MLISP: Machine Learning in Signal Processing

Lecture 7

Prof. V. I. Morgenshtern

Scribe: M. Elminshawi

Illustrations: CS 229, Stanford, A. Ng; The ESL, Hastie, Tibshirani, Friedman

Agenda:

1. Making linear regression more flexible
2. Polynomial features
3. Feature design adapted to the problem
4. Regularization
5. Splines

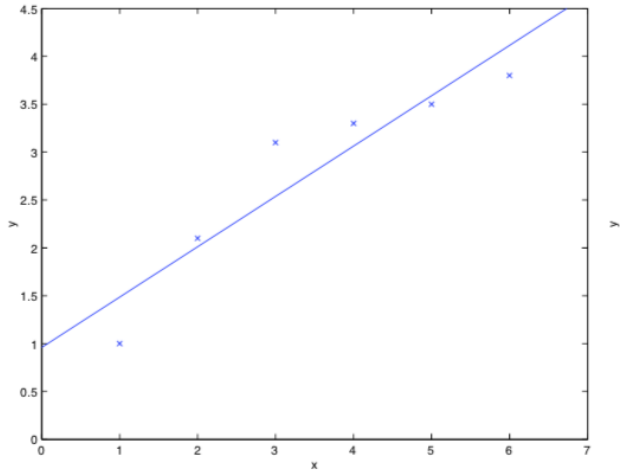
1 Making linear regression more flexible

In the previous lecture we saw that the nearest neighbors algorithm works well in very low dimensional spaces, say $p = 2$ or $p = 3$. We also saw that this approach fails badly when the dimension is modestly large, say $p = 10$.

In contrast, the linear model works well in very high dimensions, even in the presence of noise, assuming that the true model is nearly linear. More often than not the true model is not linear. What can we do then?

1.1 Features

We will start with an example where we show how to estimate accurately a *nonlinear* function of one variable. Suppose we have data points as in the following figure:



We can then fit the linear regression line $y = \theta_0 + \theta_1 x$ and get the results as shown above. However, it seems that the data presents more structure: the data points seem to lie on a curve.

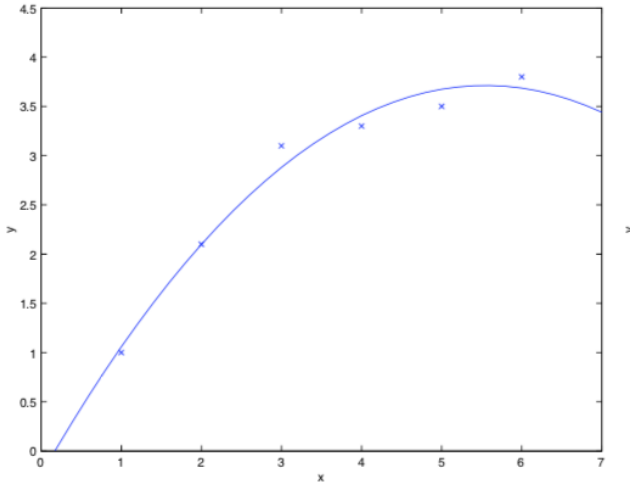
To capture this behavior we can design a *new artificial feature* and fit the polynomial of the second degree

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 \tag{1}$$

by using the linear regression approach. Concretely, we add an extra column to the data matrix, calculate values in the column based on the values in the column x and then treat the new column as a new variable:

y	x	x^2
100,000	75 m^2	$(75)^2$
200,000	130 m^2	$(130)^2$
200,000	275 m^2	$(275)^2$
.	.	.
.	.	.

When we fit the function in (1), which is still linear in $\boldsymbol{\theta} = [\theta_0, \theta_1, \theta_2]^T$, we get something like this:

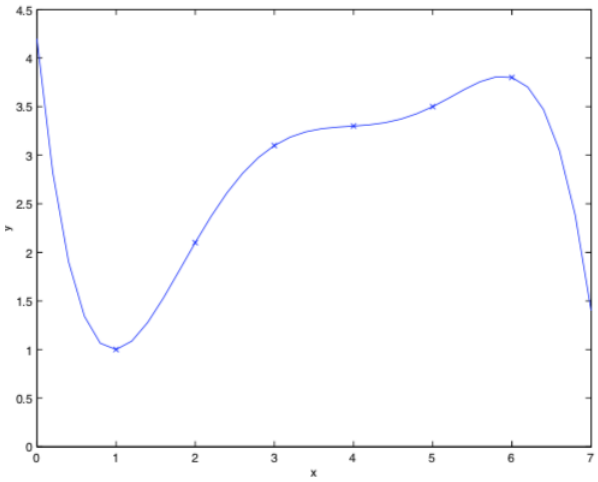


This is clearly a better fit. But how do we select features in general? How many features and which ones?

