Machine Learning

| Polynomial Regression

Prof. Dr. Mostafa Elhosseini

https://youtube.com/@ProfElhosseiniTechVerse

Agenda

- What happens if our data doesn’t resemble a line?
- Polynomial Regression
- Add more columns to our dataset
- Function of Polynomial Regression in Analyzing Feature Relationships
- How do we decide on this degree?

Polynomial Regression

- What happens if our data doesn't resemble a line?
- Polynomials are a class of functions that are helpful when modeling nonlinear data.

- $y = 2x^3 + 8x^2 - 40$: Degree 3

- The graph of a **polynomial** looks a lot like a curve that **oscillates several times**.

Polynomial Regression

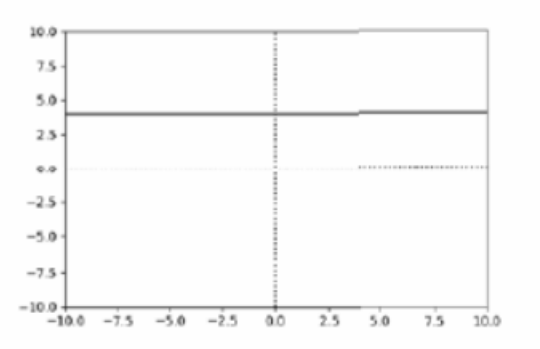
What happens if our data doesn't resemble a line?

- An extension of linear regression that uses **curved lines** instead of straight ones
- Allows **for fitting more complex**, non-linear patterns in data

- Uses **polynomial functions** (e.g., quadratic, cubic) to represent the relationship
- Can be more **flexible** but also more **prone to overfitting** than linear regression

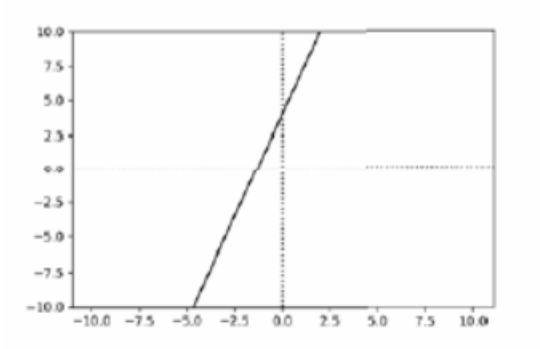
Polynomial Regression

- If a polynomial has degree d, then the graph of that polynomial is a curve that **oscillates at most d – 1 times** (for d > 1).



