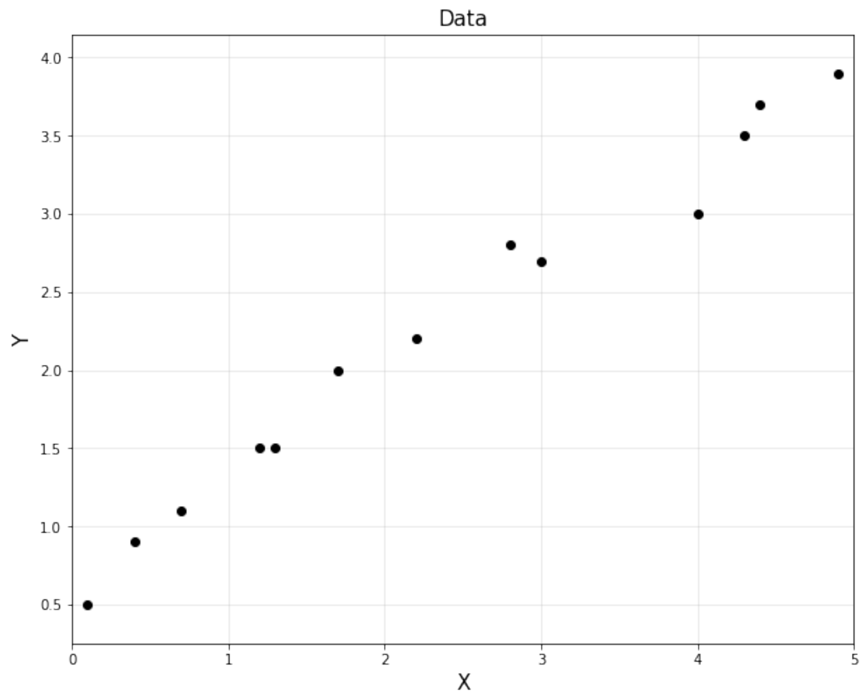
Regression

Prof. Seungchul Lee
Industrial AI Lab.

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

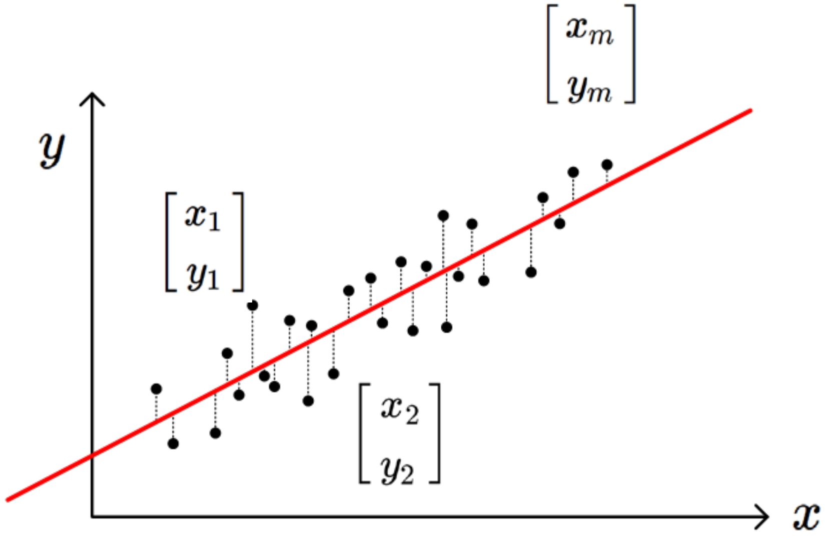
$$\hat{y}_i = \theta_1 x_i + \theta_2$$

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

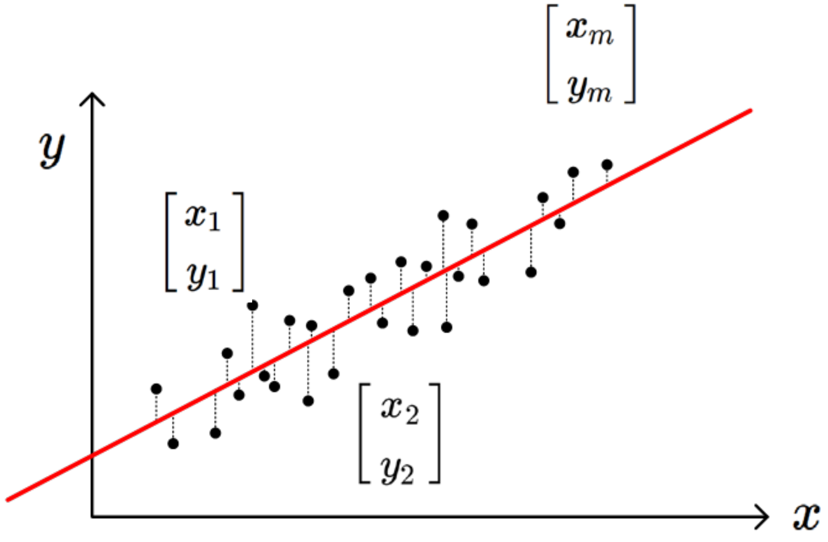
$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \approx \hat{y}_i = \theta_0 + \theta_1 x_i$$



- $\hat{y}_i$: 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

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

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

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

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

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

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

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

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

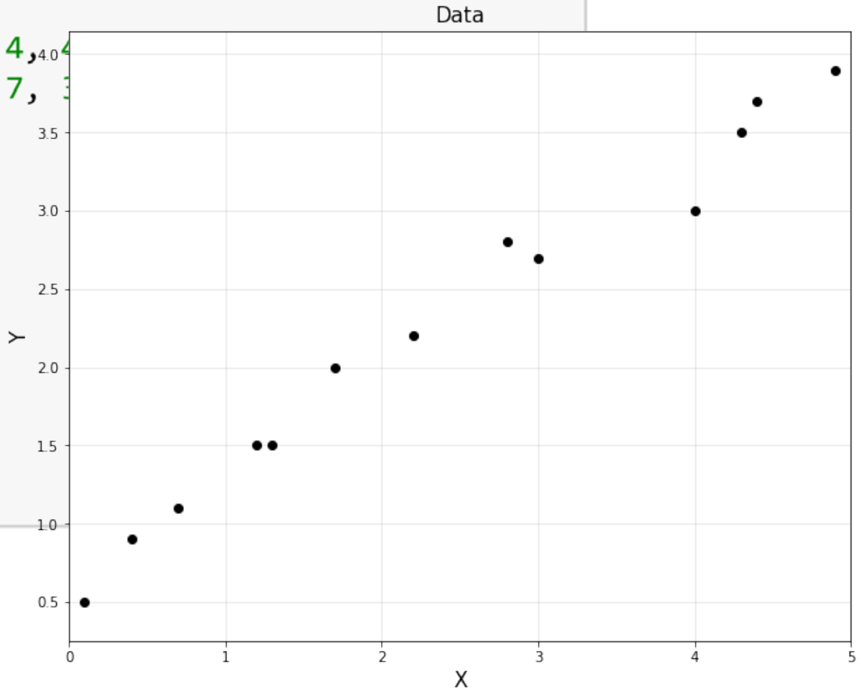
Solve using Linear Algebra

- known as *least square*

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

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

```
# 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,
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,

plt.figure(figsize=(10,8))
plt.plot(x,y,'ko')
plt.title('Data', fontsize=15)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.axis('equal')
plt.grid(alpha=0.3)
plt.xlim([0, 5])
plt.show()
```