For example, if we take the polynomial of degree 5

$$y = \sum_{j=0}^{5} \theta_j x^j \tag{2}$$

we would get:



If y represents the house prices, this is not how house prices usually behave!

The reason for failure here is that to fit a linear model robustly, we need to have many more data points than the number of variables in the model. A reasonable rule of thumb might be that with n data points you might be able to fit, say $n/3$ coefficients. In the example above, we have 6 variable and 6 data points. Hence, the fit would not be very stable in the presence of noise. These dimension counting considerations apply for other classes of models as well.

There are two methods to solve a problem like this.

1.2 Feature design adapted to the problem

The first method is to use cleverly designed features. On the one hand, these features should provide the necessary flexibility for the problem at hand. On the other hand, they should not require fitting unnecessarily many parameters. Let's illustrate this method with a simple example.

Suppose originally we have $p = 3$ features: x_1, x_2, x_3. If we want to generate every term for polynomial of degree 3 we get $p' = 20$ terms as follows:

$$\begin{array}{cccccc} 1 & & & & & \\ x_1 & x_2 & x_3 & & & \\ x_1^2 & x_2^2 & x_3^2 & x_1x_2 & x_1x_3 & x_2x_3 \\ x_1^2x_2 & x_1^2x_3 & x_2^2x_1 & x_2^2x_3 & x_3^2x_1 & x_3^2x_2 \\ x_1^3 & x_2^3 & x_3^3 & & & \\ x_1x_2x_3 & & & & & \end{array}$$

In general polynomials in p variables of degree d will have $p' = \binom{p+d}{d}$ terms. To see this, assume for concreteness that $d = 9$ and $n = 3$. Then the monomial term $x_1^3x_2x_3^3$ can be written as

$$11 \times x_1x_1x_1 \times x_2 \times x_3x_3x_3.$$

In this encoding it is clear that the number of possible monomials is equal to the number of possible locations of $n = 3$ $\times$ signs out of $d = 9$ possible locations, which gives $p' = \binom{p+d}{n} = \binom{p+d}{d}$.

This is the number of coefficients we would need to fit if we use this model for learning.

A standard[1] lower bound on the binomial coefficients gives:

$$p' = \binom{p+d}{d} \geq \left(\frac{p+d}{d}\right)^d.$$

For example, for $d = 3$ this gives $p' \approx \frac{(p+3)^3}{27} \sim p^3$. To be able to fit this model, we would need to have many more than p^3 data points, which might not be feasible for large p, say, $p > 1000$.

Also, consider the case $d = p$. Observe that in this case p' grows exponentially with d:

$$p' = \left(\frac{p+d}{d}\right)^d = 2^d$$

so for $p = 100$ and $d = 100$, p' would be more than the number of atoms in the Universe and fitting such model would be out of the question for any dataset.

Sometimes the structure of the problem allows us to generate much less features. For example, if we know that the variables interact in small clusters, but not across clusters, we can write our model as a linear combination of smaller models:

$$\theta_1 f_1(x_1, x_2, x_3) + \theta_2 f_2(x_4, x_5) + \theta_3 f_3(x_6) + ... \tag{3}$$

[1] see, for example, "A brief note on estimates of binomial coefficients" by S. Das

Here, the nonlinear features are hidden inside the functions $f_i(\cdot)$ but across the functions everything is linear. Then we can model $f_i(\cdot)$ as high degree polynomials but in few variables and keep the model complexity under control.

1.3 Regularization

The second method is to invent many features, p', many more than we could possibly fit based on the number of data points, n, that we have. For example:

$$y = \theta_0 + \theta_1 x + \theta_2 x^7 + \theta_3 \cos x + ... \tag{4}$$

Most of these features would likely be redundant.

