Machine Learning Course - CS-433

Underfitting and Overfitting

Oct 1, 2020

EPFL

Motivation

Models can be *too limited* or they can be *too rich*. In the first case we cannot find a function that is a good fit for the date in our model. We then say that we underfit. In the second case we have such a rich model family that we do not just fit the underlying function but we in fact fit the noise in the date as well. We then talk about an overfit. Both of these phenomena are undesirable. This discussion is made more difficult since all we have is data and so we do not know a priori what part is the underlying signal and what part is noise.

Underfitting with Linear Models

It is easy to see that linear models might underfit. Consider a scalar case as shown in the figure below.
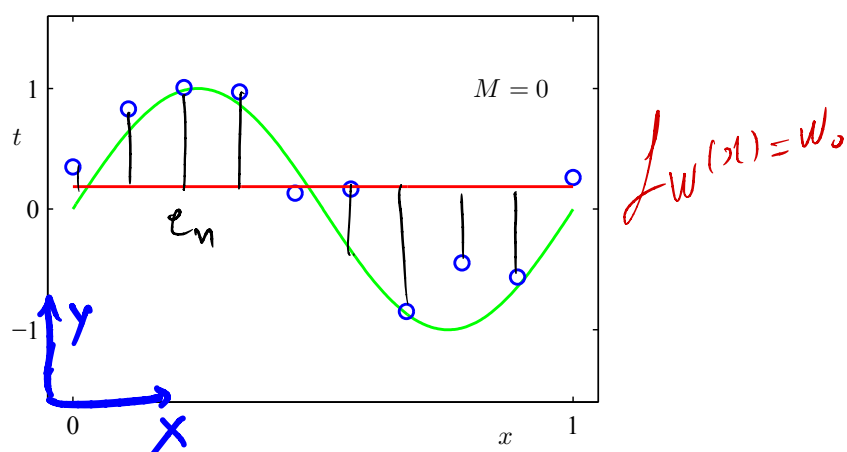
The solid curve is the underlying function and the circles are the actual data. E.g., we assume that there is a scalar function $g(x)$ but that we do not observe $g(x_n)$ directly but only a noisy version of it, $y_n = g(x_n) + Z_n$, where Z_n is

the noise. The noise might be due for example to some measurement inaccuracies. The y_n are shown as blue cirles. If our model family consists of only linear functions of the scalar input x, i.e., $\mathcal{F} = \{f_w(x) = wx\}$, where w is a scalar constant (the slope of the function), then it is clear that we cannot match the given function accurately, regardless how many samples we get and how small the noise is. We therefore will underfit.

Extended/Augmented Feature Vectors From the above example it might seem that linear models are too simple to ever overfit. But in fact, linear models are highly prone to overfitting, much more so than complicated models like neural nets.

Since linear models are inherently not very rich the following is a standard "trick" to make them more powerful.

In order to increase the representational power of linear models we typically "augment" the input. E.g., if the input (feature) is one-dimensional we might add a polynomial basis (of arbitrary degree M),

$$\phi(x_n) := [1, x_n, x_n^2, x_n^3, \ldots, x_n^M]$$

[handwritten left margin: Before: $X_n \in R$; after: $\phi(x_n) \in R^{M+1}$]

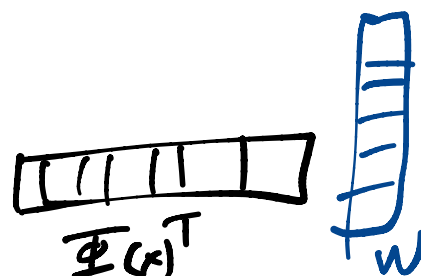*[handwritten above x_n: $\in R$]*

so that we end up with an extended feature vector.

We then fit a linear model to this extended feature vector $\phi(x_n)$:

