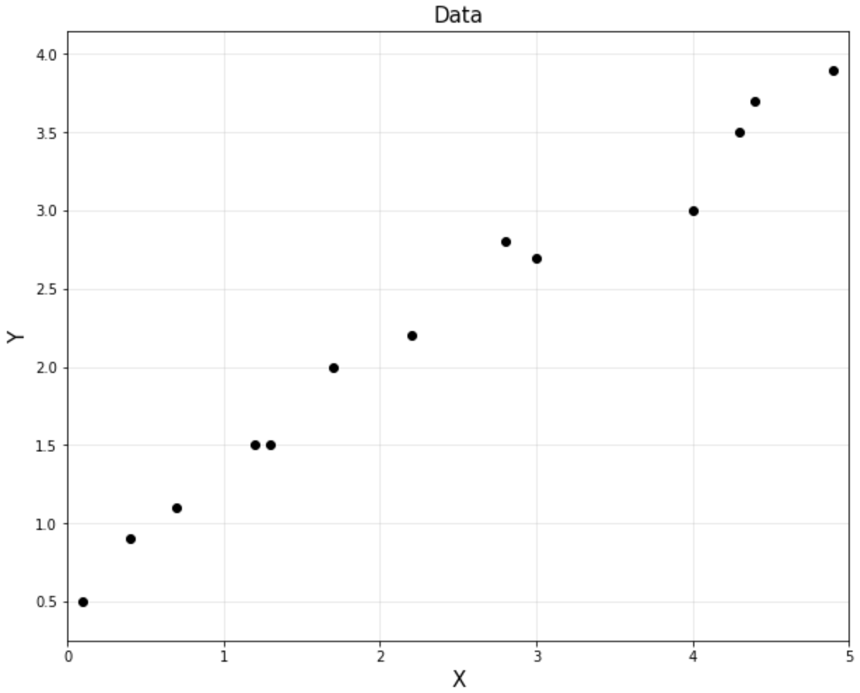
Regression

Prof. Seungchul Lee
Industrial AI Lab.

