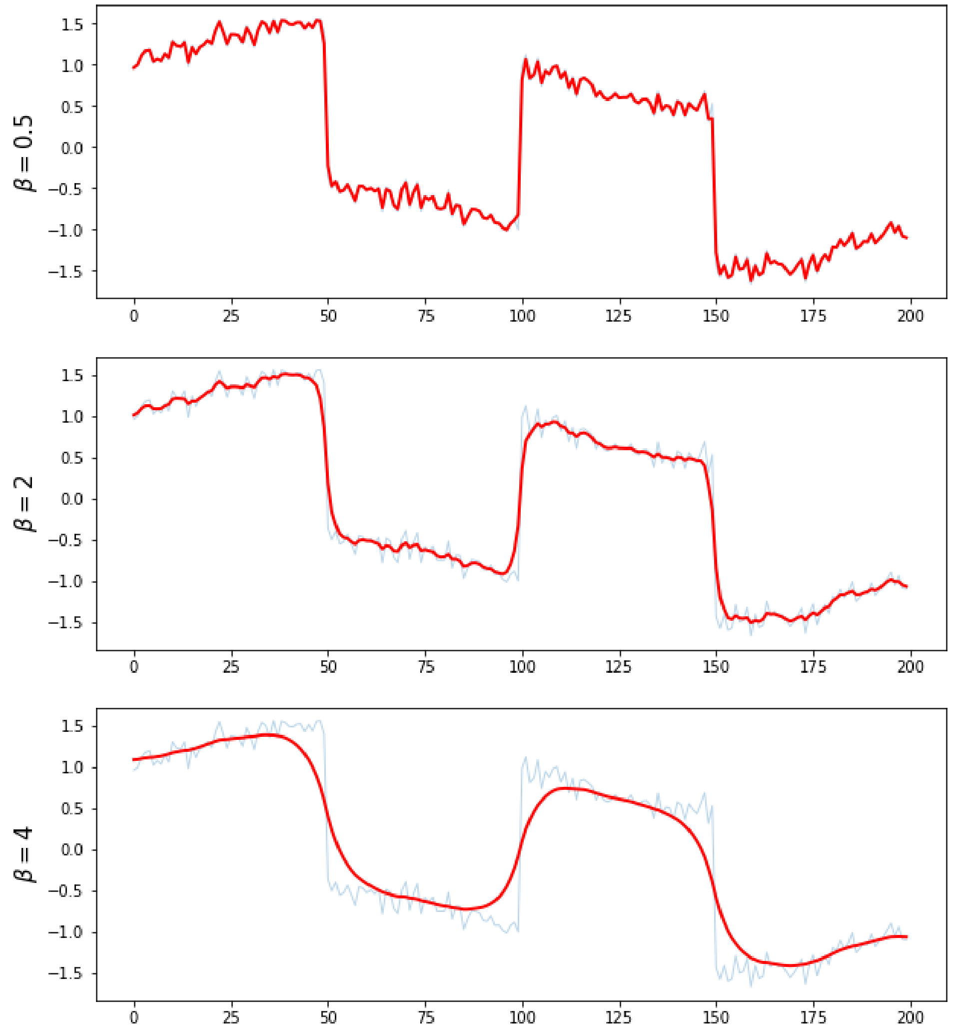
- Suppose we have a signal x, which is mostly smooth, but has several rapid variations (or jumps).



- First, apply the same method that we used for smoothing signals before
- known as a *total variation problem*

Quadratic Smoothing (L_2 Norm)

$$\begin{aligned} &\min \ \|Dx\|_2 \\ s.t. \quad &\|x - x_{cor}\|_2 < \beta \end{aligned}$$



- Quadratic smoothing smooths out both *noise and sharp transitions* in signal, but this is not what we want
- We will not be able to preserve the signal's sharp transitions.
- Any ideas ?