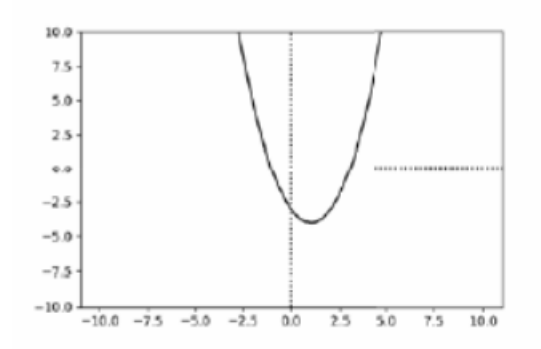
Degree 0

$y = 4$



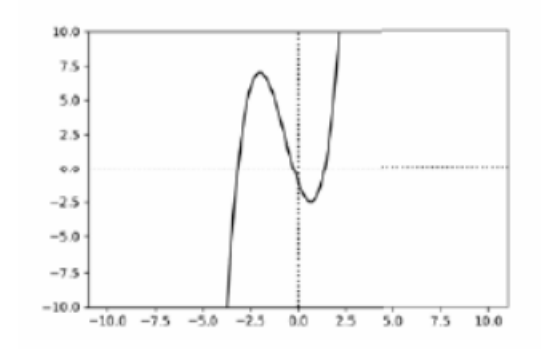
Degree 1

$y = 3x + 4$



Degree 2

$y = x^2 - 2x - 3$

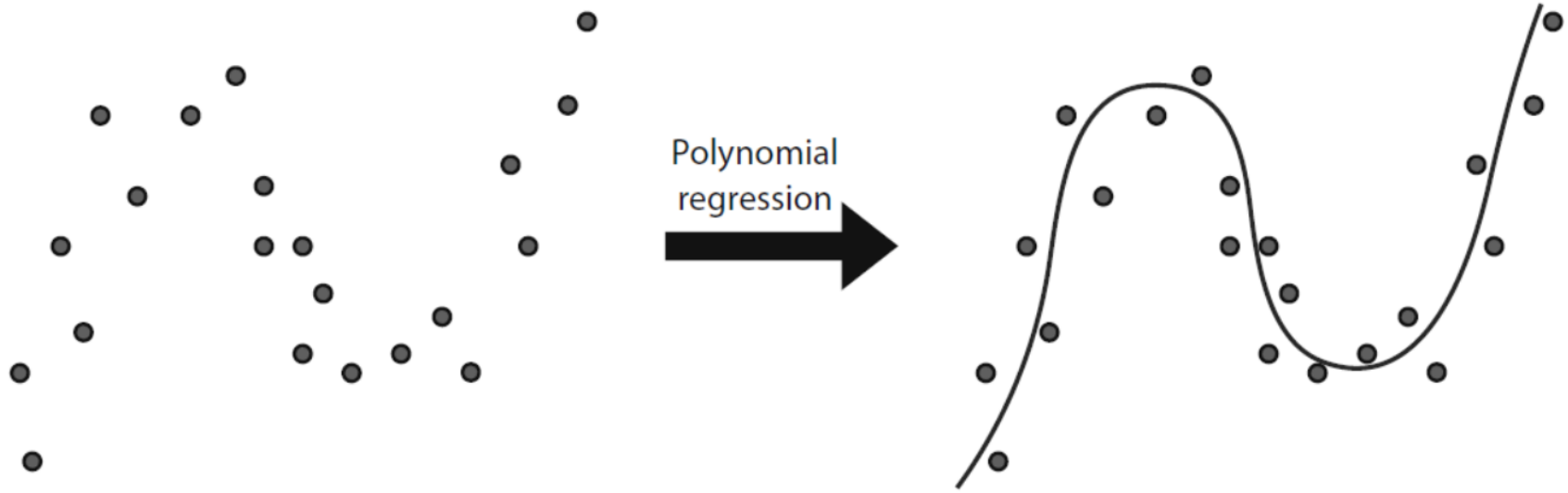


Degree 3

$y = x^3 + 2x^2 - 4x - 1$

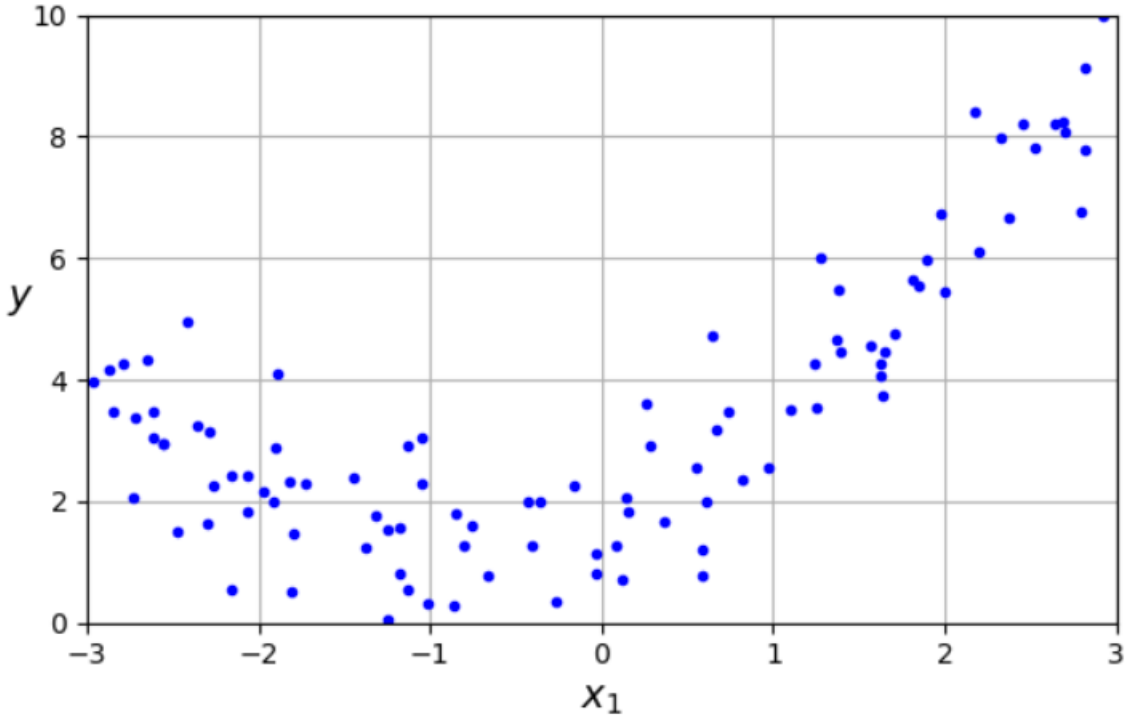
Polynomial Regression

- It will be **hard to find a line** that fits it well. However, a **curve will fit** the data well, as you can see in the right part of the figure.
- **Polynomial regression** helps us find this curve