Assumption: Linear Model

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

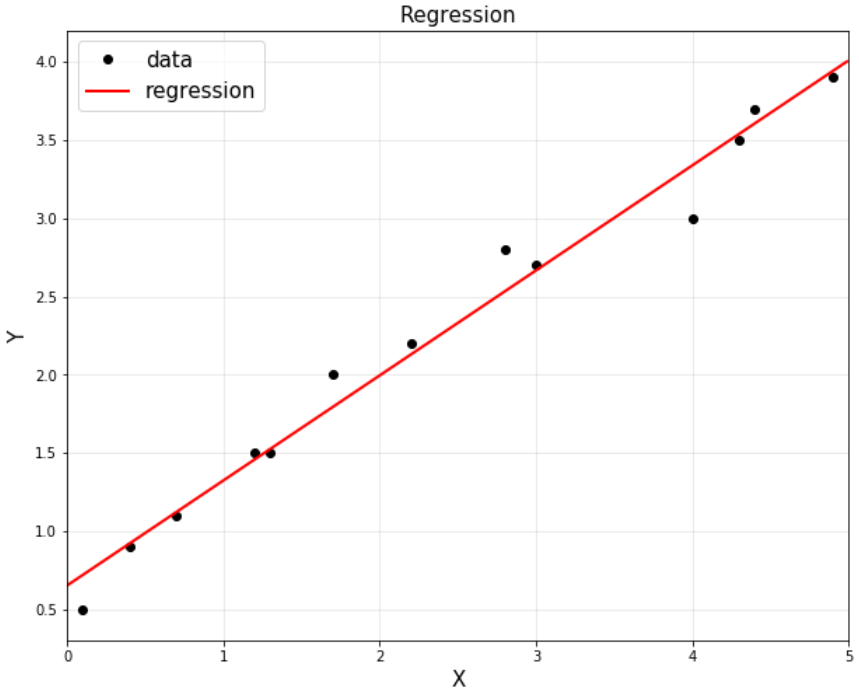


- In many cases, a linear model is used to predict y_i

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

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

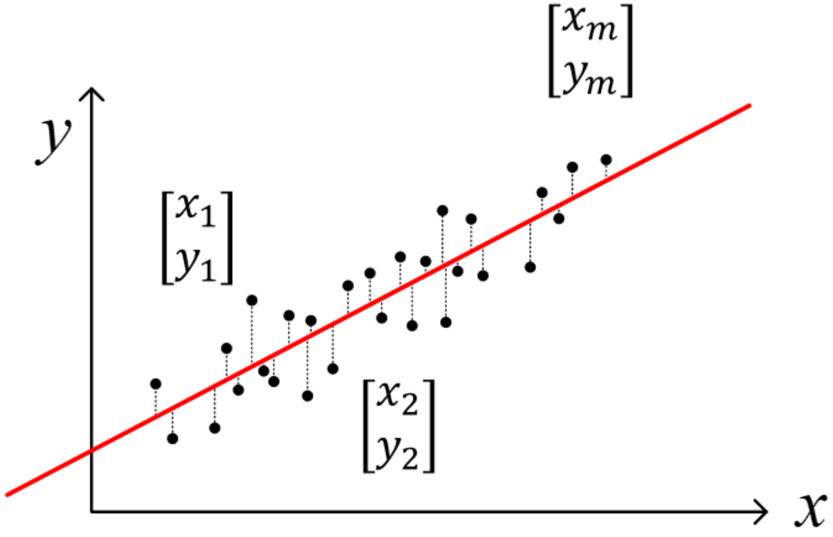
$$\hat{y}_i = \theta_1 x_i + \theta_2$$

Linear Regression

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

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

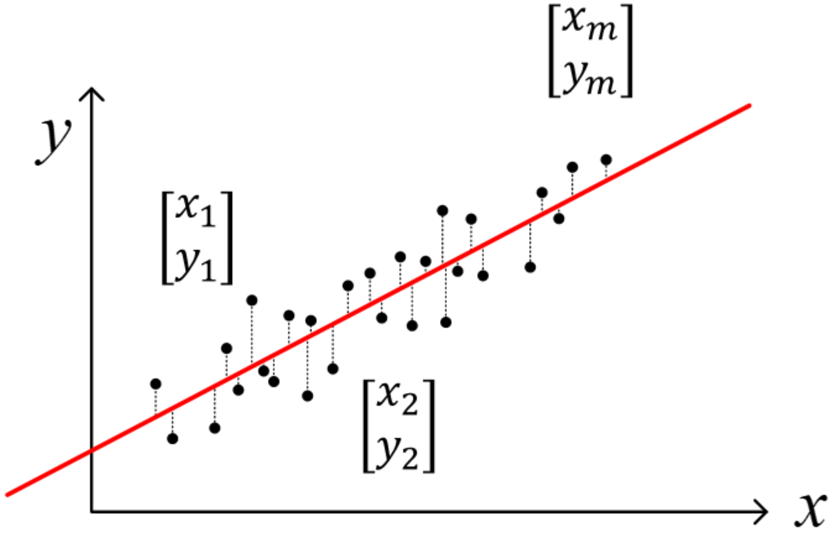
$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \approx \hat{y}_i = \theta_0 + \theta_1 x_i$$



- $\hat{y}_i$: 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

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

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

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

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

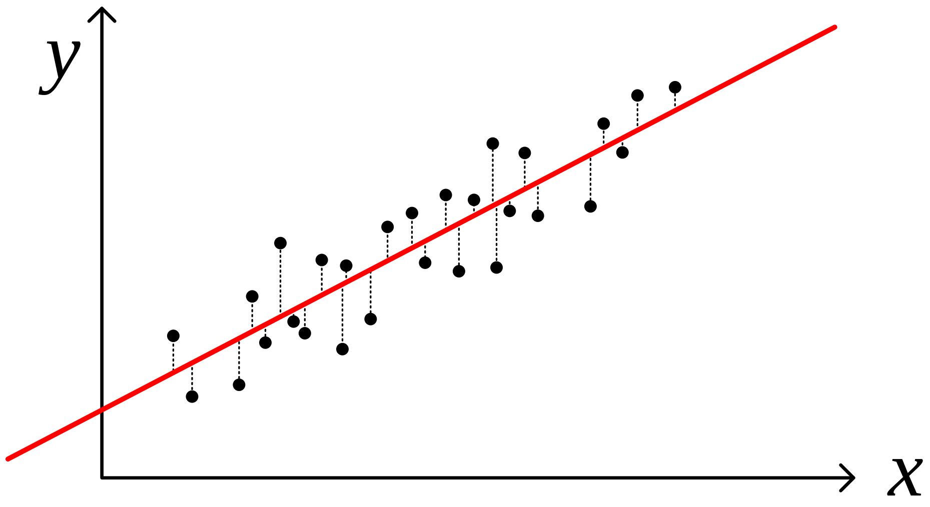
$$\text{feature vector } \phi(x_i) = \begin{bmatrix} 1 \\ x_i \end{bmatrix}$$

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

Optimization

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

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



Solve using Gradient Descent

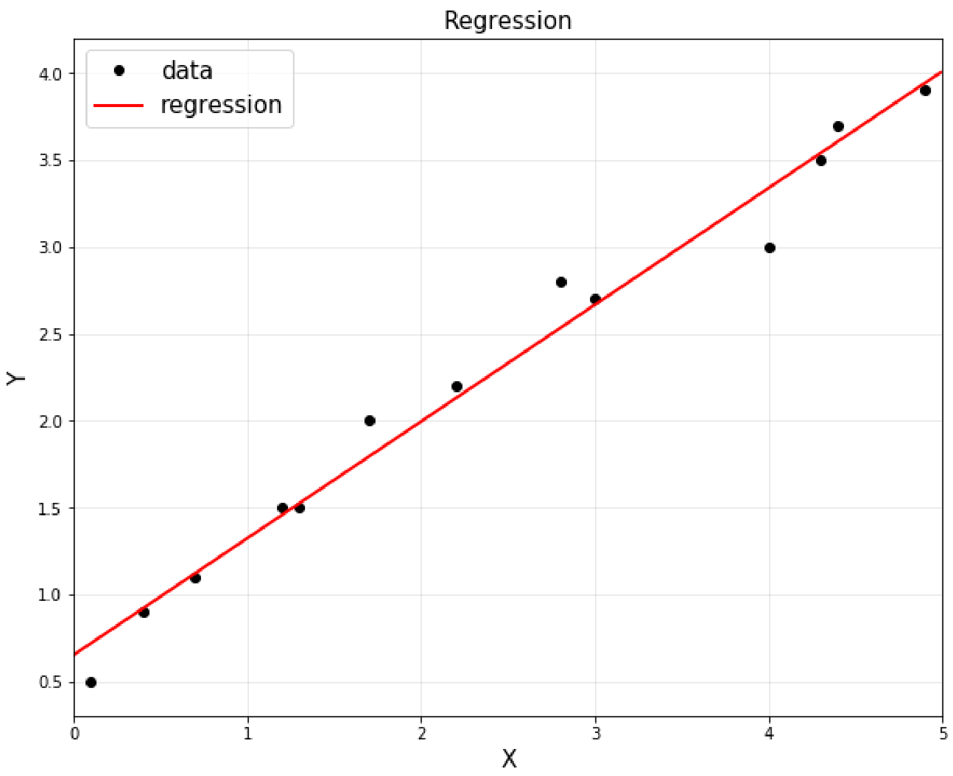
$$\min_{\theta} \; \|\hat{y} - y\|_2^2 = \min_{\theta} \; \|A\theta - y\|_2^2$$