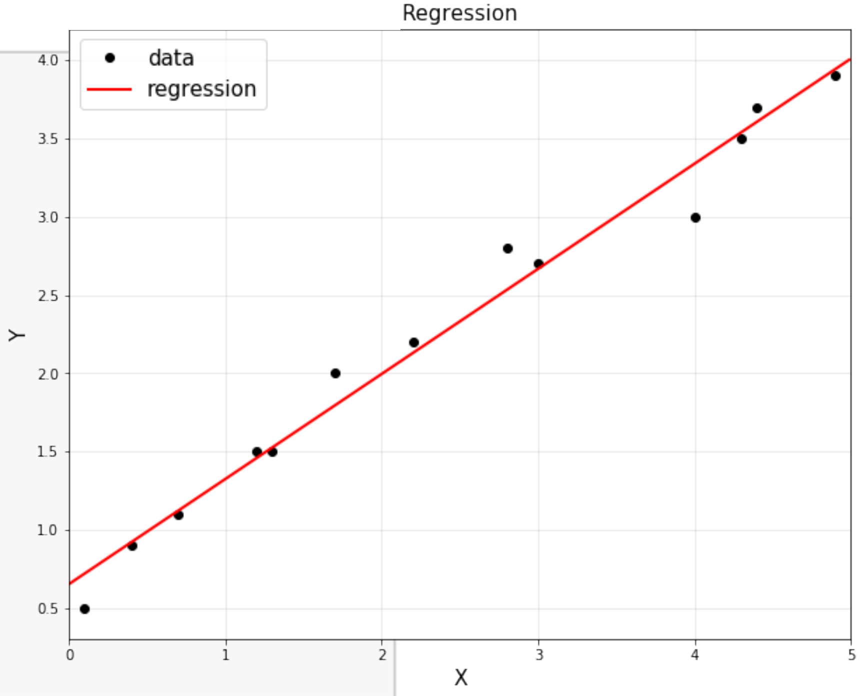


Solve using Linear Algebra

```python
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]]
```

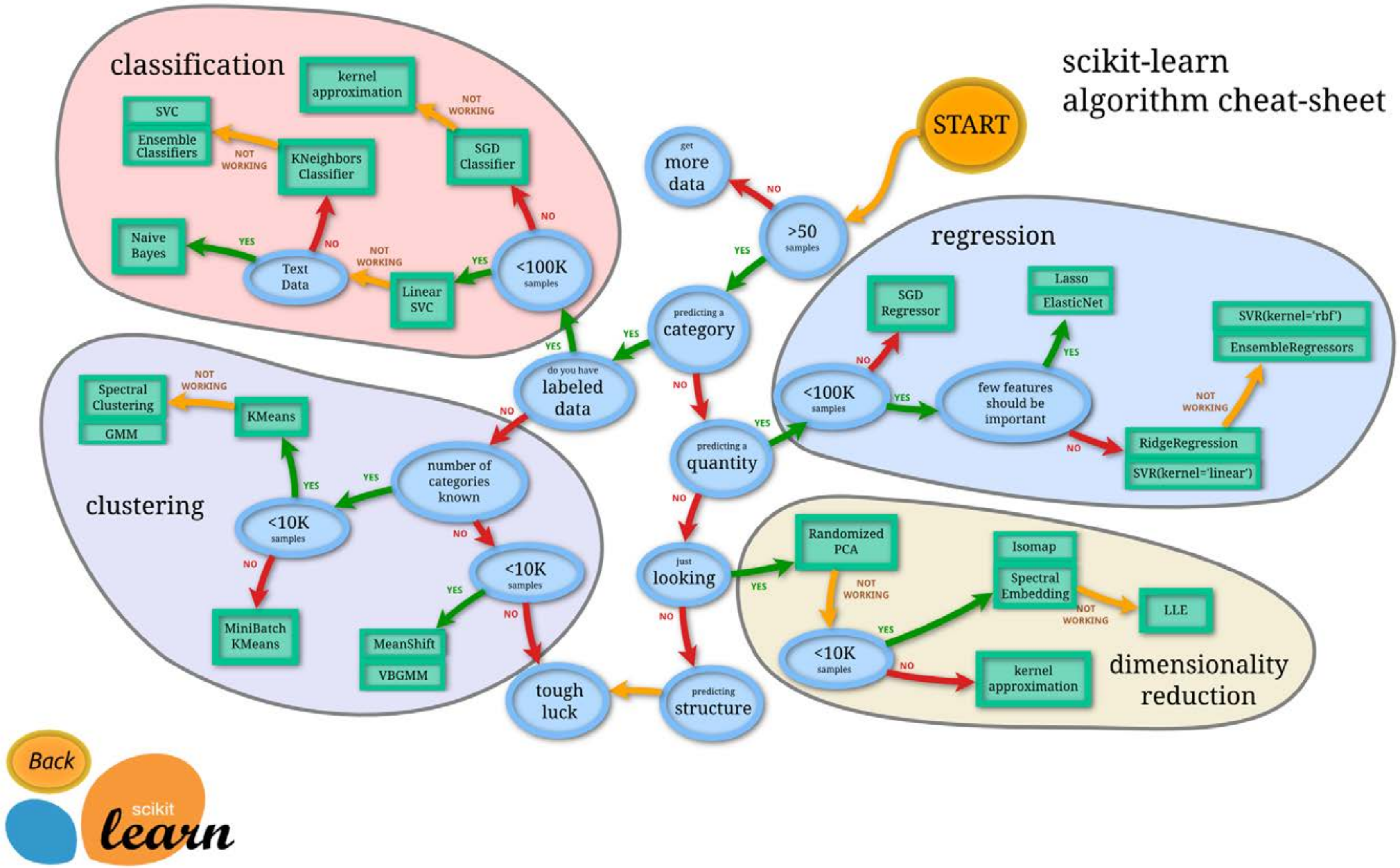
```python
# to plot
plt.figure(figsize=(10, 8))
plt.title('Regression', fontsize=15)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.plot(x, y, 'ko', label="data")

# 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

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



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_
```

