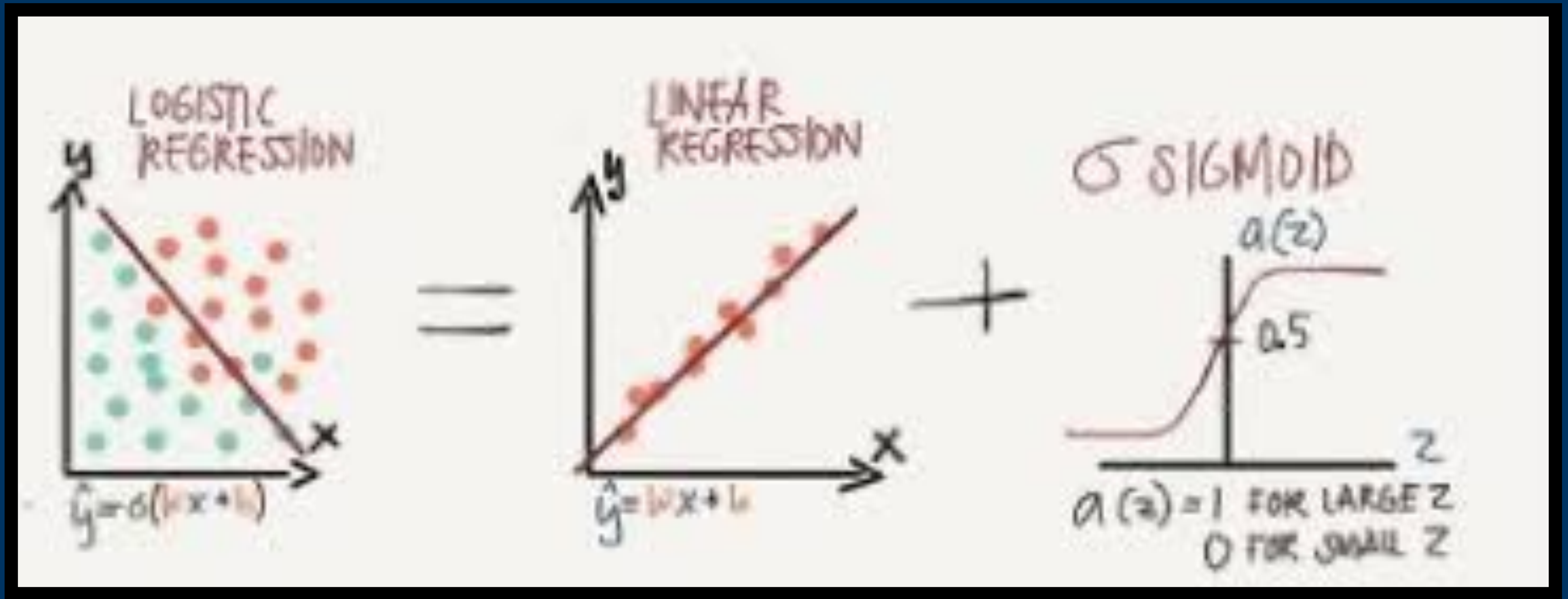
Nonlinear Regression

CSCI-P556 Applied Machine Learning
Lecture 12

D.S. Williamson



Agenda and Learning Outcomes

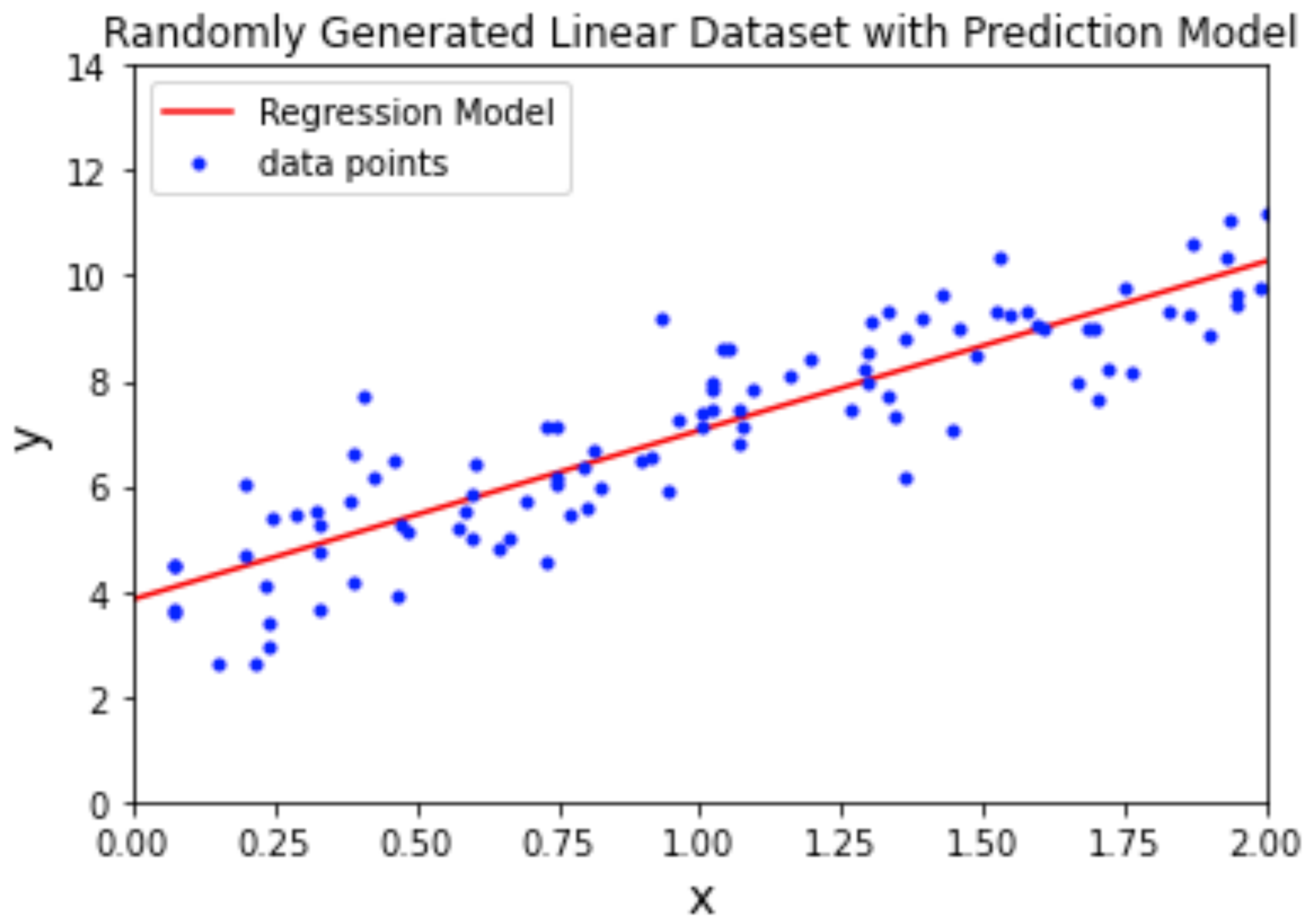
Today’s Topic(s)

- **Topic(s)**:
    - Nonlinear Regression
- **Announcements**
    - Project information has been posted to Canvas (will go over at the end of class). Upcoming deadlines on the following two Mondays
    - Homework #2 will be posted tonight or tomorrow.
        - Paired assignment
        - Due by Saturday, March 20th