Polynomial Regression

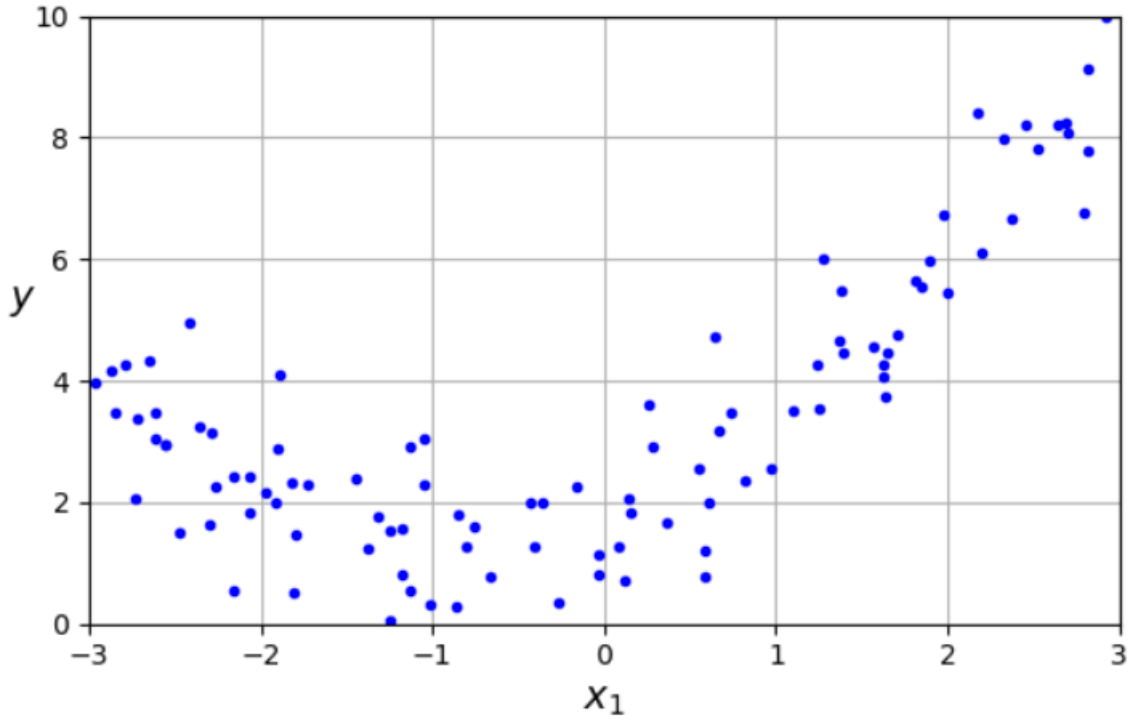
- What if your data is more complex than a simple straight line?




```
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.5 * X**2 + X + 2 + np.random.randn(m, 1)
```

Polynomial Regression

- Use a linear model
- A simple way to do this is to **add powers of each feature** as new features, then **train a linear model** on this extended set of features.
- This technique is called **Polynomial Regression**