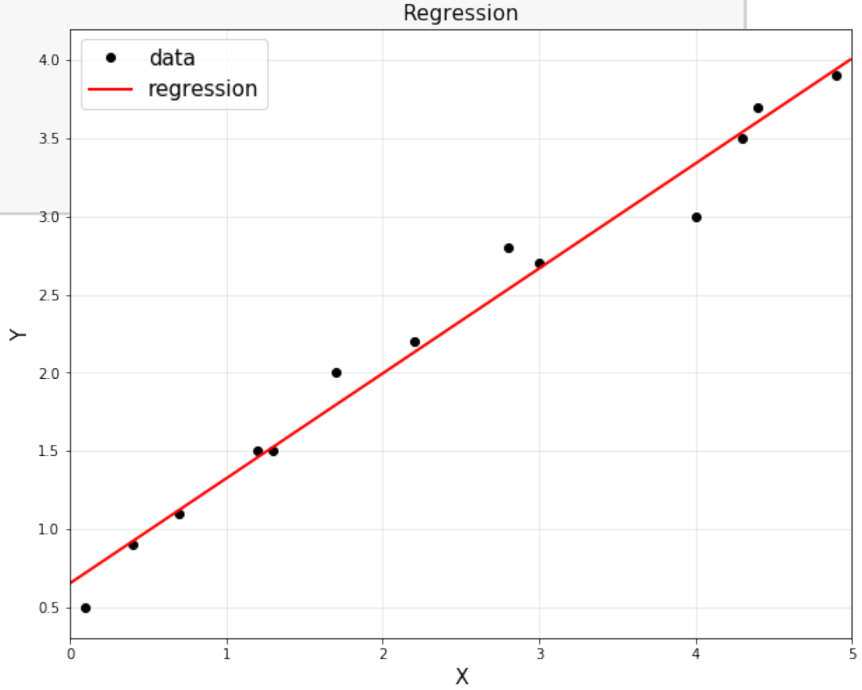
```
array([[0.67129519]])
```

```
reg.intercept_
```

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

Scikit-Learn: Regression

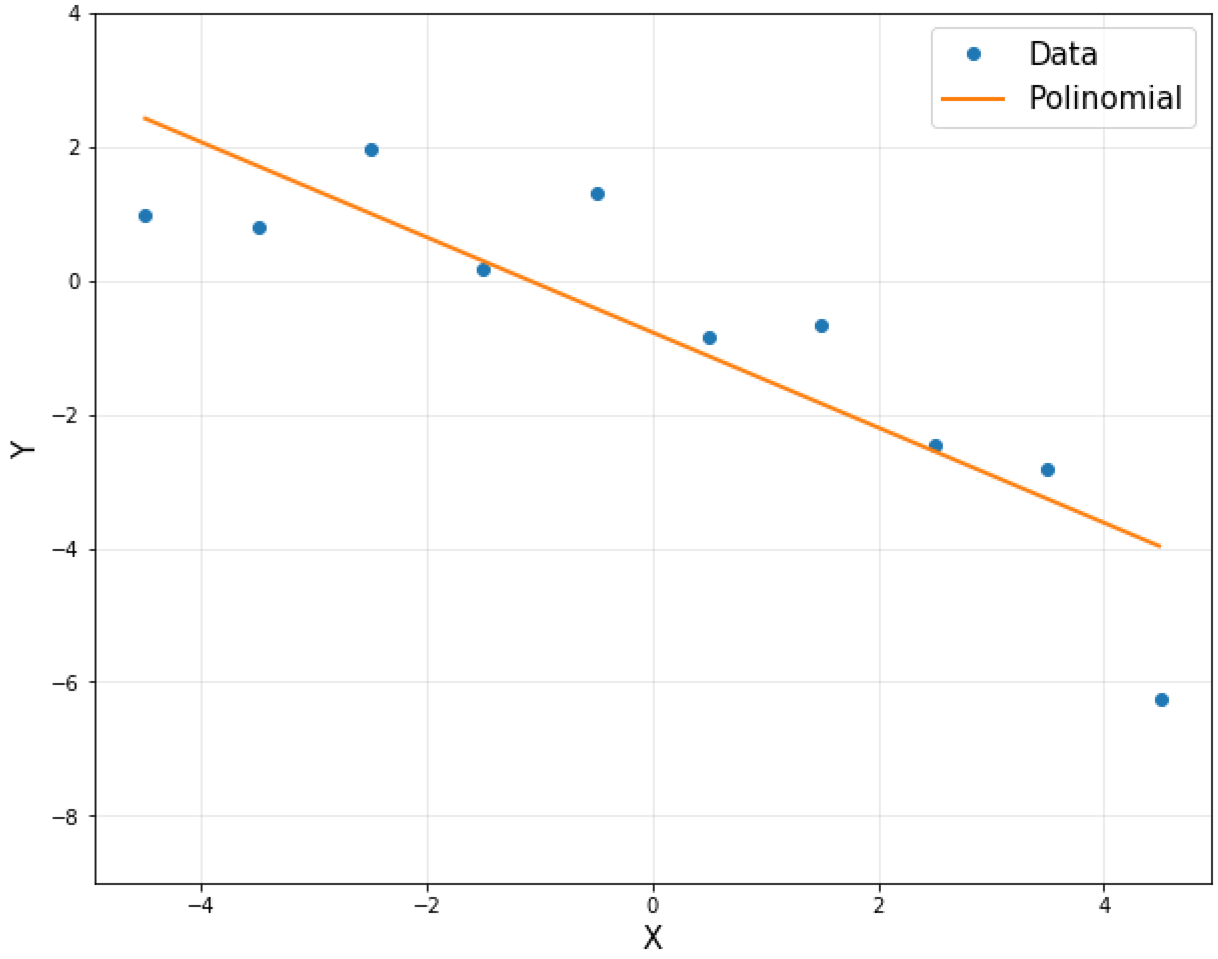
```
# to plot
plt.figure(figsize=(10, 8))
plt.title('Regression', fontsize=15)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.plot(x, y, 'ko', label="data")

# 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()
```