$$y_n \approx w_0 + w_1 x_n + w_2 x_n^2 + \ldots + w_M x_n^M =: \phi(x_n)^\top \mathbf{w}.$$
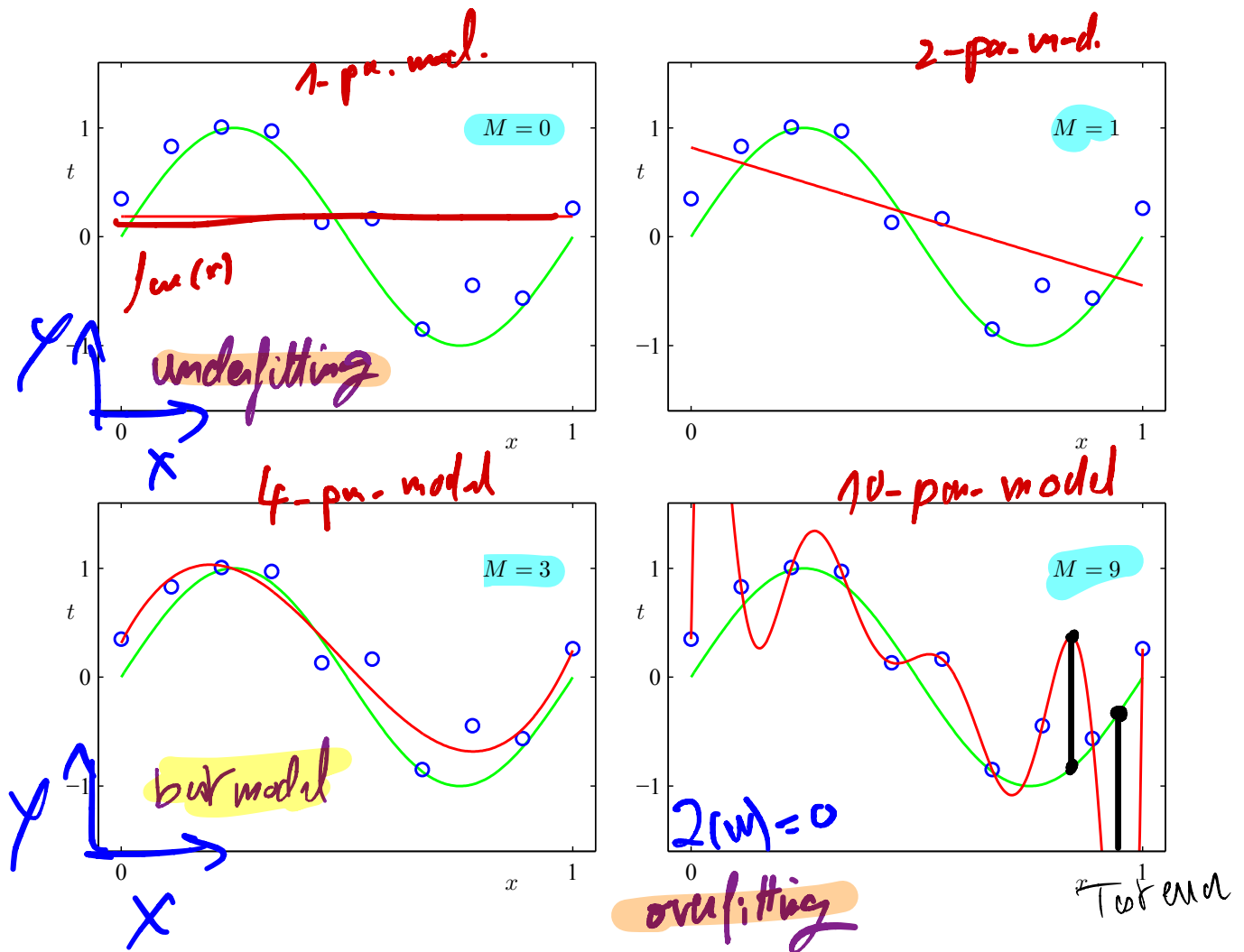
[handwritten: $f_w(\phi(x)) = f_w(x_n)$]

[handwritten bottom: M: model complexity]

[handwritten bottom right: $\phi(x)^\top$ ... w]

Overfitting with Linear Models

In the following four figures, circles are data points, the green line represents the "true function", and the red line is the model. The parameter M is the maximum degree in the polynomial basis.

For $M = 0$ (the model is a constant) the model is underfitting and the same is true for $M = 1$. For $M = 3$ the model fits the data fairly well and is not yet so rich as to fit in addition the small "wiggles" caused by the noise. But for $M = 9$ we now have such a rich model that it can fit every single data point and we see severe overfitting taking place. What can we do to avoid overfitting? If you increase the amount of data (increase N, but keep M fixed), overfitting

might reduce. This is shown in the following two figures where we again consider the same model complexity $M = 9$ but we have extra data ($N = 15$ or even $N = 100$).
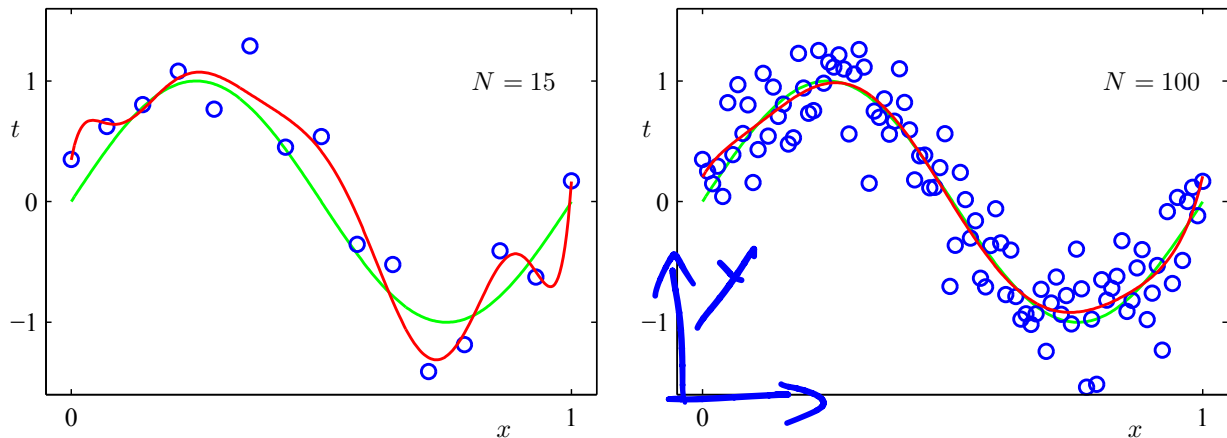
fight overfitting

① simpler model

② more data

(Linear on $\Phi(x)$)

A Word About Notation

If it is important to distinguish the original input x from the augmented input then we will use $\phi(x)$ to denote this augmented input vector. But we can consider this augmentation as part of the pre-processing, and then we might simply write $\mathbf{x}$ to denote the input. This will save us a lot of notation.

Additional Materials

Read about overfitting in the paper by Pedro Domingos (Sections 3 and 5 of "A few useful things to know about machine learning").