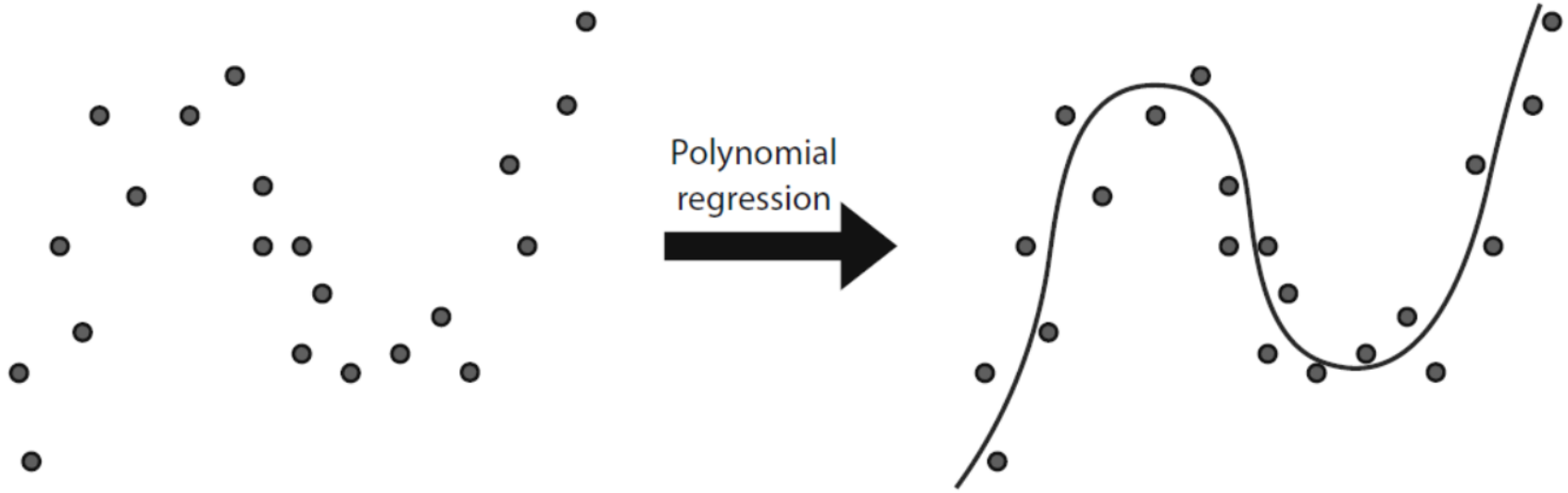
Polynomial Regression

- The process to train a polynomial regression model is similar to the process of training a linear regression model.
- The only difference is that we **need to add more columns** to our dataset before we apply linear regression

x	x^2	x^3	Label
−5.0	25.0	−125.0	−81.1
−4.5	20.25	−91.125	−57.3
−4.0	16.0	−64.0	−52.2
−3.5	12.25	−42.875	−34.7
−3.0	9.0	−27.0	−18.3
−2.5	6.25	−15.625	−21.5
−2.0	4.0	−8.0	−4.6
−1.5	2.25	−3.375	3.2
−1.0	1.0	−1.0	4.5
−0.5	0.25	−0.125	5.3
0.0	0.0	0.0	4.0
0.5	0.25	0.125	8.8
1.0	1.0	1.0	10.9
1.5	2.25	3.375	21.2
2.0	4.0	8.0	18.3
2.5	6.25	15.625	43.4
3.0	9.0	27.0	47.5
3.5	12.25	42.875	67.3
4.0	16.0	64.0	79.1
4.5	20.25	91.125	106.5

Polynomial Regression

- For example, if we decide to fit a polynomial of degree 3 to the data just mentioned, we need to **add two columns**: one corresponding to the **square** of the feature and one corresponding to the **cube** of the feature.



Polynomial Regression

- Clearly, a straight line will never fit this data properly.
- So let's use Scikit-Learn's **PolynomialFeatures** class to transform our training data, **adding the square** (2nd-degree polynomial) of each feature in the training set as new features

```
from sklearn.preprocessing import PolynomialFeatures

poly_features = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly_features.fit_transform(X)
X[0]
```

```
array([-0.75275929])
```

```
X_poly[0]
```

```
array([-0.75275929,  0.56664654])
```

Polynomial Regression

- X_poly now contains the original feature of X plus the square of this feature.
- Now you can fit a LinearRegression model to this extended training data

```
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y)
lin_reg.intercept_, lin_reg.coef_
```

```
(array([1.78134581]), array([[0.93366893, 0.56456263]]))
```