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

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

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

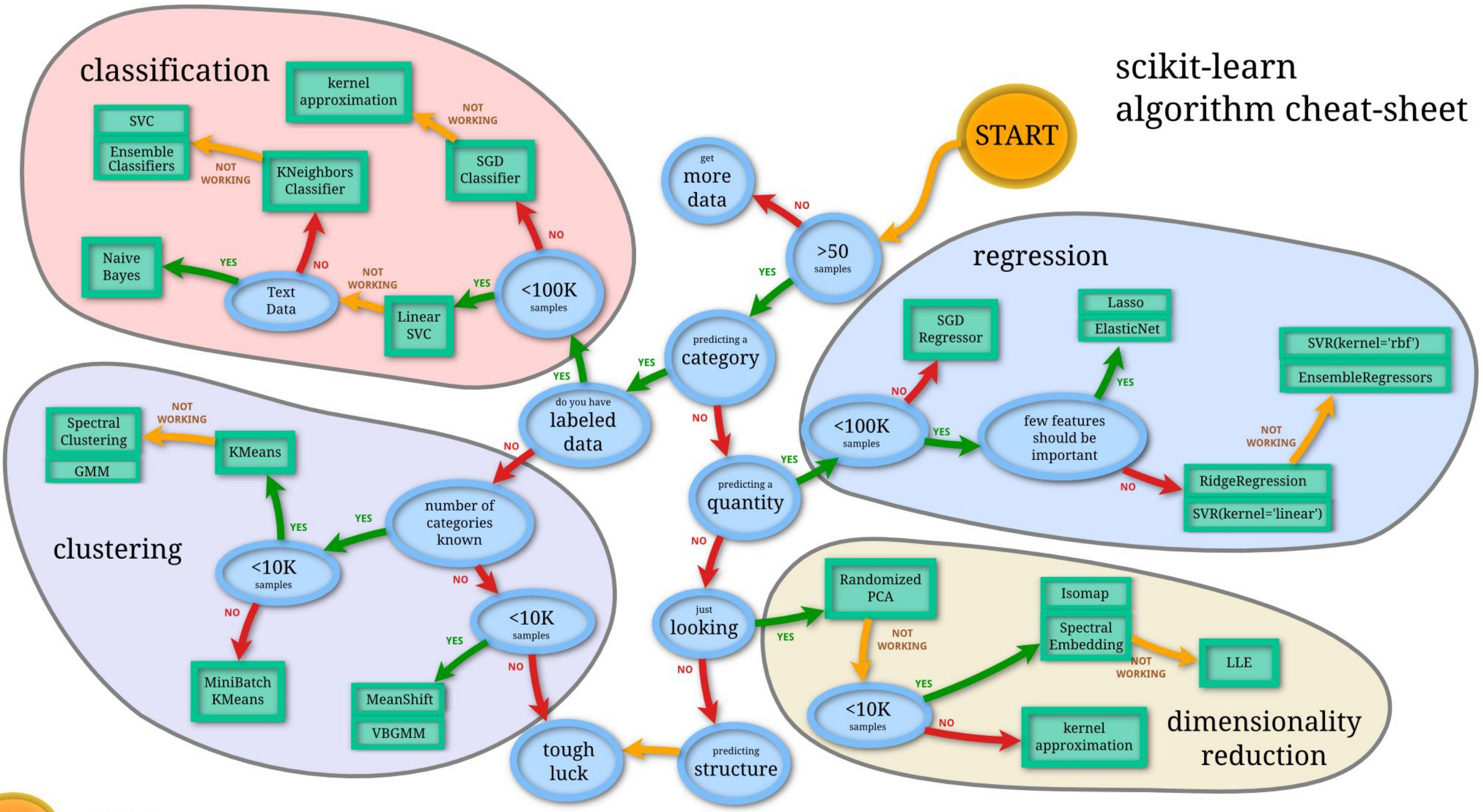
```
theta = np.random.randn(2,1)
theta = np.asmatrix(theta)

alpha = 0.001

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

print (theta)
```



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

```python
from sklearn import linear_model
```

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

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

```python
reg.coef_
```

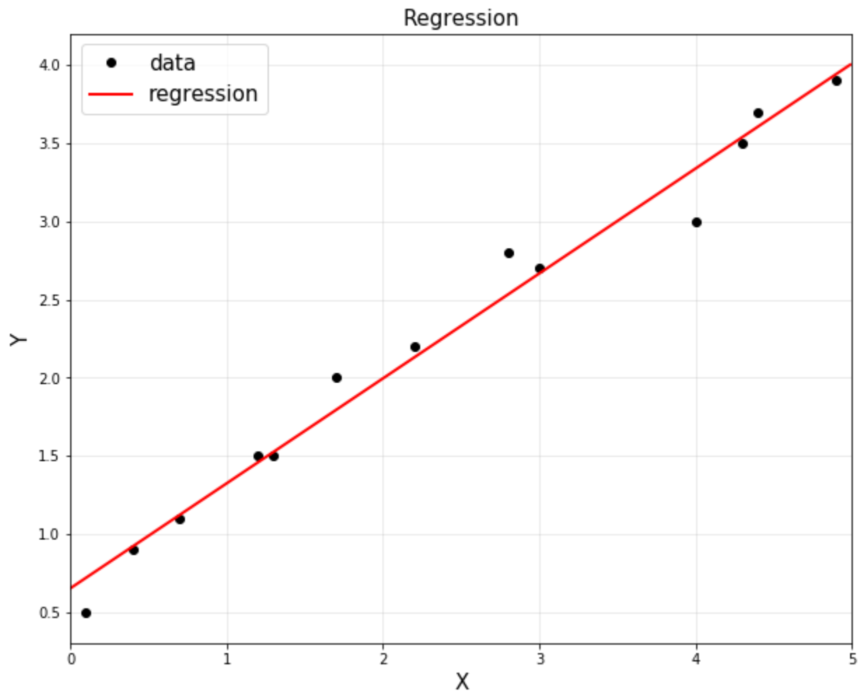
```
array([[0.67129519]])
```

```python
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()
```