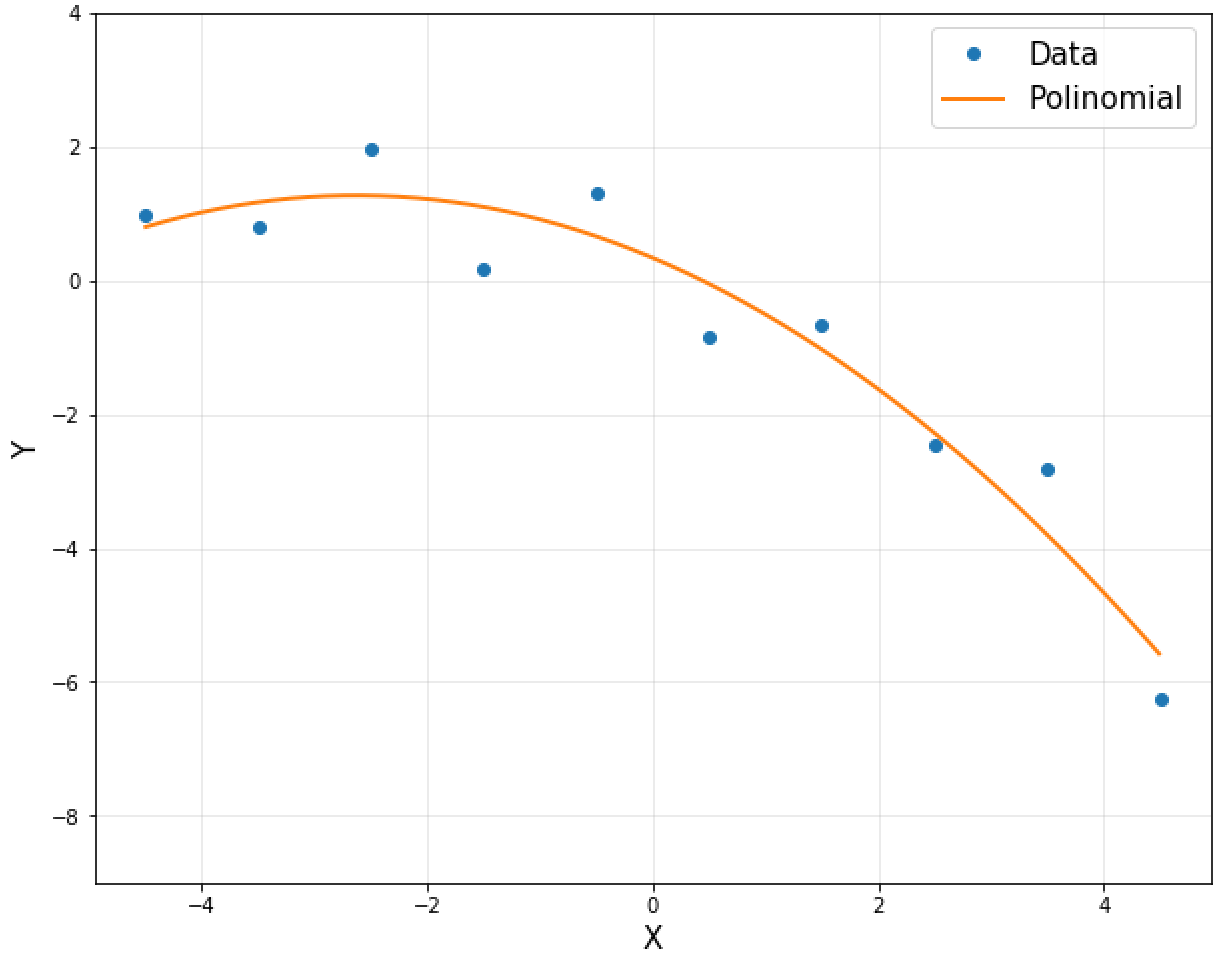
Overfitting

Industrial AI Lab.
Prof. Seungchul Lee

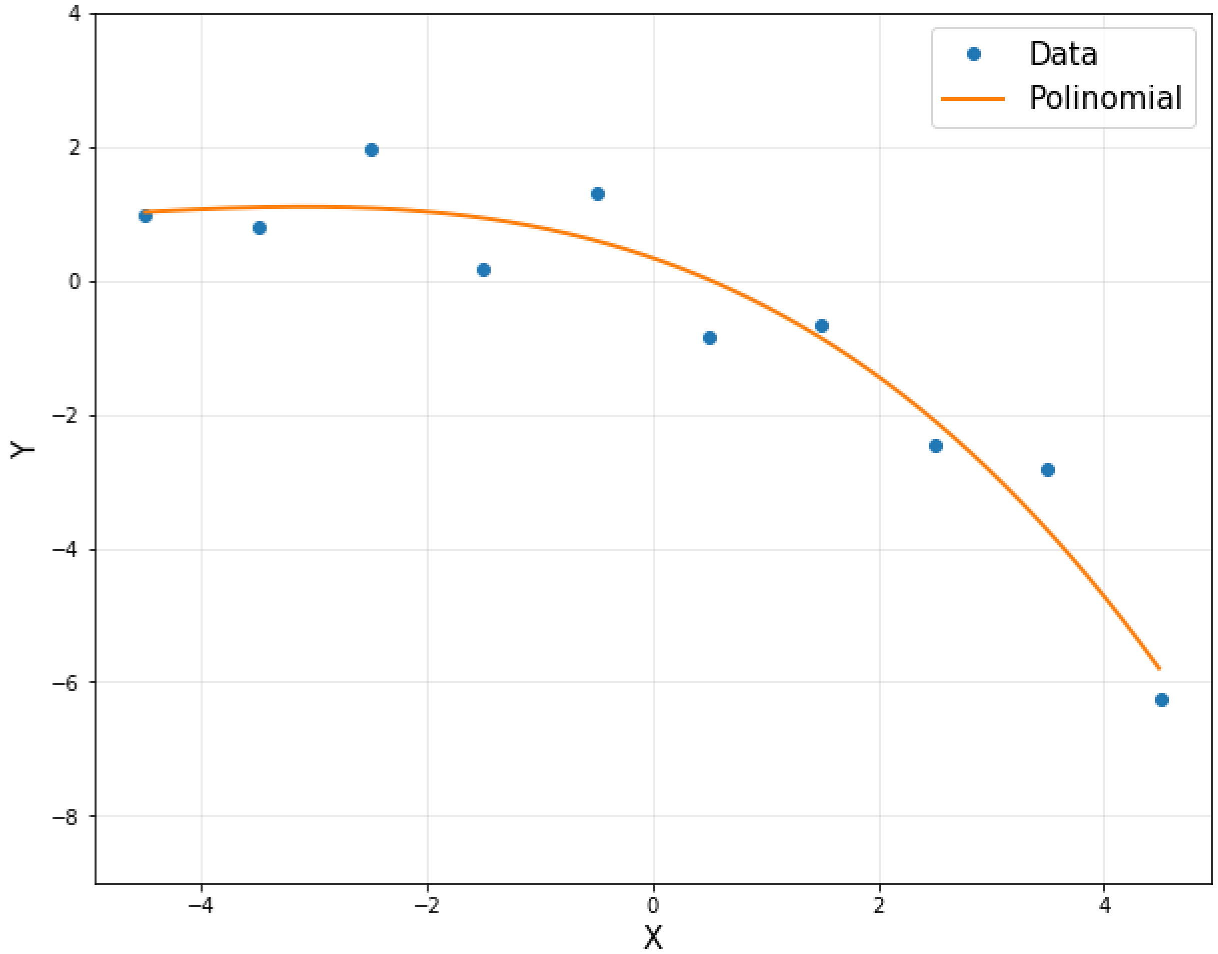
Polynomial Regression (d = 1)