Polynomial Regression

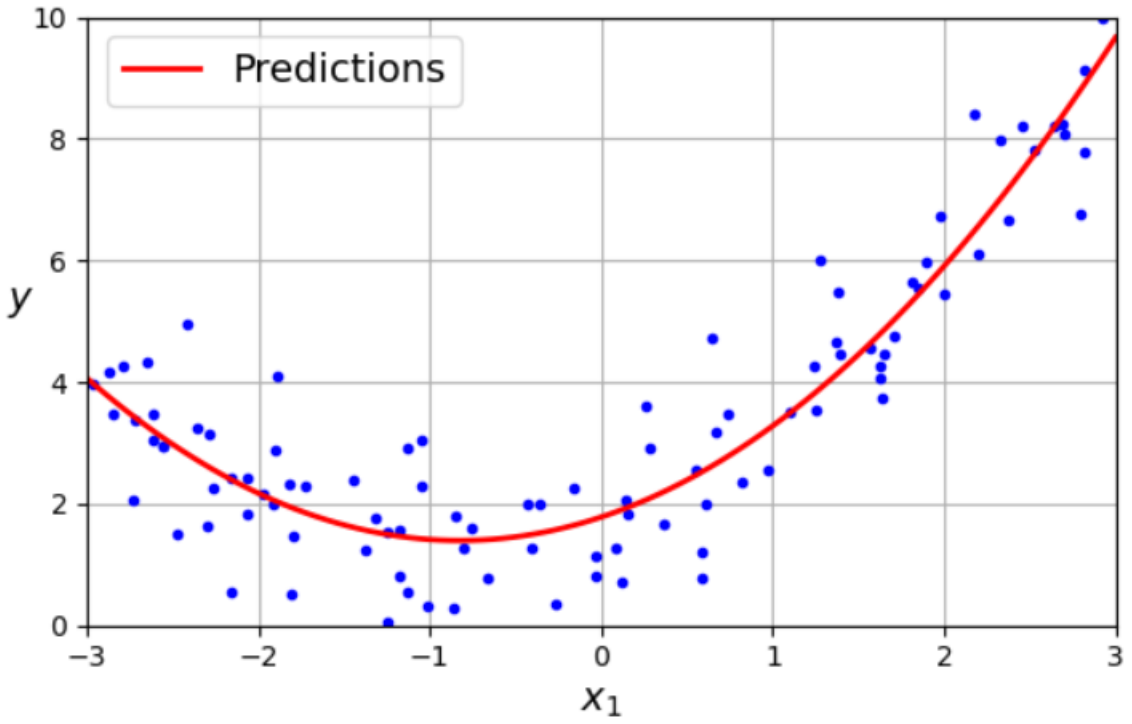
```
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y)
lin_reg.intercept_, lin_reg.coef_
```

```
(array([1.78134581]), array([[0.93366893, 0.56456263]]))
```

$$\hat{y} = 0.56\, x_1^2 + 0.93\, x_1 + 1.78$$

$$y = 0.5x_1^2 + x_1 + 2 + \textit{Gaussian noise}$$

```
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.5 * X**2 + X + 2 + np.random.randn(m, 1)
```



Example

- Let's fit a 2nd-degree polynomial (quadratic) regression:
- $y = \beta_0 + \beta_1 x + \beta_2 x^2$

Hours studied (x)	Test score (y)
1	65
2	70
3	80
4	82
5	85
6	87

Example

- $y = \beta_0 + \beta_1 x + \beta_2 x^2$

Hours studied (x)	Test score (y)
1	65
2	70
3	80
4	82
5	85
6	87

- $y = \beta_0 + \beta_1 x + \beta_2 x^2$
- **Step 1**: Set up the system of equations
- For each data point, we have:
- $65 = \beta_0 + \beta_1(1) + \beta_2(1^2)$
- $70 = \beta_0 + \beta_1(2) + \beta_2(4)$
- $80 = \beta_0 + \beta_1(3) + \beta_2(9)$
- $82 = \beta_0 + \beta_1(4) + \beta_2(16)$
- $85 = \beta_0 + \beta_1(5) + \beta_2(25)$
- $87 = \beta_0 + \beta_1(6) + \beta_2(36)$