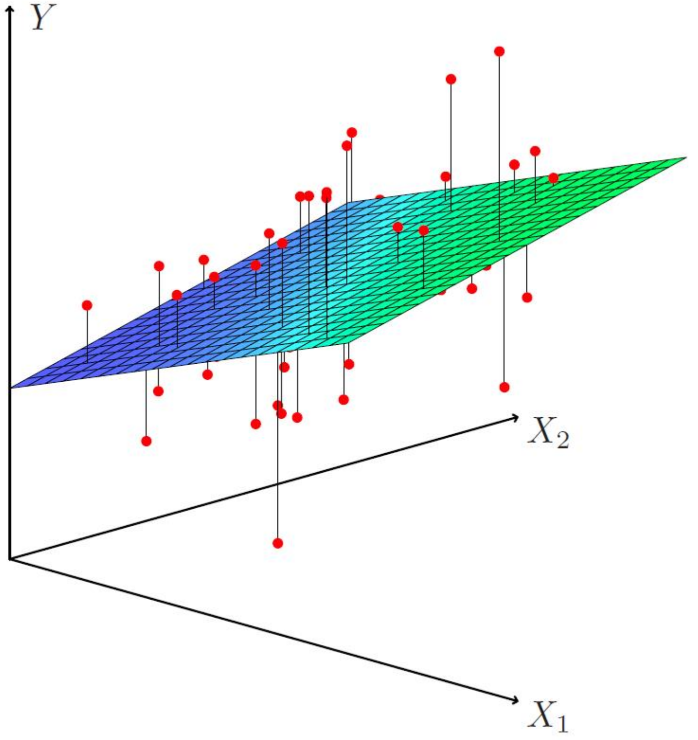
Multivariate Linear Regression

- Linear regression for multivariate data

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

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

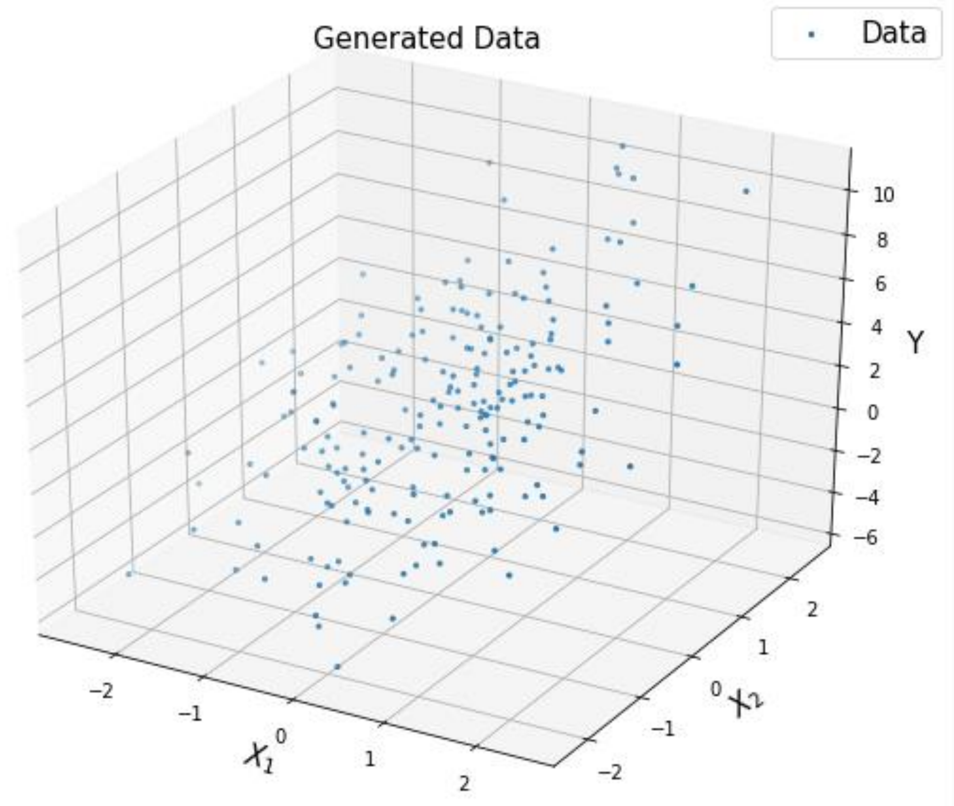
$$\Phi = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \\ \vdots & & \\ 1 & x_1^{(m)} & x_2^{(m)} \end{bmatrix} \quad \Longrightarrow \quad \hat{y} = \begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \vdots \\ \hat{y}^{(m)} \end{bmatrix} = \Phi\theta$$



- Same in matrix representation

Multivariate Linear Regression

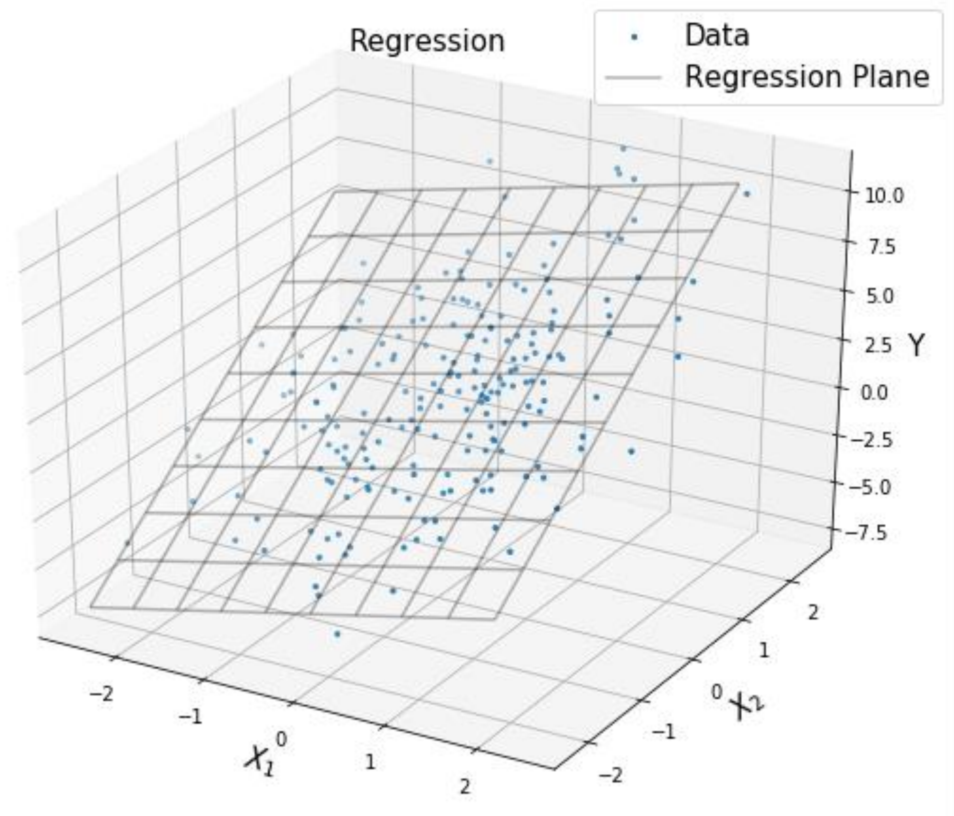
```python
# y = theta0 + theta1*x1 + theta2*x2 + noise

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
```