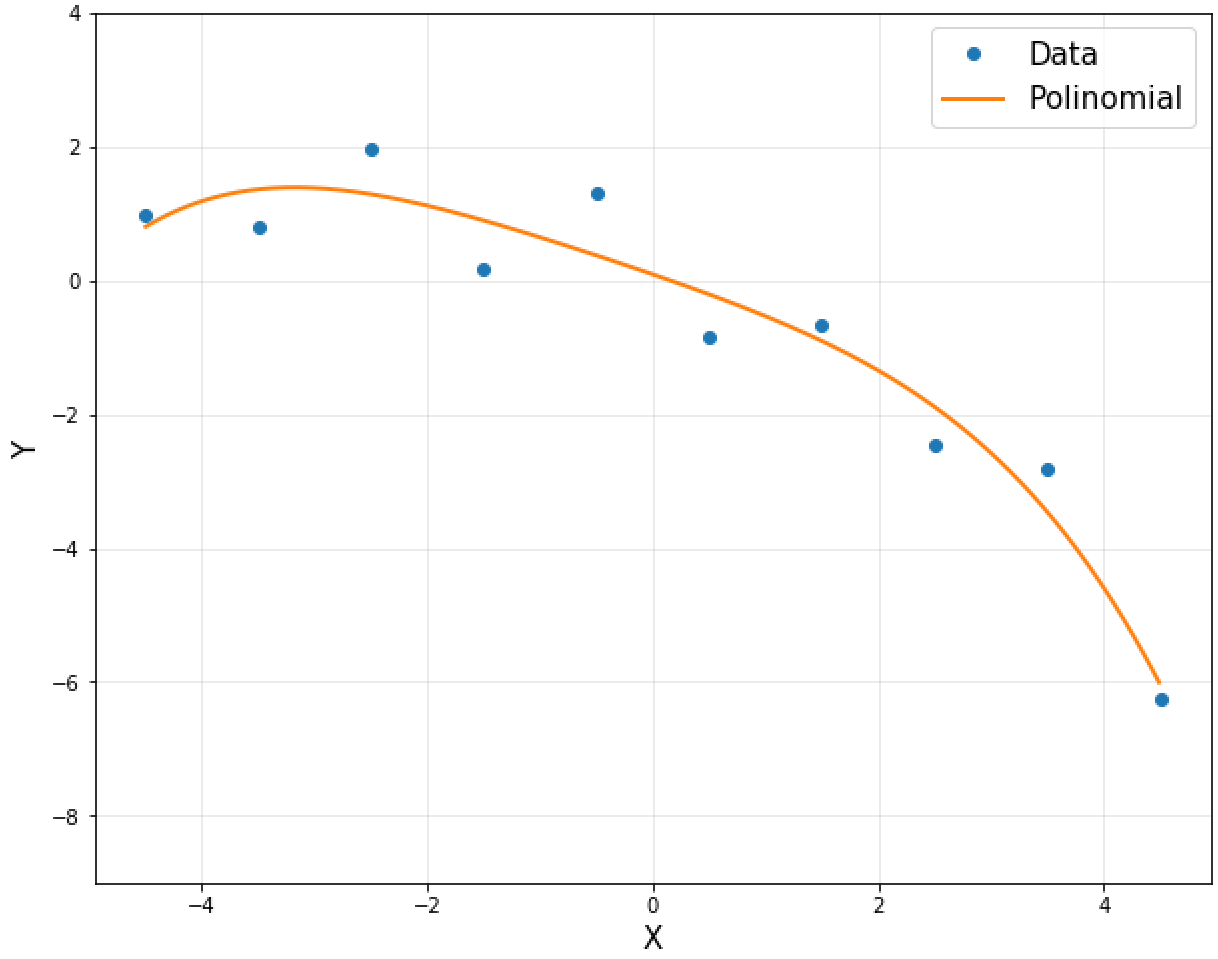
Polynomial Regression (d = 2)



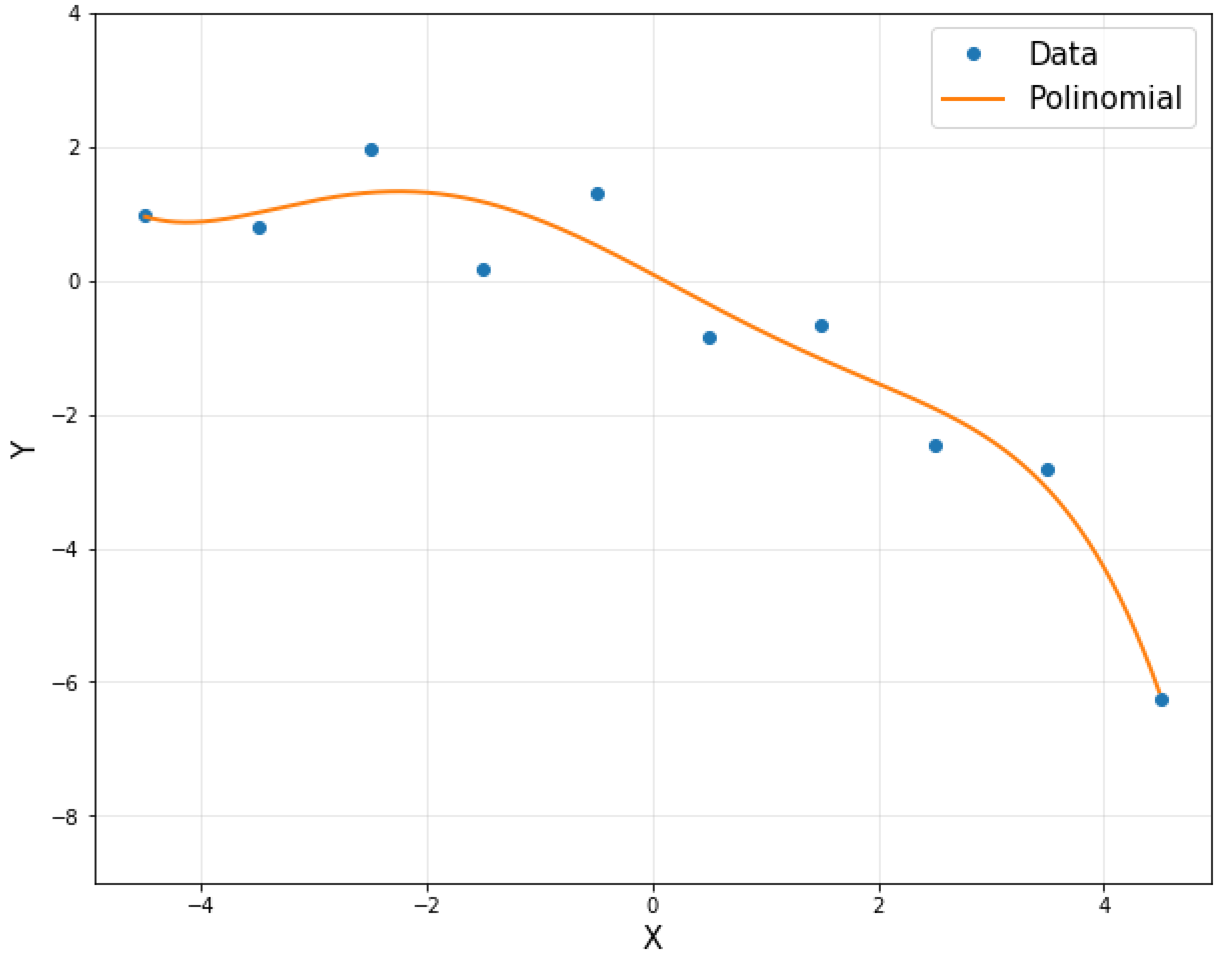
Polynomial Regression (d = 3)