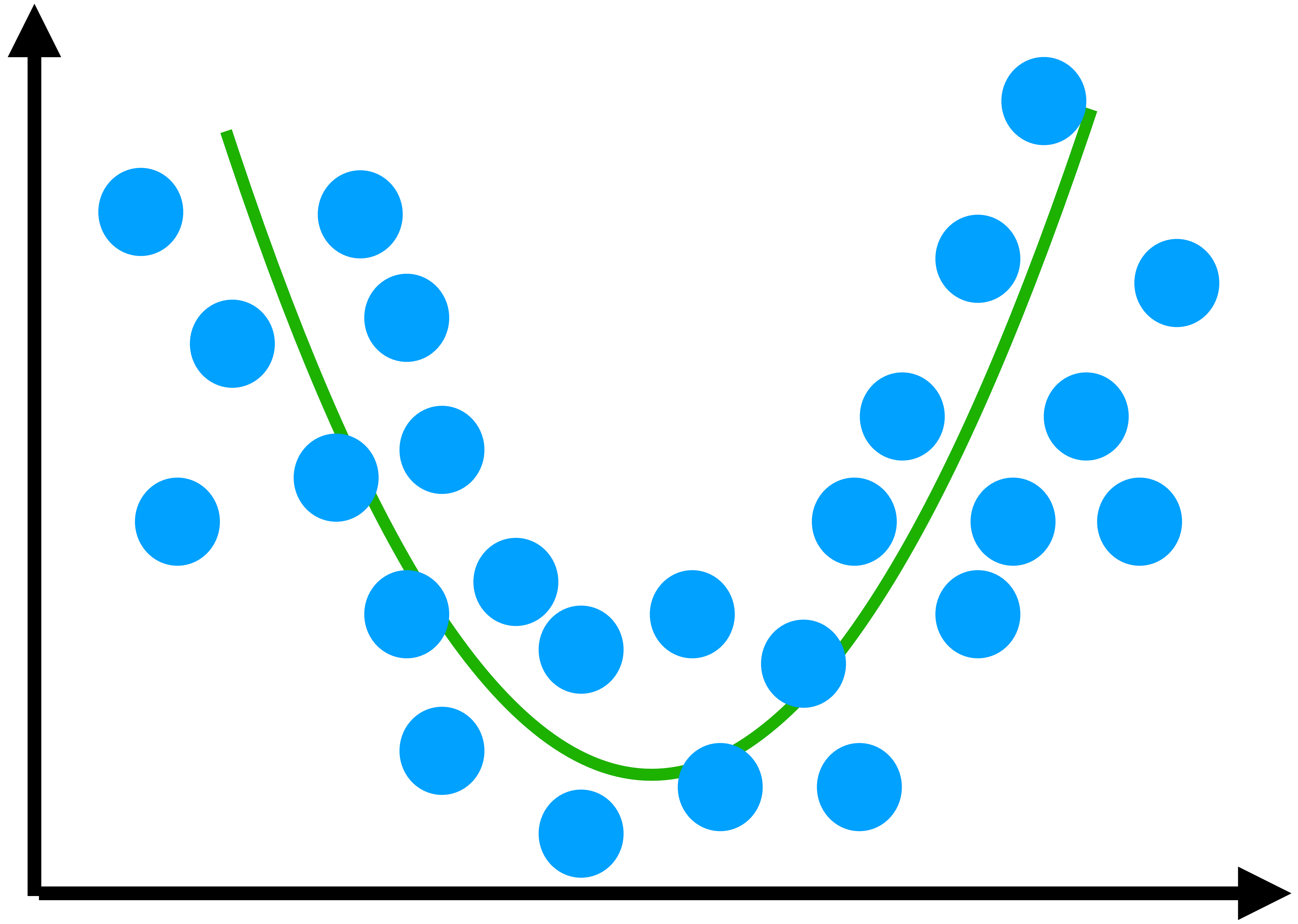
Non-Linear Regression

- Linear regression assumes a linear relationship between input *x* and label *y*
- How is regression performed if this relationship is not linear?

Linear Relationship



Nonlinear Relationship



Polynomial Regression Model

A nonlinear regression approach

- Fortunately, the techniques used for linear regression are useful for nonlinear regression (e.g. polynomial, logistic, etc.)

- ***Polynomial regression*** (aka polynomial curve fitting) is a special case of general linear regression
  - The regression model contains squared and higher-order terms of the predictor variable X
  - **Example**: Polynomial regression model with one predictor variable

$$y = W_0 + W_1 x + W_2 x^2 + \cdots + W_p x^p + \epsilon$$

Polynomial Regression Models

When to try Polynomial Regression

- ***Polynomial regression*** models have ***two basic types*** of uses:

  1. When the true response function is indeed a polynomial function
  2. When the true response function is unknown (or complex) but a polynomial function may be a good approximation to the true function

The second type of use is very common

Danger:

- Extrapolations beyond the range of the data may be hazardous
  - It may turn in unexpected directions when extrapolated beyond the range of the data
- This is important for prediction, because new (testing) data may occur outside of the training range

Polynomial Regression

- Suppose we do not know the true response function, but we want to fit a polynomial curve to it. Let's *define this response function, E[Y|X], as f(x)*
    - In the linear case (single predictor): $f(x) = W_0 + W_1 x$
- Using a ***polynomial curve with degree p***, the *response function* is

$$f(x) = W_0 + W_1 x + W_2 x^2 + \cdots + W_p x^p = \sum_{j=0}^{p} W_j x^j$$

- This can be re-written as a linear regression problem involving basis functions $\phi_j(x)$

$$\phi_j(x) = x^j$$

Linear function of ϕ

$$f(x) = W_0 + W_1 \phi_1(x) + W_2 \phi_2(x) + \cdots + W_p \phi_p(x) = \sum_{j=0}^{p} W_j \phi_j(x)$$

Polynomial Regression (cont.)