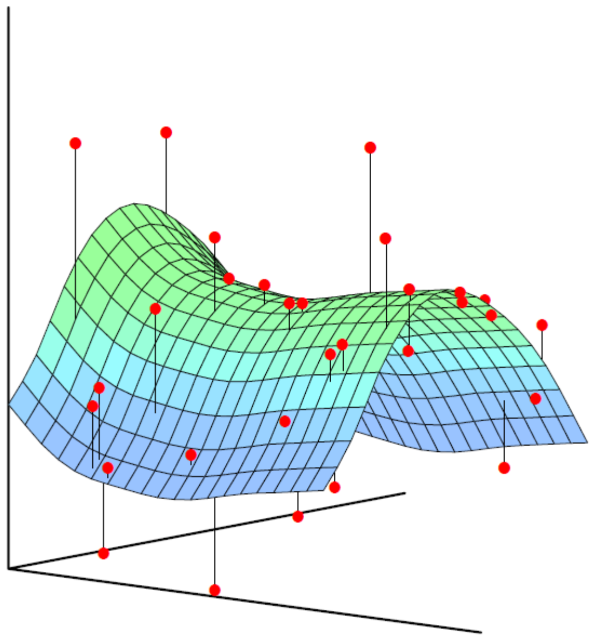
Multivariate Linear Regression

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

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



Nonlinear Regression



- 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)

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

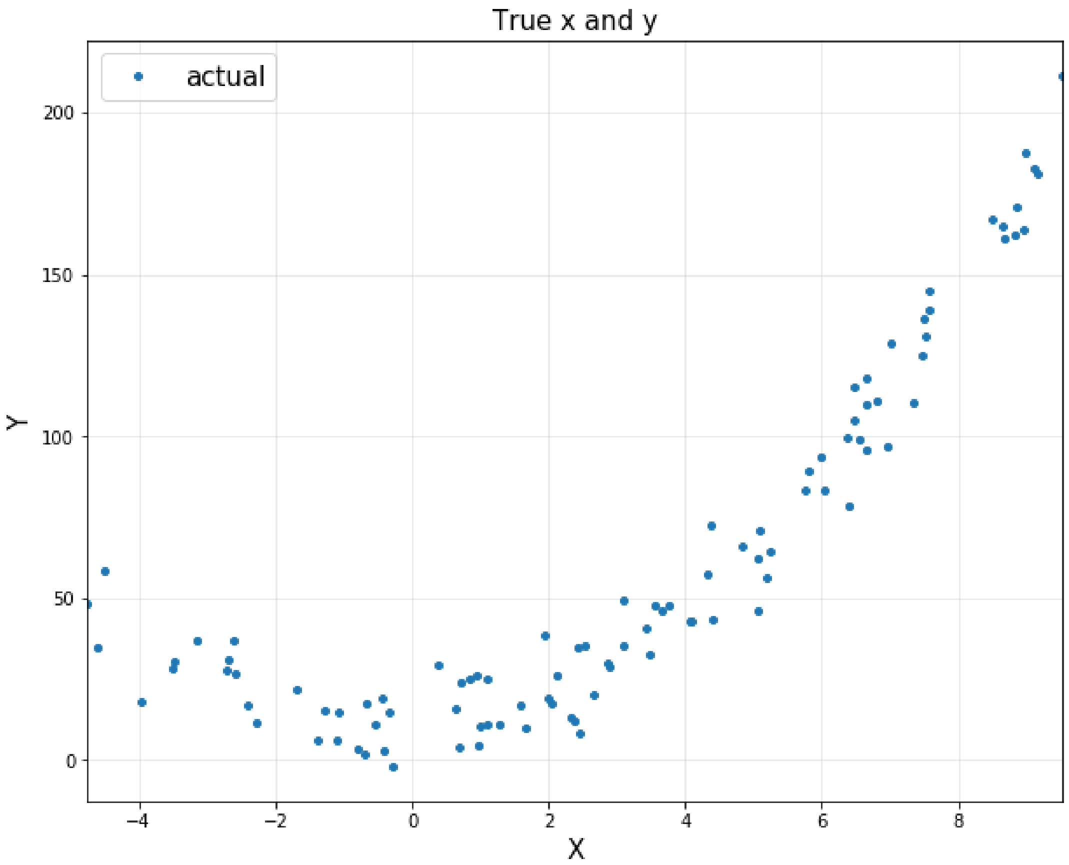
$$\phi(x_i) = \begin{bmatrix} 1 \\ x_i \\ x_i^2 \end{bmatrix}$$