Example

- $y = \beta_0 + \beta_1 x + \beta_2 x^2$

Hours studied (x)	Test score (y)
1	65
2	70
3	80
4	82
5	85
6	87

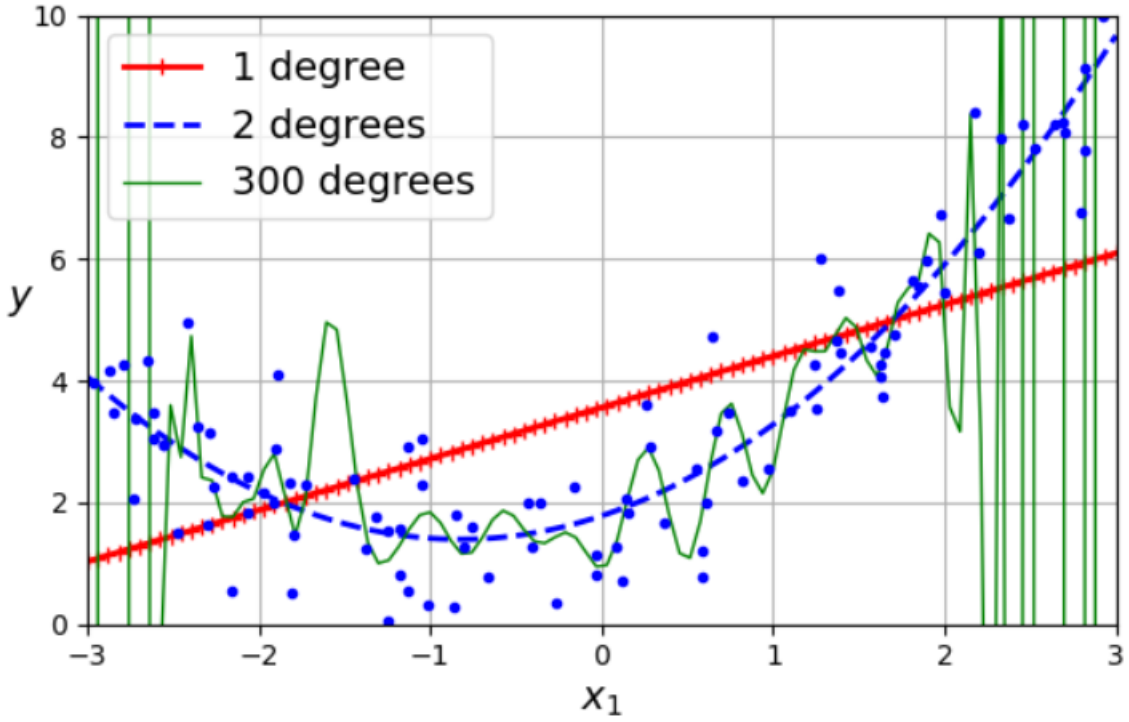
- **Step 2**: Solve for β_0, β_1, and β_2
- Using a statistical software or matrix operations, we find:
- $\beta_0 \approx 59.14$
- $\beta_1 \approx 9.21$
- $\beta_2 \approx -0.76$
- Step 3: Write the fitted polynomial equation
- $y = 59.14 + 9.21x - 0.76x^2$

Feature Correlation

- When there are **multiple features**, polynomial regression is capable of finding relationships between features, which is something a plain linear regression model cannot do.
- This is made possible by the fact that PolynomialFeatures also adds all combinations of features up to the given degree
- if there were two features a and b, PolynomialFeatures with degree=3 would not only add the features a^2 , a^3 , b^2 , and b^3 , but also the combinations ab, a^2b, and ab^2 .

How do we decide on this degree?

- This high-degree polynomial regression model is severely overfitting the training data, while the linear model is underfitting it



Polynomial Regression - Overfitting

- Can be more **flexible** but also more **prone to overfitting** than linear regression
- Increased complexity: Higher-degree polynomials can create more **intricate (معقّد) curves**, potentially **fitting noise** in the data.
- More parameters: Each degree added introduces a **new parameter**, increasing the model's ability to conform to the training data.
- *Sensitivity to outliers*: Higher-degree polynomials can be dramatically **affected by extreme** data points.

How do we decide on this degree?

- How do we decide on this degree? Do we want a line (degree 1), a parabola (degree 2), a cubic (degree 3), or some curve of degree 50?

- This question is important, - Recall back to **overfitting**, underfitting, and regularization

Next Lecture

- Polynomial regression