Finding the regression coefficients

- Now the response function *E[Y | X]* is modeled as

$$f(x) = \sum_{j=0}^{p} W_j \phi_j(x) = \mathbf{W}^T \phi \qquad \text{where } \boldsymbol{w} = \begin{bmatrix} W_0 \\ W_1 \\ \vdots \\ W_p \end{bmatrix}, \ \phi = \begin{bmatrix} \phi_0(x) \\ \phi_1(x) \\ \vdots \\ \phi_p(x) \end{bmatrix}$$

- Hence, our polynomial model in terms of x, now looks like a linear regression model in terms of $\phi_j(x)$

- We can then use Linear Regression, where our $\mathbf{\Phi}$ feature vector replaces $\mathbf{X}$, to find the ***closed-form solution***.

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

$$\mathbf{\Phi} = \begin{bmatrix} \phi_0(x_1) & \cdots & \phi_p(x_1) \\ \phi_0(x_2) & \cdots & \phi_p(x_2) \\ \vdots & & \\ \phi_0(x_n) & \cdots & \phi_p(x_n) \end{bmatrix}$$

Polynomial Regression: The Normal Equation approach

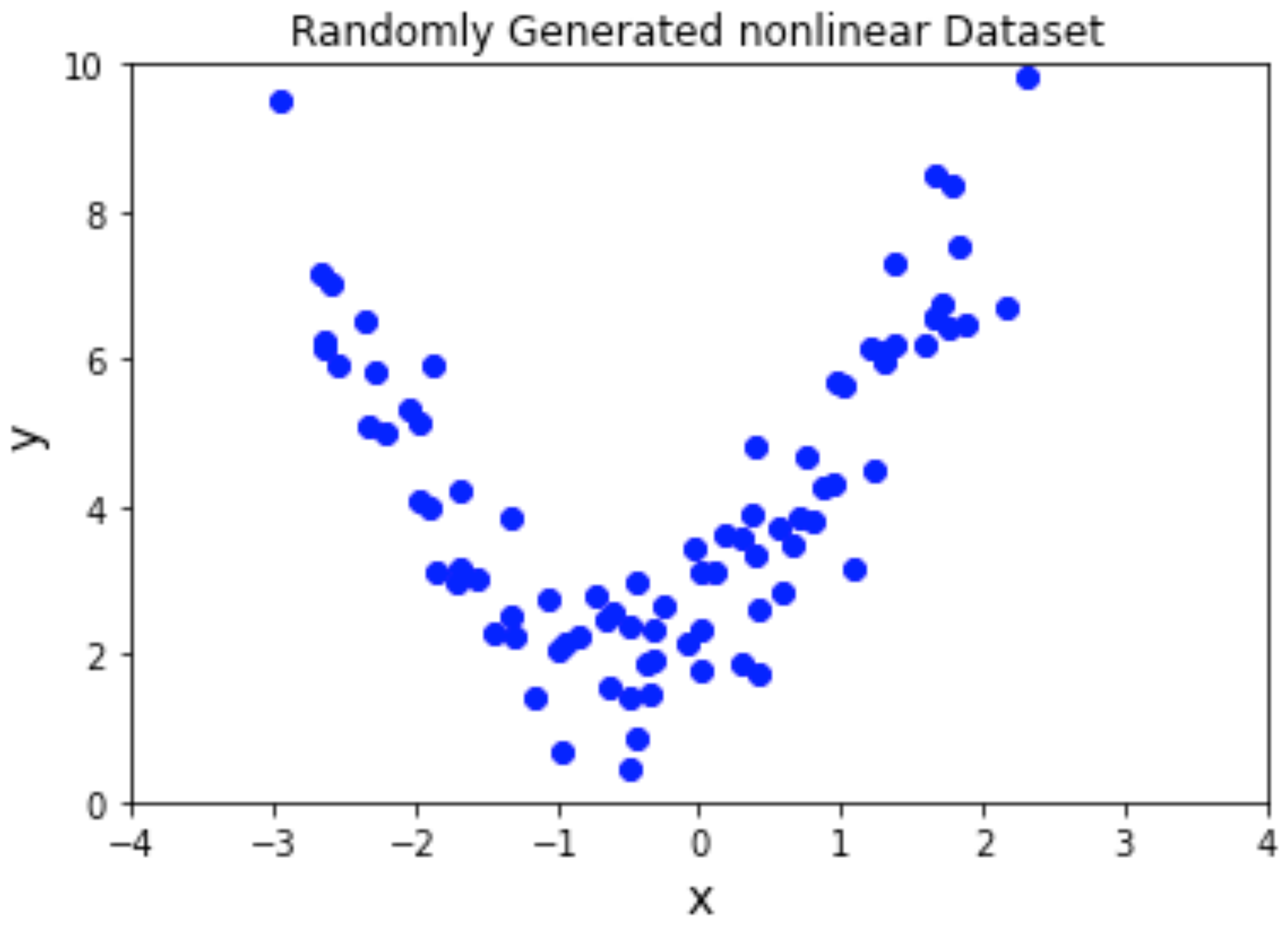
A Python Example

- Randomly generate x values between -3 and 3.
- Model polynomial equation, $y = x^2 + x + 2.5$, with random Gaussian noise added

```python
import numpy as np
import matplotlib.pyplot as plt

N = 100
x = 6*np.random.rand(N,1) - 3
y = x**2 + x + 2.5 + np.random.randn(N,1)

plt.plot(x,y,'bo')