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

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

Polynomial Regression

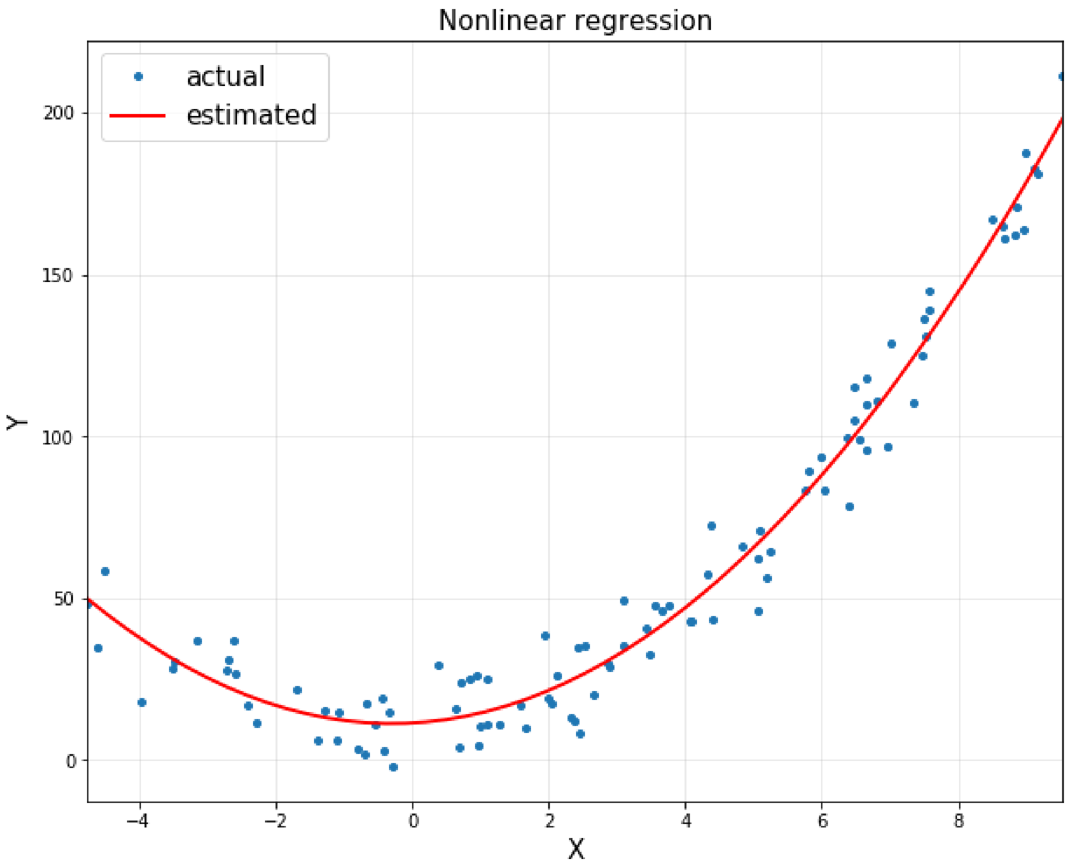
```python
# y = theta0 + theta1*x + theta2*x^2 + noise

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

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

```
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

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

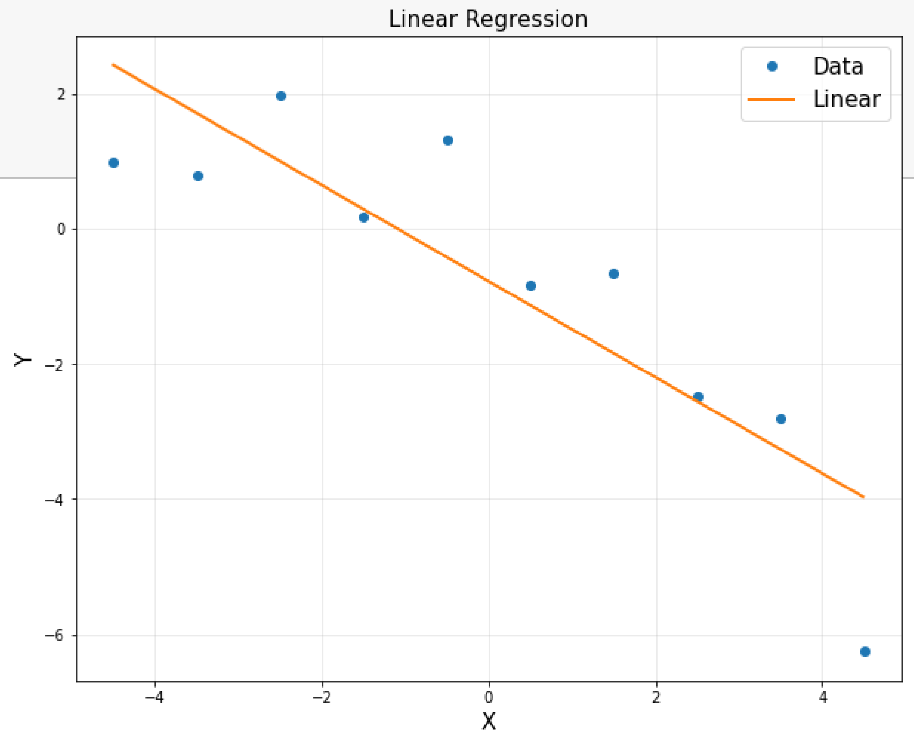
Overfitting

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)

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

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

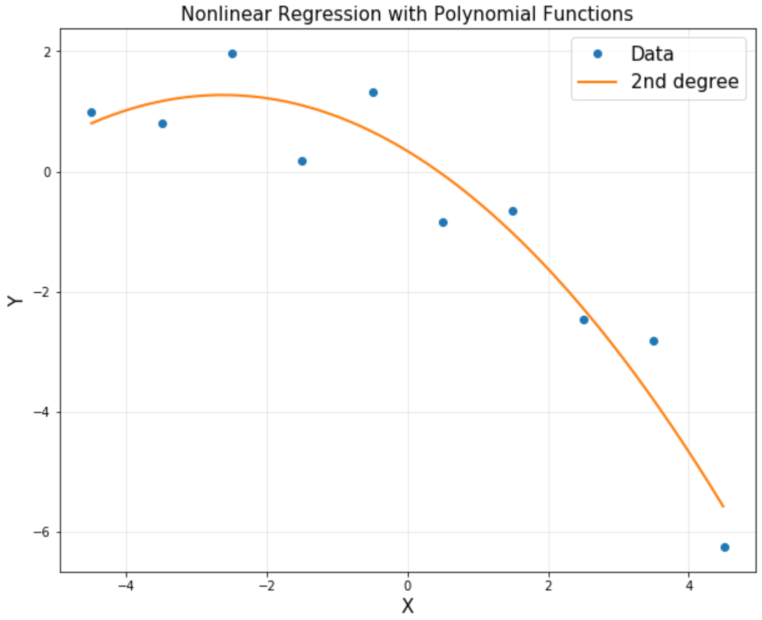
$$\Phi = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & & \\ 1 & x_m & x_m^2 \end{bmatrix} \implies \hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_m \end{bmatrix} = \Phi\theta$$