L_1 Norm

- We can instead apply total variation reconstruction on the signal by solving

$$\min \|x - x_{cor}\|_2 + \lambda \sum_{i=1}^{n-1} |x_{i+1} - x_i|$$

where the parameter λ controls the "smoothness" of x

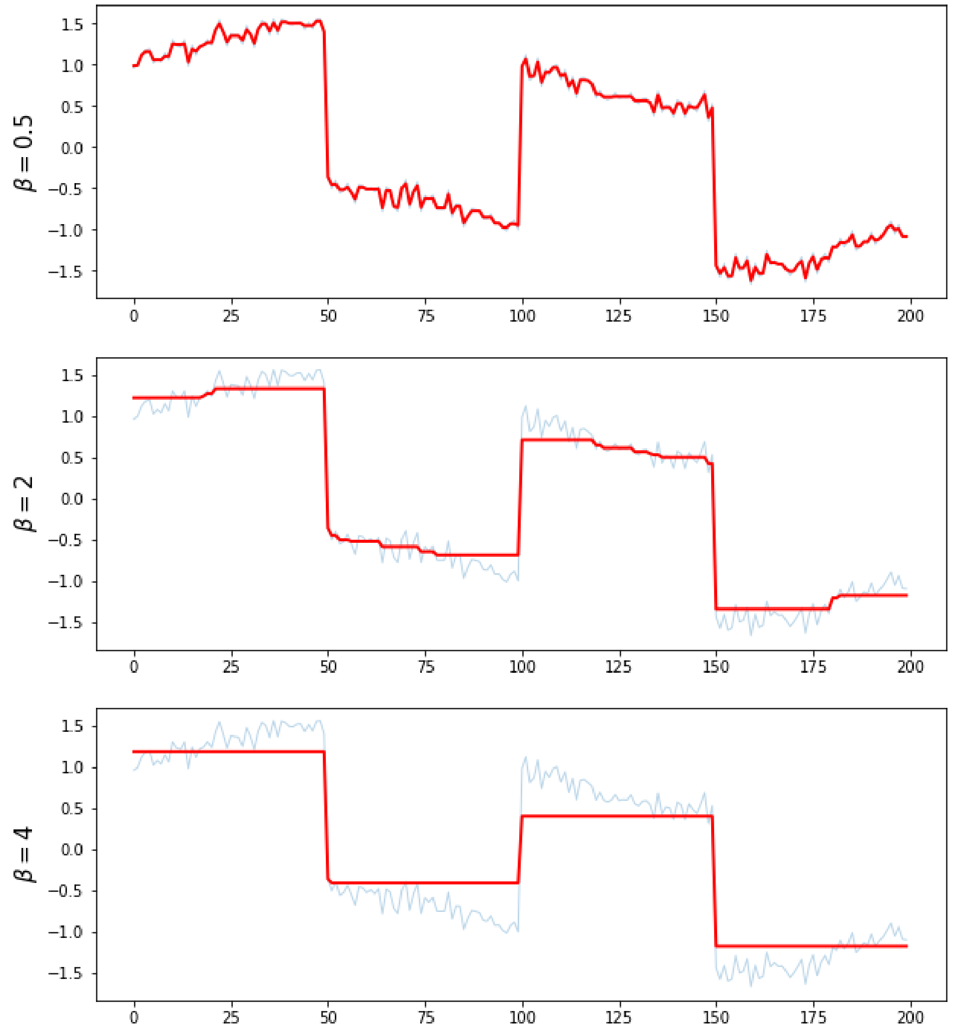
$$\begin{aligned} &\min \|Dx\|_2 \\ s.t. \quad &\|x - x_{cor}\|_2 < \beta \end{aligned} \quad \Longrightarrow \quad \begin{aligned} &\min \|Dx\|_1 \\ s.t. \quad &\|x - x_{cor}\|_2 < \beta \end{aligned}$$

L_1 Norm

$$\min \; \|Dx\|_1$$
$$s.t. \quad \|x - x_{cor}\|_2 < \beta$$

- Total Variation (TV) smoothing preserves sharp transitions in signal, and this is not bad

- Note that how TV reconstruction does a better job of preserving the sharp transitions in the signal while removing the noise.



Total Variation Image

- Q: Apply L_1 norm to the image, and guess what kind of an image will be produced ?

```
n = row*col
imbws = resized_imbw.reshape(-1, 1)
```

Idea comes from http://www2.compute.dtu.dk/~pcha/mxTV/

Total Variation Image

$$
\begin{aligned}
&\min \ \|Dx\|_1 \\
s.t. \quad &\|x - x_{cor}\|_2 < \beta
\end{aligned}
$$

```python
n = row*col
imbws = resized_imbw.reshape(-1, 1)

beta = 1500

x = cvx.Variable([n,1])
obj = cvx.Minimize(cvx.norm(x[1:n] - x[0:n-1],1))
const = [cvx.norm(x - imbws,2) <= beta]
prob = cvx.Problem(obj, const).solve()

imbwr = x.value.reshape(row, col)

plt.figure(figsize = (8,8))
plt.imshow(imbwr,'gray')
plt.axis('off')
plt.show()
```

Idea comes from http://www2.compute.dtu.dk/~pcha/mxTV/