plt.ylabel('y', fontsize=14)
plt.xlabel('x', fontsize=14)
plt.xlim(-4,4)
plt.ylim(0,10)
plt.title("Randomly Generated nonlinear Dataset")
```



Polynomial Regression: The Normal Equation approach

A Python Example

- Note that p=2 in this problem, so we will assume this is known when trying to find the regression coefficients

- Scikit-Learn's *PolynomialFeatures* class can be used to combine the linear and square terms of each feature

```
from sklearn.preprocessing import PolynomialFeatures
poly_features = PolynomialFeatures(degree=2, include_bias=False)
x_poly = poly_features.fit_transform(x)
x[0]
x_poly[0]
```

```
array([0.41711634, 0.17398604])
```

Returns x and x^2

Polynomial Regression: The Normal Equation approach

A Python Example

- Now we can fit a Linear Regression model to this transformed data (e.g. Polynomial Regression)

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(x_poly,y)
lin_reg.intercept_, lin_reg.coef_
```

```
(array([2.42373112]), array([[1.05095741, 1.03516327]]))
```

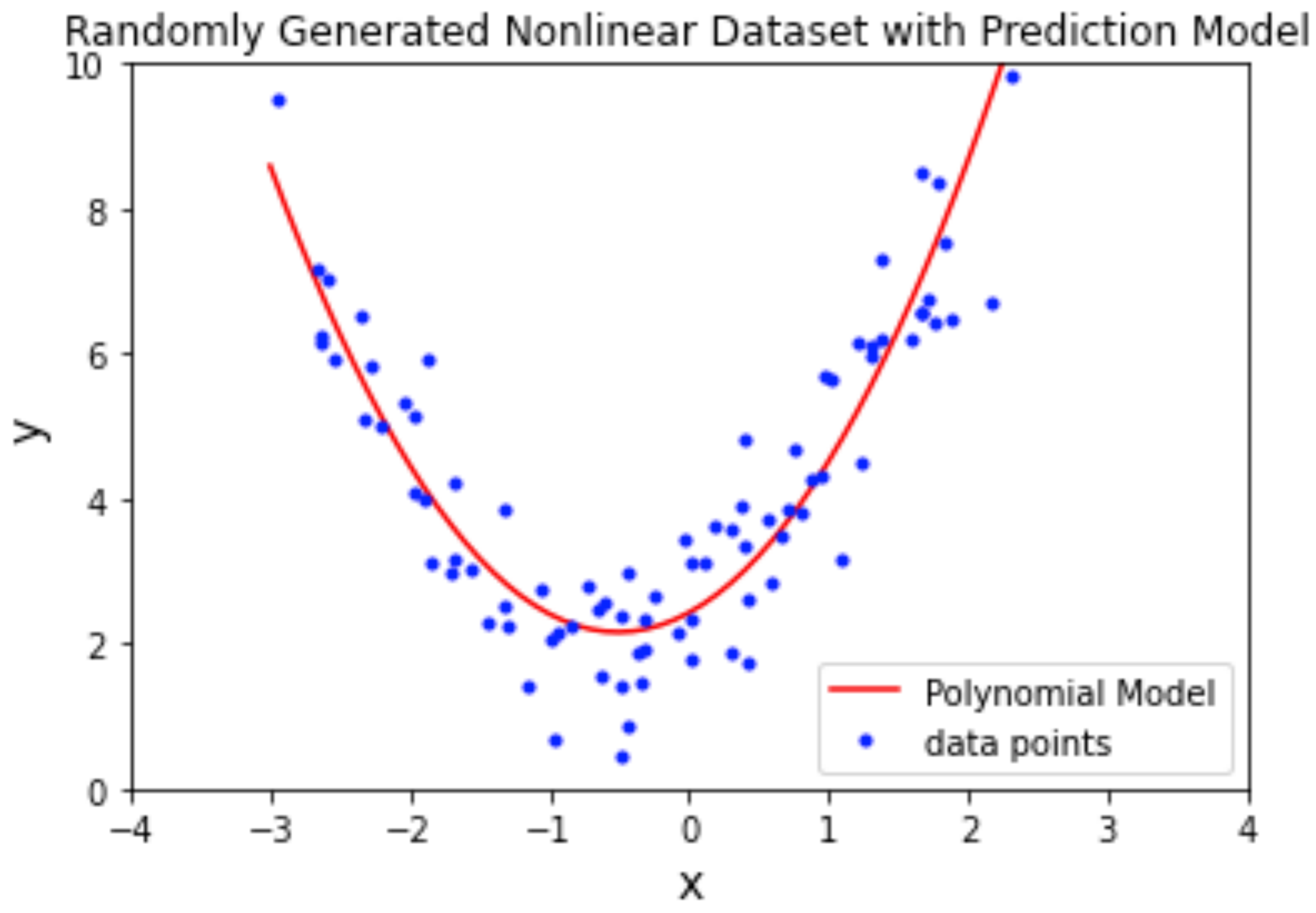
Polynomial Regression: The Normal Equation approach

A Python Example

- Now we can fit a Linear Regression model to this transformed data (e.g. Polynomial Regression)