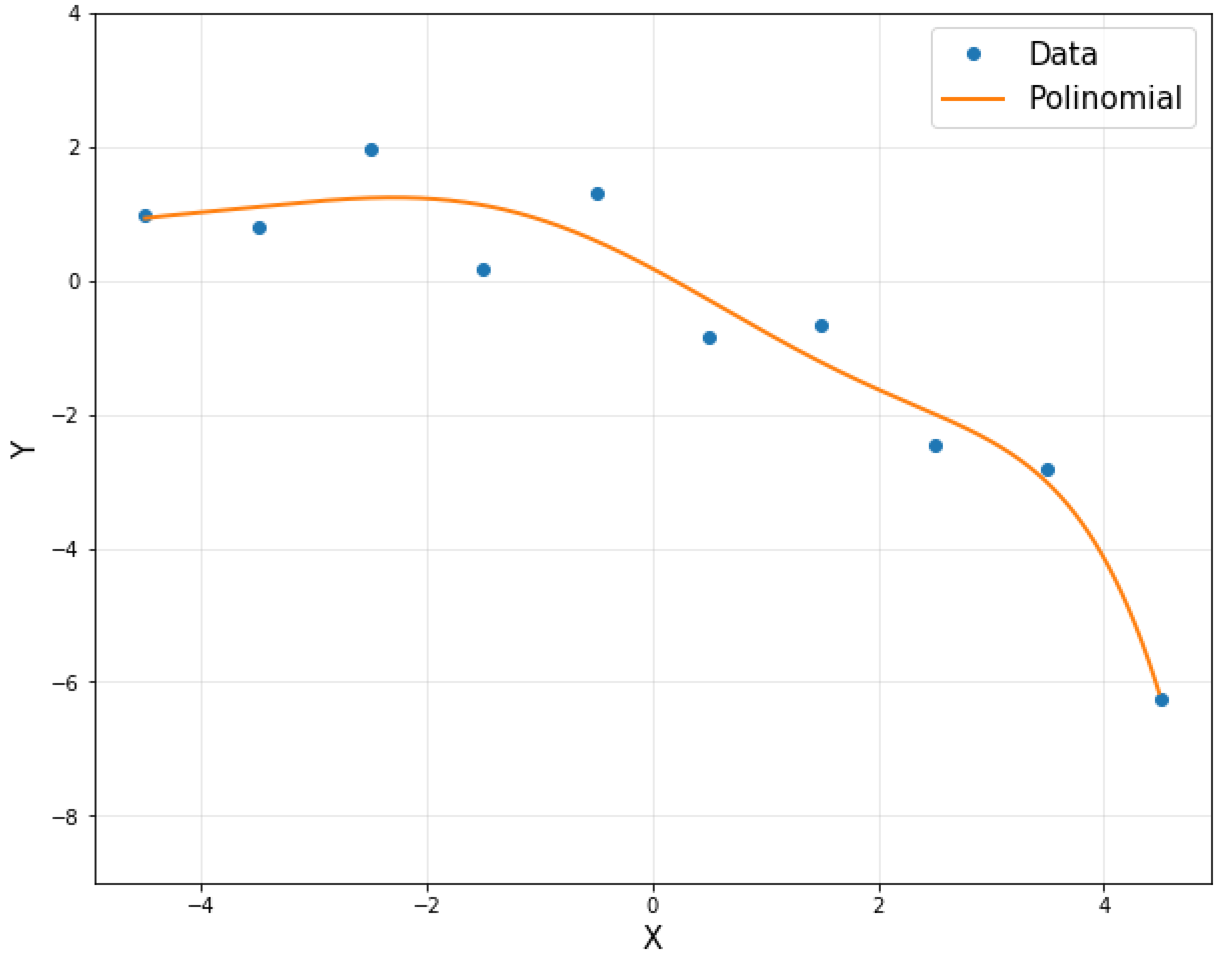
Polynomial Regression (d = 4)



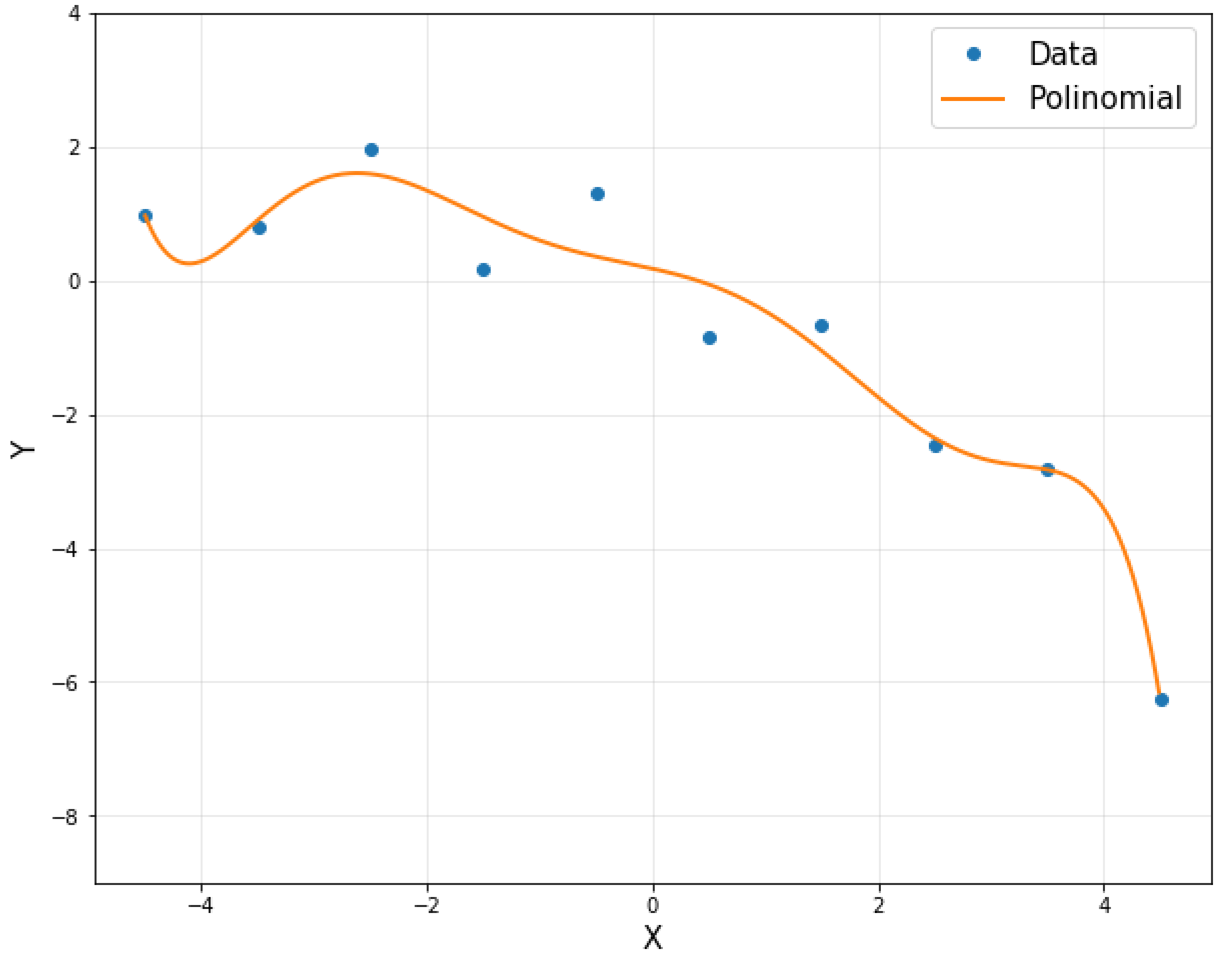
Polynomial Regression (d = 5)