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

Nonlinear Regression

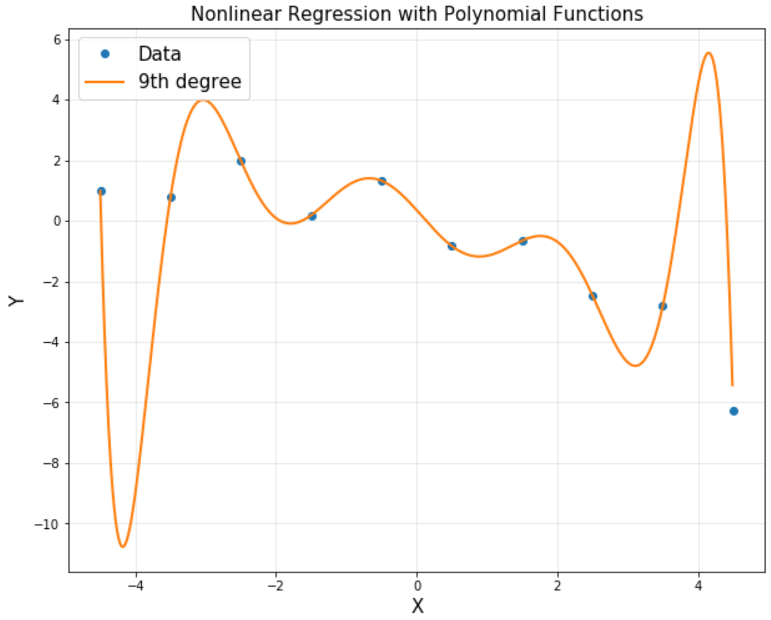
```python
A = np.hstack([x**0, x, x**2])
A = np.asmatrix(A)

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

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

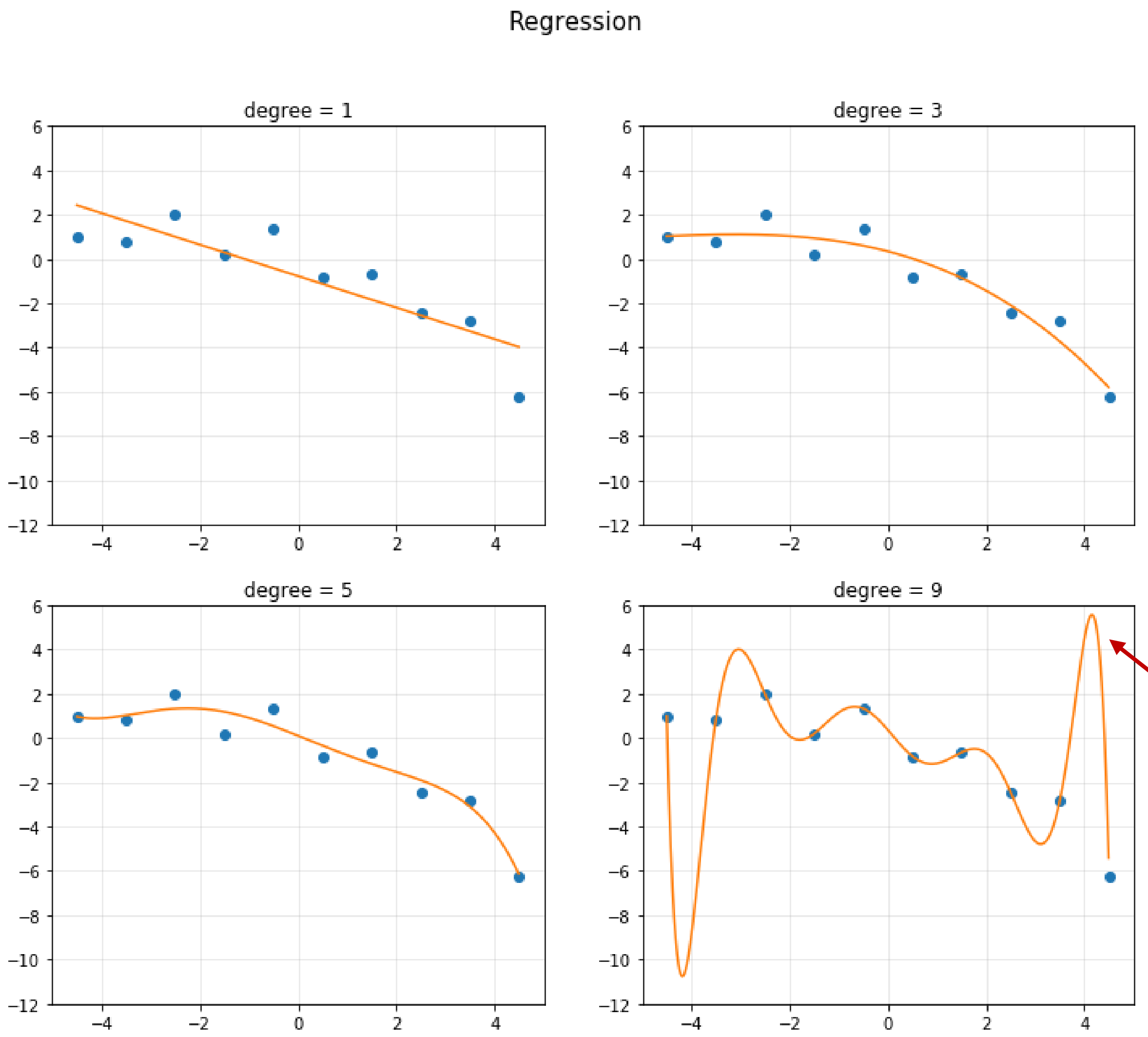



```python
A = np.hstack([x**i for i in range(10)])
A = np.asmatrix(A)

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

10 input points with degree 9 (or 10)