```
x_new = np.arange(-3, 3,0.1)
x_new_mat = np.c_[x_new**0, x_new, x_new**2]
theta_hat = [lin_reg.intercept_[0],lin_reg.coef_[0][0],lin_reg.coef_[0][1]]
print(theta_hat)
y_hat = x_new_mat.dot(theta_hat)
```

```
[2.423731117658376, 1.05095740979356, 1.0351632739272136]
```



- Model: $\hat{y} = 1.035x^2 + 1.051x + 2.424$
- True: $y = x^2 + x + 2.5$

$$R^2 = 0.91$$

Evaluating Regression Performance

Coefficient of Determination

Sum of square errors

$$\text{SSE} = \sum_i (y_i - \hat{y}_i)^2$$

- In addition to MSE and MAE, regression performance can be assessed using the ***coefficient of determination***. It measures the effect of *x* in reducing uncertainty in *y*.

$$R^2 = 1 - \frac{SSE}{SSTO}$$

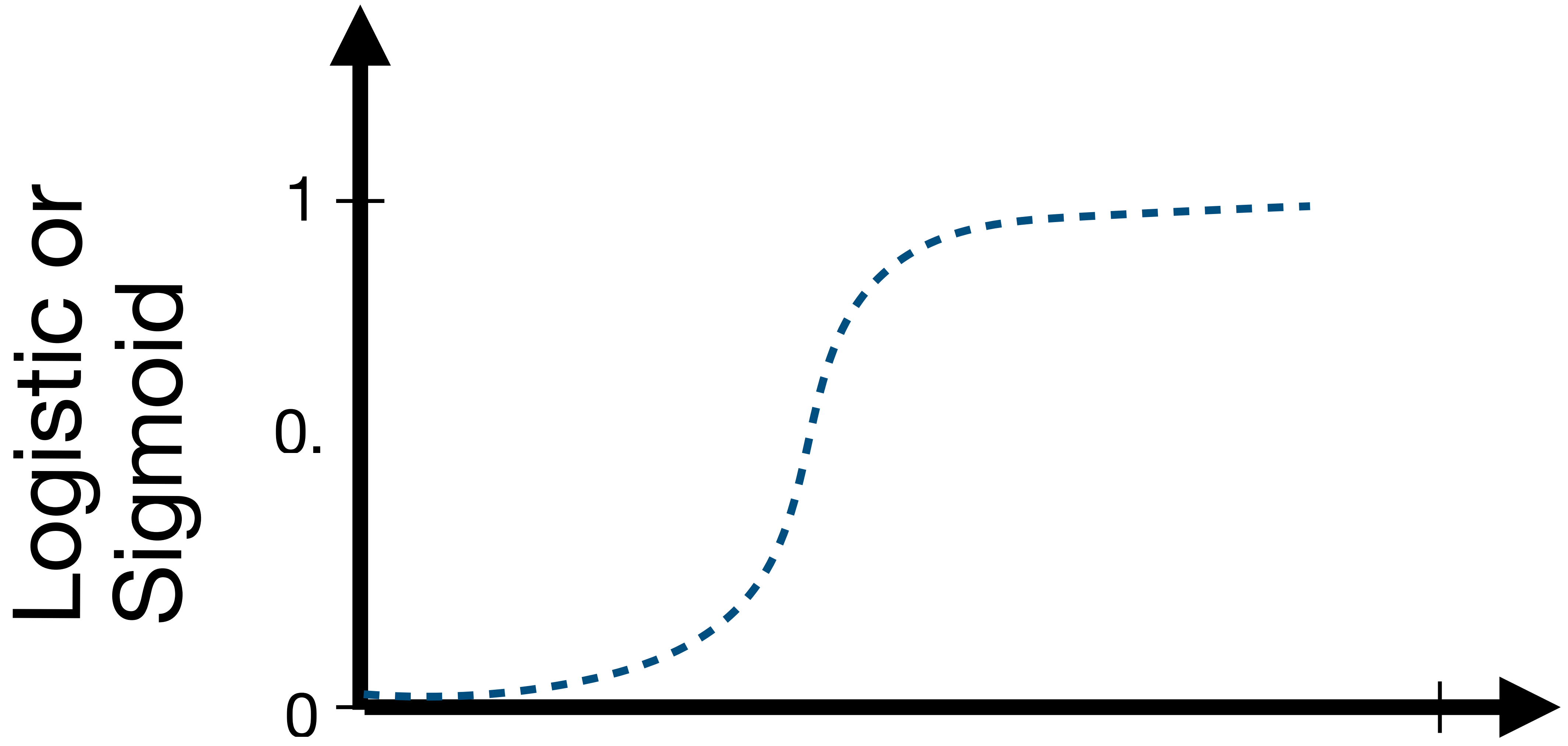
$$\text{SSTO} = \sum_i (y_i - \mu_y)^2$$

Total Sum of Squares: Proportional to the variance of y

- R^2 can be positive or negative, where the best possible value is 1.
  - A baseline model that predicts μ_y for each sample will have $R^2 = 0$
  - Models that do worse than the baseline will have negative R^2 values
  - Hence, values closer to 1 tend to be better (e.g. for linear models)
- Use the *score()* function of *LinearRegression* to compute this value in Python

General Nonlinear Regression