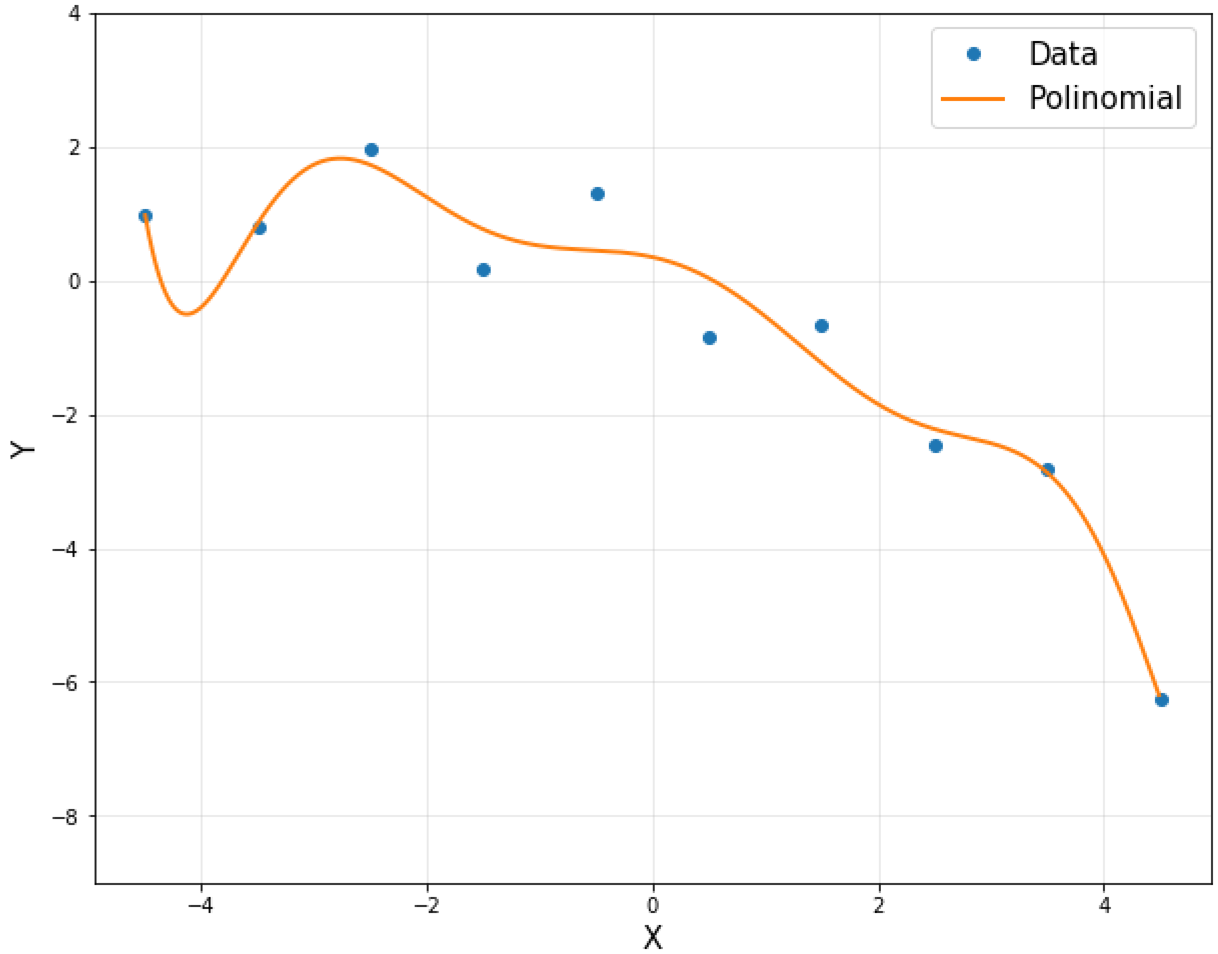
Polynomial Regression (d = 6)