Consider the data matrix $\mathbf{X}$ that is $n \times p'$. To fit this model via linear regression we would like to solve the least squares problem:

$$\min_{\theta} \quad \underbrace{\|\mathbf{y} - \mathbf{X}\theta\|_2^2}_{\text{least squares regression}} \quad . \tag{5}$$

However, when the number of features, p' is larger than n, infinitely many different vectors θ will satisfy $\mathbf{y} = \mathbf{X}\theta$. [Please convince yourself that this statement is true!] Therefore, we cannot obtain a unique solution for θ by solving the least squares problem. Instead we will solve this convex optimization problem:

$$\min_{\theta} \quad \underbrace{\|\mathbf{y} - \mathbf{X}\theta\|_2^2}_{\text{least squares term}} + \underbrace{\lambda \|\theta\|}_{\text{regularization term}} \quad . \tag{6}$$

Above, we can choose, for example, to use the l2 norm, $\|\theta\|_2$, or the l1 norm, $\|\theta\|_1$. The meaning of the regularization term is this: among the infinitely many solutions of the least squares problem, choose the one that is *simple* in some way. Simplicity might mean different things for different problems. For example, $\|\theta\|_2$ would enforce the property that θ does not have elements that are too large; $\|\theta\|_1$ would enforce the property that most of the elements of θ are zero. The weight λ is a metaparameter that controls how much we value simplicity, $\|\theta\|$ small, versus data fidelity, $\|\mathbf{y} - \mathbf{X}\theta\|_2^2$, small.

1.4 Summary

This subsection is a preview. We have seen that it is important to use feature models that are economical and are well adapted to the problem at hand. Polynomial features are reasonable, but most of the time, splines are a better choice. For image and sound data, wavelet-based features are very useful. Our plan is to study splines and their applications, then wavelets and their applications, then discuss powerful regularization methods based on convex optimization. We begin with splines.

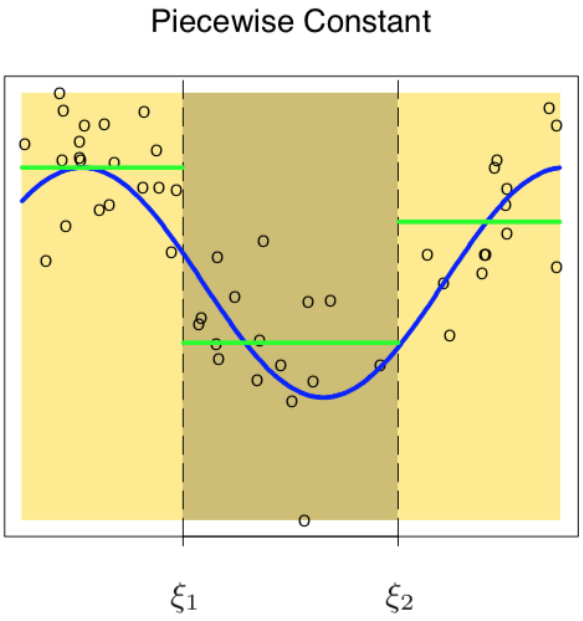
2 Splines

Polynomial are limited by their global nature: tweaking the coefficients in one point can disproportionately affect the behavior in remote points. Splines are *local* polynomial representations and, due to locality, avoid these problems.

Suppose we are trying to approximate a *continuous* nonlinear function $f(x)$ of one variable based on noisy samples. The samples are generated according to:

$$\mathrm{y}^{(i)} = f(x^{(i)}) + \mathrm{n}^{(i)}$$

where $\mathrm{n}^{(i)} \sim \mathcal{N}(0, \sigma^2)$ is the noise that is independent across i, and the samples $x^{(i)}$ are fixed. In the figure below, the data points are the small black dots and $f(\cdot)$ is the function depicted in blue:



Consider the interval $[0, 1]$ and divide it into three subintervals: $[0, \varepsilon_1], [\varepsilon_1, \varepsilon_2], [\varepsilon_2, 1]$. We will represent $f(\cdot)$ by a polynomial in each interval.

Start with piece-wise constants, with three basis function:

$$\begin{aligned} h_1(x) &= I(x \leq \varepsilon_1) \\ h_2(x) &= I(\varepsilon_1 \leq x \leq \varepsilon_2) \\ h_3(x) &= I(\varepsilon_2 \leq x) \end{aligned}$$

where $I(\cdot)$ denotes the indicator function of the corresponding interval (one on the interval and zero everywhere else). Approximating $f(\cdot)$ in terms of these basis functions we have:

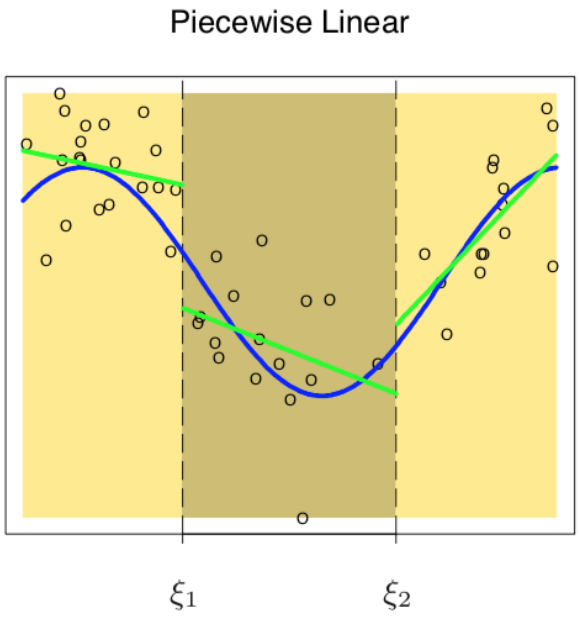
$$f(x) = \sum_{m=1}^{3} \theta_m h_m(x).$$