- Polynomial regression is only one form of nonlinear regression $\phi_j(X) = X^j$
- Other forms of nonlinear regression are possible by changing the basis function $\phi(x)$



  - Radial basis functions (RBFs) → $\phi_j(x) = e^{-\frac{(x-\mu_j)^2}{2\sigma_j^2}}$
  - Sigmoid basis function → $\phi_j(x) = \dfrac{1}{1 + e^{-\frac{x-\mu_j}{s_j}}}$
  - Logistic regression → $\phi(x) = \dfrac{1}{1 + e^{-x}}$
  - Softmax Regression → $\phi_j(x) = \dfrac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}$

We'll discuss more about this when we cover neural networks

Nonlinear Regression Models

- Nonlinear regression models follow the same general form as linear regression models

$$y = f(\mathbf{x}) + \epsilon$$

y is the label value
$f(\mathbf{x})$ is a nonlinear function
ϵ is a random noise term (often 0 mean, with variance σ^2)

- We can use Linear Regression techniques to find the optimal regression coefficients
  - The Normal Equation
  - Gradient Descent (Batch, Stochastic, Mini-Batch)
- We just need to re-write $f(\mathbf{x})$ in terms of $f(\phi(\mathbf{x}))$

In Class Practice: Group Work

- You have a small dataset that consists of four (x,y) points: (1,1.2), (2,2.3), (3,2.3), and (4, 3.3). The true relationship is: $y = 1 + 0.5x + \epsilon$