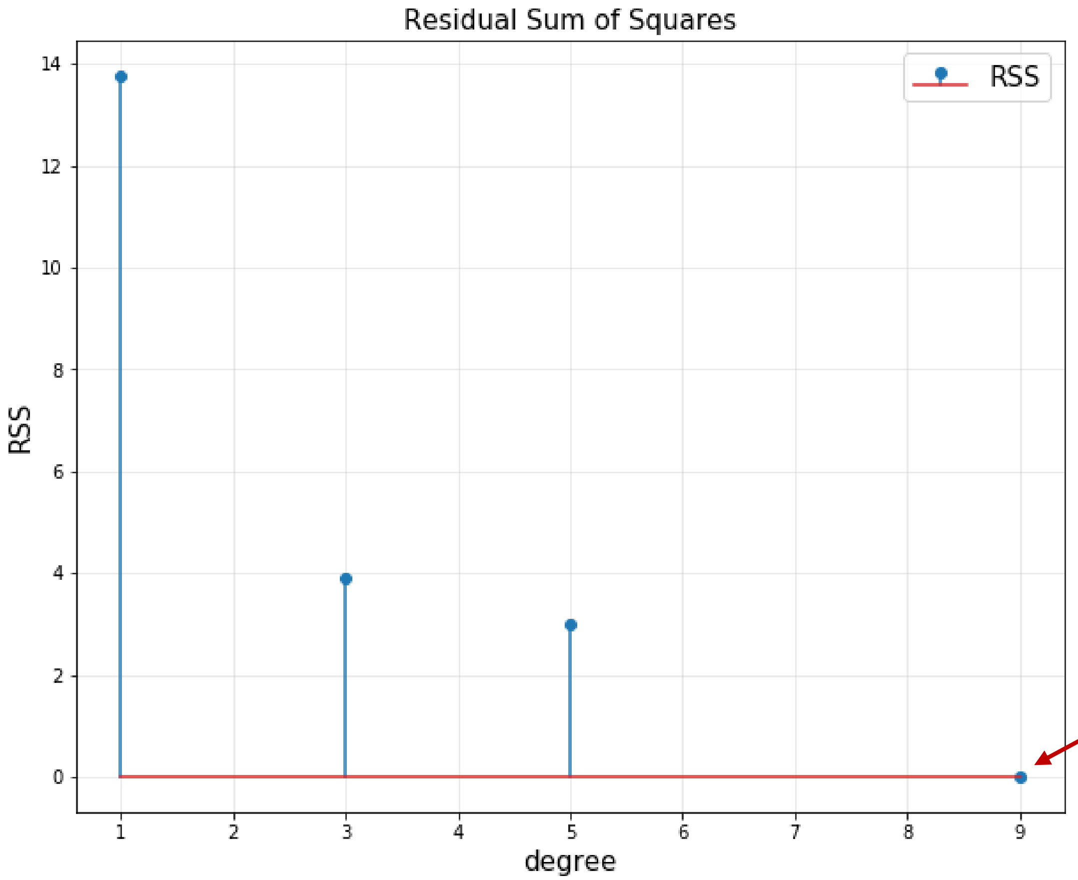
Polynomial Fitting with Different Degrees



Low error on input data points, but high error nearby

Loss

- Loss: Residual Sum of Squares (RSS)



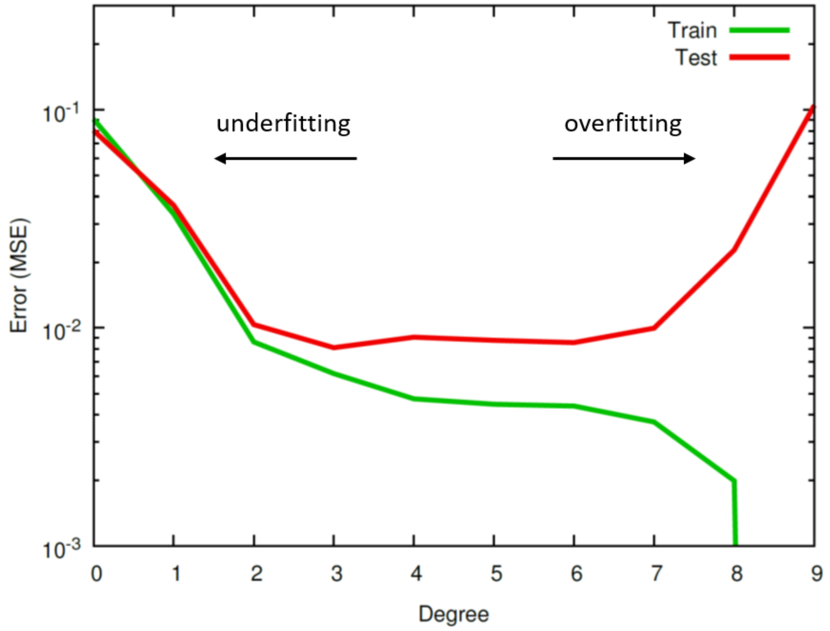
$$\min_{\theta} \|\hat{y} - y\|_2^2$$

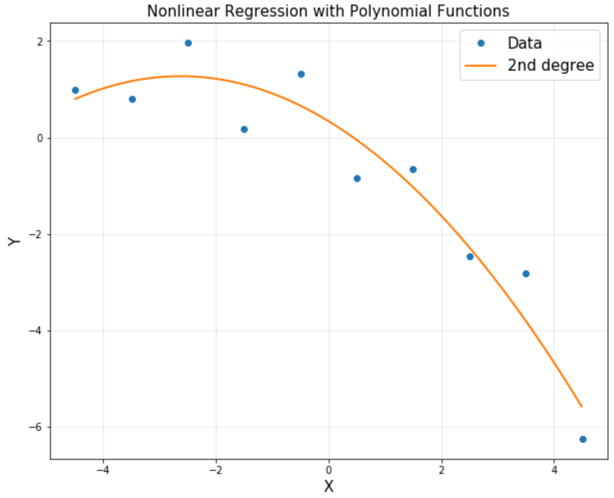
Minimizing loss in training data is often not the best

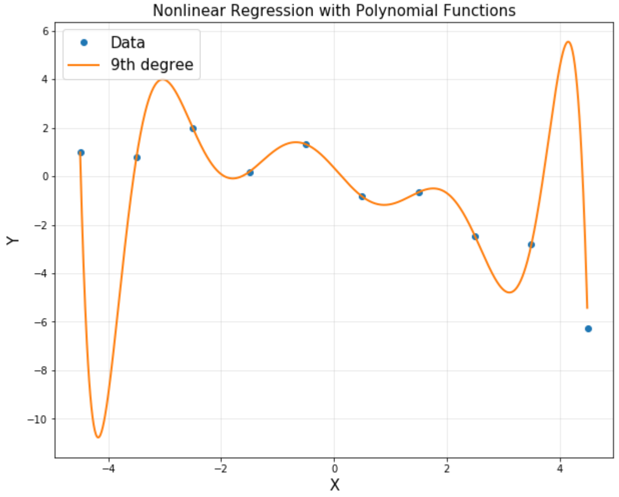
Low error on input data points, but high error nearby

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







POSTECH