Fitting the coefficients θ_m by least squares, we find:

$$\hat{\theta}_m = \bar{\mathrm{y}}_m$$

where $\bar{\mathrm{y}}_m$ denotes the average of $\mathrm{y}^{(i)}$'s in the mth interval.

Next, consider piece-wise linear fit:



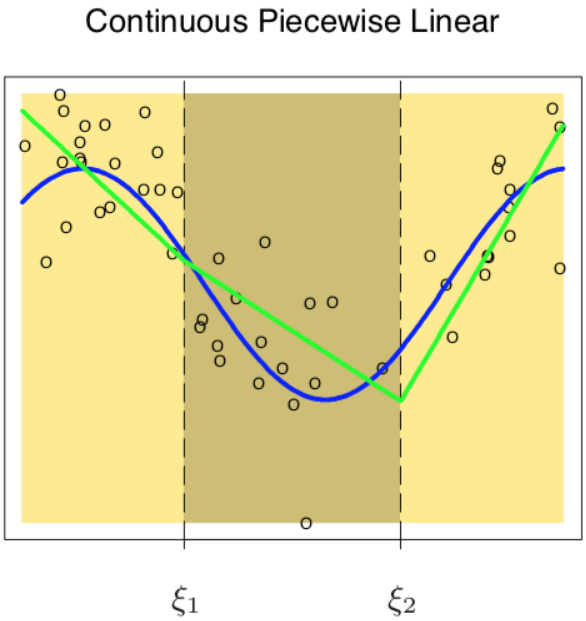
Three additional basis functions are needed:

$$h_{m+3}(x) = h_m(x)x, \ m = 1, \ldots, 3.$$

and the approximation for $f(\cdot)$ now has 6 terms

$$f(x) = \sum_{m=1}^{6} \theta_m h_m(x).$$

We might prefer a continuous fit:



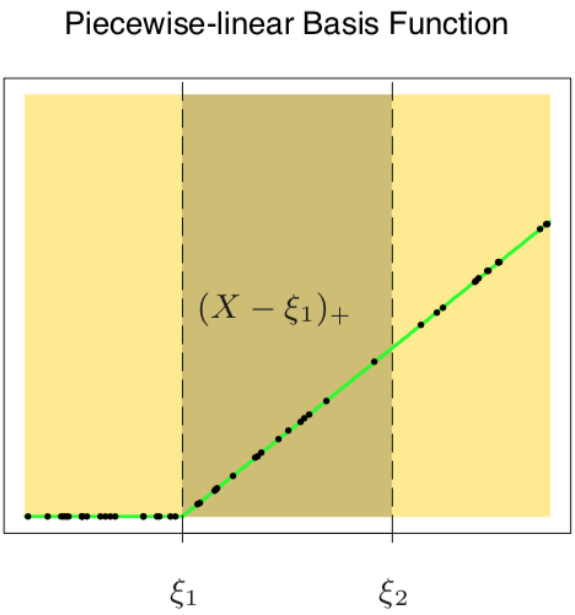
These continuity restrictions lead to *linear* constraints on the parameters. For example: $f(\varepsilon_1^-) = f(\varepsilon_1^+)$ implies