- In Python, use the ***closed-form solution*** to estimate the regression coefficients (e.g. ***do not use built-in Python regression libraries***), separately using p = 1, 2 and 3 (e.g. polynomial regression).
  - What is ϵ for each data point?
  - What are the coefficients for each value of p?
  - What is the mean-square error between the estimated values for y and the true values for y, for each value of p?

In-Class Regression Practice

$$f(X) = \sum_{j=0}^{p} W_j \phi_j(X) = \boldsymbol{W}^T \boldsymbol{\phi}$$

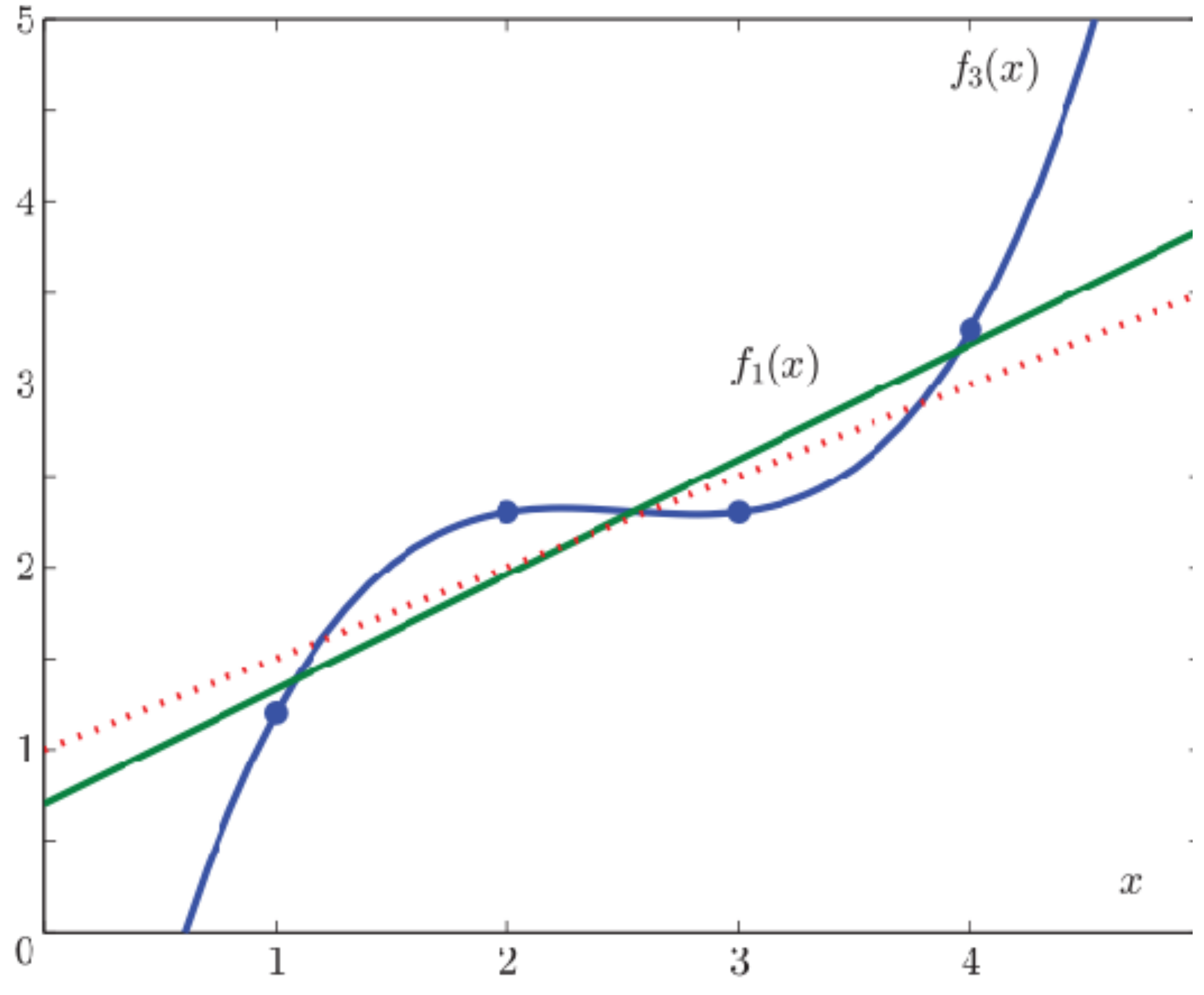
- Polynomial fitting with p=1, p=2 and p=3

$$y_i = 1 + \frac{x_i}{2} + \epsilon_i \quad , \quad \epsilon_i = \begin{bmatrix} -0.3 \\ 0.3 \\ -0.2 \\ 0.3 \end{bmatrix}$$

<u>p=1</u> $\hat{\theta} = \begin{bmatrix} 0.7 \\ 0.63 \end{bmatrix}$

<u>p=2</u> $\hat{\theta} = \begin{bmatrix} 0.575 \\ 0.755 \\ -0.025 \end{bmatrix}$

<u>p=3</u> $\hat{\theta} = \begin{bmatrix} -3.1 \\ 6.6 \\ -2.65 \\ 0.35 \end{bmatrix}$



The "best" fit occurs with the cubic polynomial

The resulting mean of square errors

<u>p=1</u> MSE = 0.06

<u>p=2</u> MSE = 0.06

<u>p=3</u> MSE = 0

If given unseen (or outside) testing data, can show that the cubic polynomial performs the worst

Hence, ***Overfitting***!!!!

Project Details

Next Class:

Neural Networks