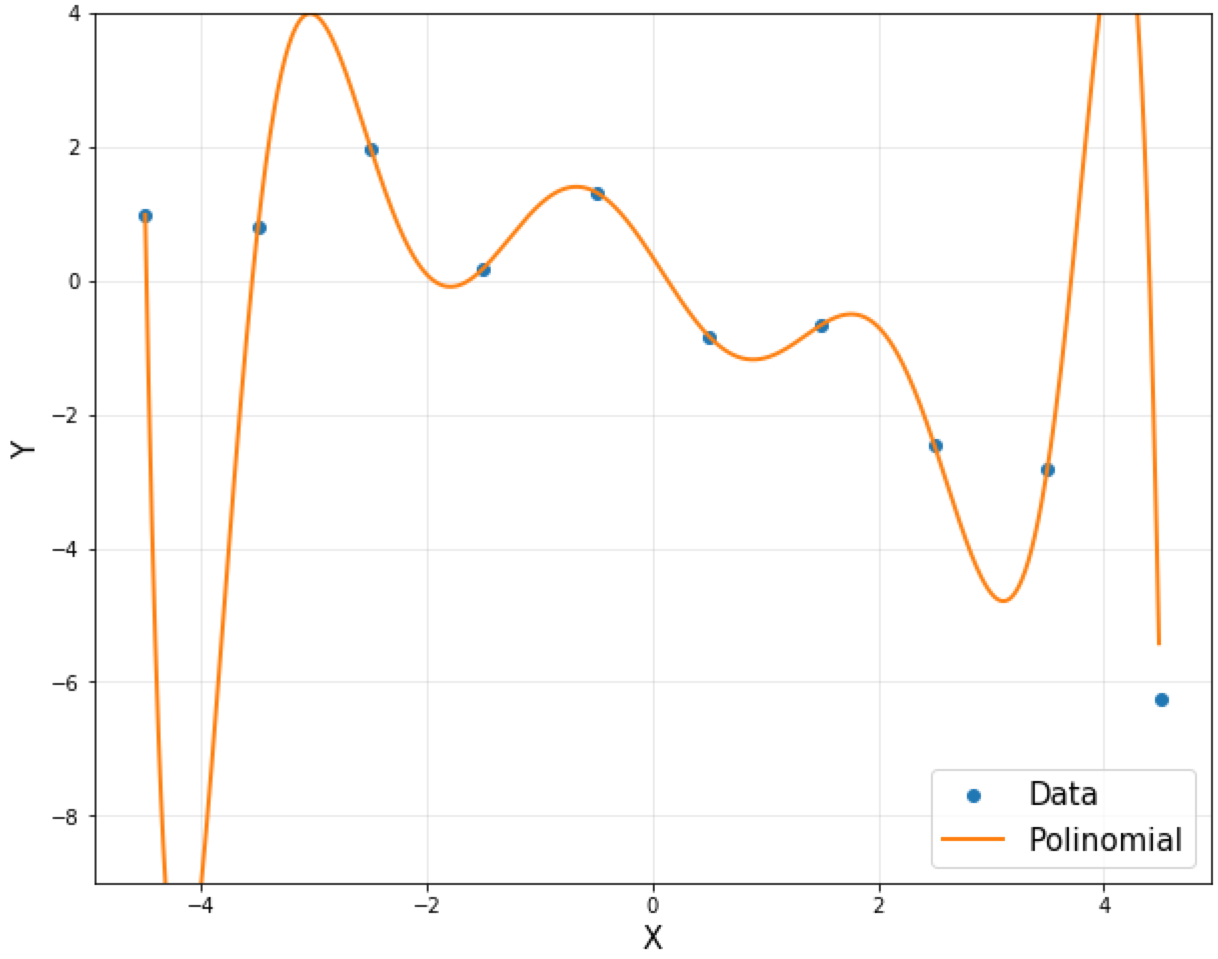
Polynomial Regression (d = 7)



Polynomial Regression (d = 8)



Polynomial Regression (d = 9)



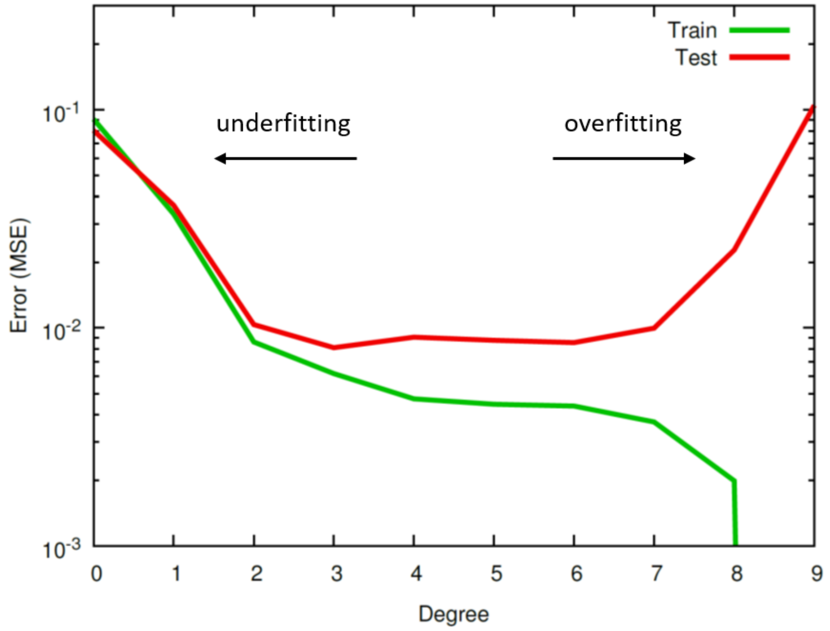
Overfitting Problem

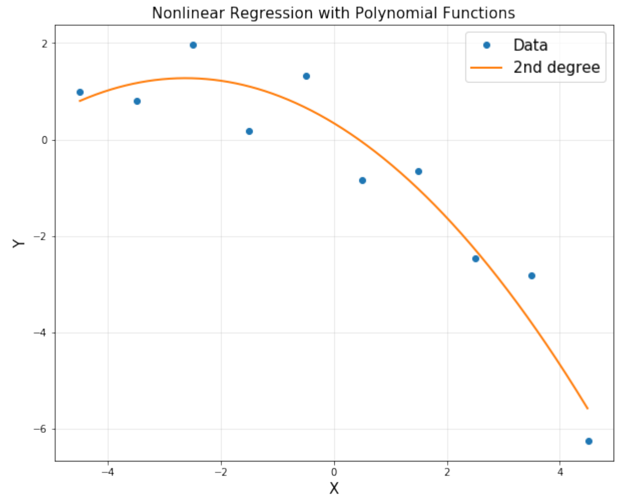
- Have you come across a situation where your model performed exceptionally well on train data, but was not able to predict test data ?

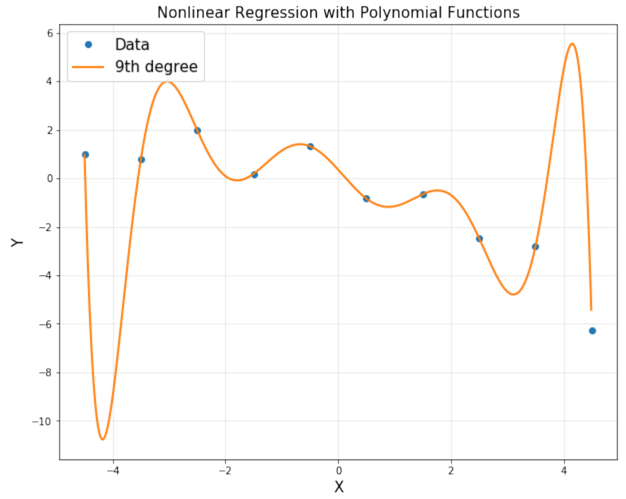
- One of the most common problem data science professionals face is to avoid overfitting.

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







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