$$\theta_1 + \varepsilon_1\theta_4 = \theta_2 + \varepsilon_1\theta_5$$

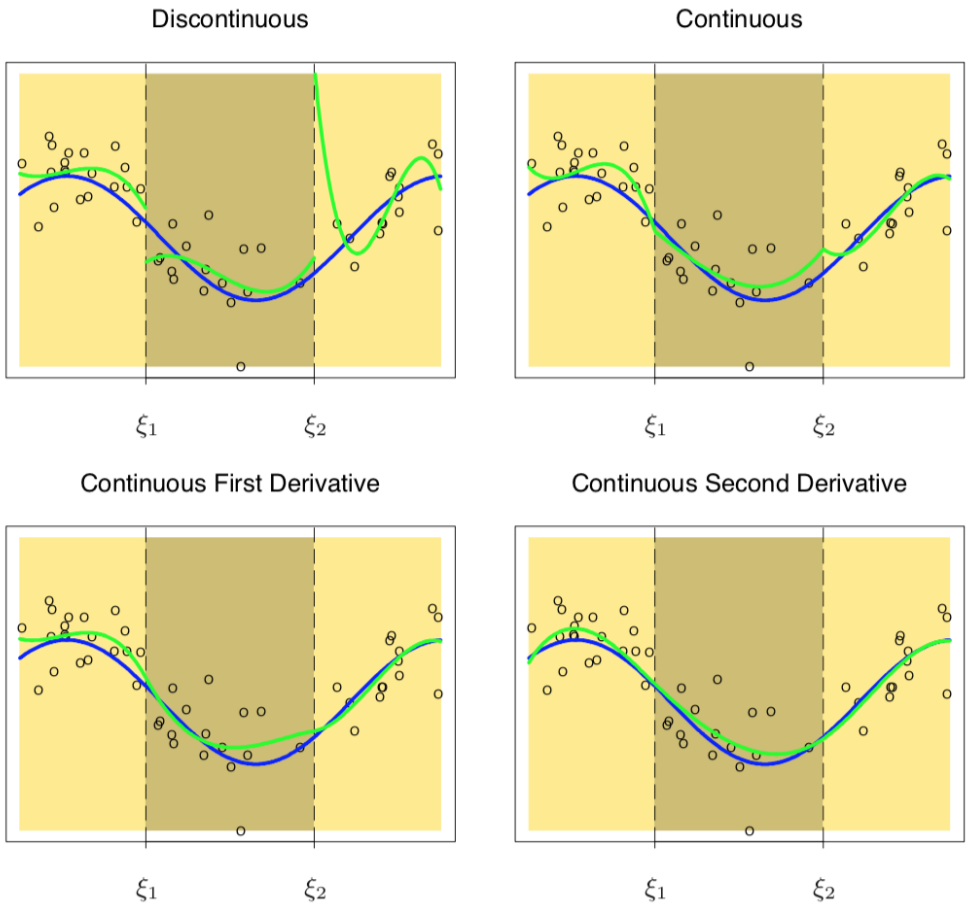
where $f(\varepsilon_1^-)$ and $f(\varepsilon_1^+)$ denote the values of $f(\cdot)$ immediately to the left and immediately to the right of ε_1, respectively. Since we have two restrictions $f(\varepsilon_1^-) = f(\varepsilon_1^+)$ and $f(\varepsilon_2^-) = f(\varepsilon_2^+)$, we have 4 free parameters out of 6 total. A more direct way to proceed is to use the basis that incorporates the constraints upfront:

$$\begin{aligned} h_1(x) &= 1 \\ h_2(x) &= x \\ h_3(x) &= (x - \varepsilon_1)_+ \\ h_4(x) &= (x - \varepsilon_2)_+ \end{aligned}$$

where $(\cdot)_+$ denotes the positive part. These 4 functions are continuous on the interval $[0, 1]$. For example, the function $h_3(x)$ looks like this:



We often prefers smoother functions, and therefore we might prefer to increase the continuity at the knots. To do this, piece-wise *cubic* polynomials are often used.



Cubic spline is a piece-wise cubic function that is continuous and has continuous first and second derivatives at the knots. The following basis represents a cubic spline with knots ε_1 and ε_2:

$$
\begin{aligned}
h_1(x) &= 1 \\
h_2(x) &= x \\
h_3(x) &= x^2 \\
h_4(x) &= x^3 \\
h_5(x) &= (x-\varepsilon_1)_+^3 \\
h_6(x) &= (x-\varepsilon_2)_+^3.
\end{aligned}
$$

In your homework, you will show that the 6 functions indeed define the linear space of piece-wise cubic functions with first and second derivatives at the knots. The number of degrees of freedom for a cubic spline with two nodes can be calculated as follows:

$$
\begin{aligned}
D &= \#\text{ free parameters} \\
&= (3 \text{ regions}) \times (4 \text{ parameters per region}) - (2 \text{ knots}) \times (3 \text{ constraints per knot}) \\
&= 6.
\end{aligned}
$$

More generally, *order-M spline* with knots $\varepsilon_j, j = 1, \ldots, K$ is a piece-wise polynomial of degree $M-1$ and has continuous derivatives up to order $M-2$. For example,

- A cubic spline has $M = 4$.
- Piece-wise constant function has $M = 1$.
- Continuous piece-wise linear function has $M = 2$.

People are rarely interested in $M > 4$.