Polynomial Regression

Machine Learning Primer Course

Georgios C. Anagnostopolous and Spencer G. Lyon

October 2, 2020

Purpose

Show how polynomial regression – fitting a polynomial curve to data – can be performed via linear regression

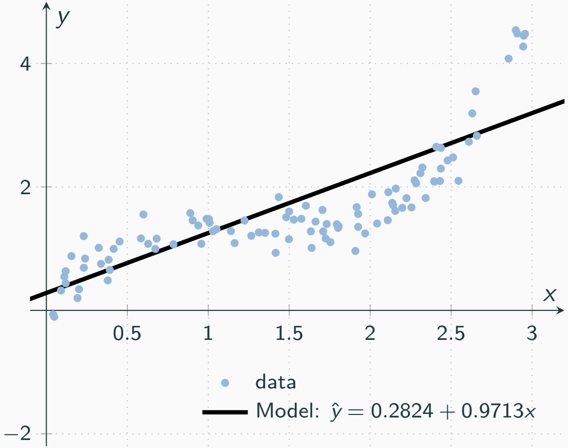
Table of contents

Motivation

Motivation

Often when modeling a single feature/single response data (x and y are scalars) it makes little sense to fit a line:

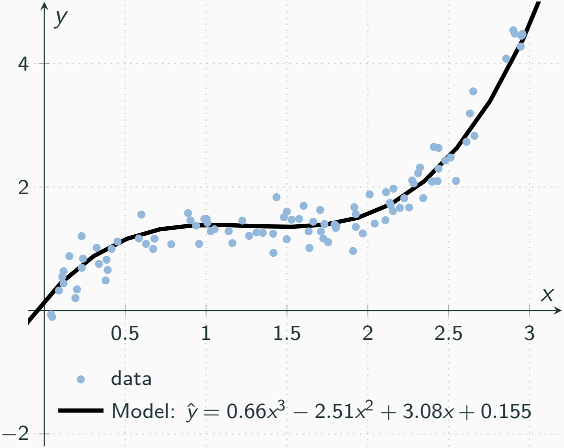
$$y = w_1 x + w_0$$



Motivation continued

It might make more sense to use a cubic polynomial in this case

$$y = w_3x^3 + w_2x^2 + w_1x + w_0$$



Theory

How does it work?

- The previous example becomes

$$y = \underbrace{\begin{bmatrix} x^3 & x^2 & x & 1 \end{bmatrix}}_{\triangleq \phi^T(x)} \begin{bmatrix} w_3 \\ w_2 \\ w_1 \\ w_0 \end{bmatrix} = \phi^T(x)\mathbf{w}$$

- Notice the prediction model is non-linear in the feature x
- ... however it is still linear in the parameters $\mathbf{w}$

$$\Longrightarrow$$

How does it work?

- The previous example becomes

$$y = \underbrace{\begin{bmatrix} x^3 & x^2 & x & 1 \end{bmatrix}}_{\triangleq \phi^T(x)} \begin{bmatrix} w_3 \\ w_2 \\ w_1 \\ w_0 \end{bmatrix} = \phi^T(x)\mathbf{w}$$

- Notice the prediction model is non-linear in the feature x
- ... however it is still linear in the parameters $\mathbf{w}$

$$\Longrightarrow$$

- We can use linear regression to fit a polynomial

Fitting Polynomials

- To fit a D-degree polynomial, each scalar x needs to be transformed into a feature vector:

$$\phi : x \mapsto \phi(x) \triangleq \begin{bmatrix} x^D \\ x^{D-1} \\ \vdots \\ x^2 \\ x \\ 1 \end{bmatrix} \in \mathbb{R}^{D+1}$$

- Then the prediction model for polynomial regression is:

$$\hat{y} = f(x|\mathbf{w}) = \phi^T(x)\mathbf{w}$$

The New Feature Matrix

- Training data: $\{x_n, y_n\}_{n=1}^{N}$
- We construct a new feature matrix by transforming each x_n using ϕ:

$$\Phi \triangleq \begin{bmatrix} \phi^T(x_1) \\ \phi^T(x_2) \\ \vdots \\ \phi^T(x_N) \end{bmatrix}$$

$$= \begin{bmatrix} x_1^D & x_1^{D-1} & \cdots & x_1^2 & x_1 & 1 \\ x_2^D & x_2^{D-1} & \cdots & x_2^2 & x_2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ x_N^D & x_N^{D-1} & \cdots & x_N^2 & x_N & 1 \end{bmatrix} \in \mathbb{R}^{N\times(D+1)}$$

What is Happening?

Hard to visualize and to appreciate at this point in time, but the following two statements are equivalent:

1. Fitting a polynomial curve, where the (x, y) samples reside[1]
2. Fitting a hyper-plane in the $(D+2)$-dimensional space, where the $(\phi(x), y)$ samples reside[2]

[1] both being scalars, they reside on the (x, y) plane
[2] $\phi(x)$ is $(D+1)$-dimensional and y is scalar; thus, in $D+1+1$ dimensions

Demo

Play time!

- Let's play with the jupyter notebooks
  - `PolyRegDemoMakeData.ipynb`
  - `PolyRegDemoFit.ipynb`

- What to do: Change the degree of polynomial to be fitted and/or number of training points and observe impact on fitted model

Remarks

- In practice we can't increase `D` arbitrarily
- There are some computational reasons:
  - As `D` increases, the matrix $R = \Phi^T\Phi$ becomes singular $\implies$ `np.linalg.inv(R)` is unreliable
  - Using `np.linalg.pinv(` Φ `)` can support larger `D`, but still has issues

[3] Statistically, this is known as multi-colinearity
[4] This makes interpreting coefficients less reliable

Remarks

- In practice we can't increase `D` arbitrarily
- There are some computational reasons:
  - As `D` increases, the matrix $R = \Phi^T\Phi$ becomes singular $\implies$ `np.linalg.inv(R)` is unreliable
  - Using `np.linalg.pinv( Φ )` can support larger `D`, but still has issues
- And (at least one) subtle statistical theory one:
  - For large `D`, x^D and x^{D-1} look quite similar[3]
  - In practice you may see small changes in x lead to large changes in weights $\mathbf{w}$[4]
  - Also increases risk of overfitting – stay tuned for next lecture!

[3] Statistically, this is known as multi-colinearity

[4] This makes interpreting coefficients less reliable

Feature Engineering

- Transforming raw input features X in potentially non-linear ways is common practice in ML
- This is known as feature engineering and is a significant part of applying classical (non-deep) ML techniques
- To do properly, requires an understanding of mathematics behind the algorithms/transformations **and** domain knowledge
- We'll encounter this concept throughout the course

Feature Engineering

- Transforming raw input features X in potentially non-linear ways is common practice in ML
- This is known as feature engineering and is a significant part of applying classical (non-deep) ML techniques
- To do properly, requires an understanding of mathematics behind the algorithms/transformations **and** domain knowledge
- We'll encounter this concept throughout the course

Deep Learning

One benefit of deep learning is *automated feature engineering*, meaning you feed in raw X and neural network learns feature transformations that lead to optimal reduction in loss