Least squares for nonlinear regression

Let us return to linear regression, which has been one of the main motivations for studying the least squares problem.

Recall that if we have data (x_m, y_m) for $m = 1, \ldots, M$, we want to find a function $f(x)$ such that $f(x_m) \approx y_m$ for each $m = 1, \ldots, M$.

As discussed on pages 3 and 4 of the course notes, we consider functions of the form

$$f(x) = \sum_{n=1}^{N} \alpha_n \phi_n(x),$$

where the "feature" functions $\phi_n(x)$ can be anything. We will abbreviate

$$\boldsymbol{\phi}(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \vdots \\ \phi_N(x) \end{bmatrix}$$

so that we can write $f(x) = \boldsymbol{\alpha}^T \boldsymbol{\phi}(x)$. We often call $\boldsymbol{\phi}(x)$ the "feature map."

Recall[1] that the least-squares estimate of the coefficients $\alpha_1, \ldots, \alpha_N$ is the solution to

$$\underset{\boldsymbol{\alpha} \in \mathbb{R}^N}{\text{minimize}} \ \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{\alpha}\|_2^2, \tag{1}$$

[1] It is important to distinguish between the sample points $x_m, m = 1, \ldots, M$ and the vector $\boldsymbol{x} \in \mathbb{R}^N$ that we have seen in the previous several notes. In both regression and optimization, "x" is usually the argument of some function, but whereas in optimization the function is what we are trying to minimize (e.g., $\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2$), in regression we are most interested in the function $f(x)$ that fits the data! In these notes, $\boldsymbol{\alpha} \in \mathbb{R}^N$ will once again take the place of $\boldsymbol{x}$.

where

$$\boldsymbol{A} = \begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \cdots & \phi_N(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \cdots & \phi_N(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_M) & \phi_2(x_M) & \cdots & \phi_N(x_M) \end{bmatrix} = \begin{bmatrix} — & \boldsymbol{\phi}(x_1)^{\mathrm{T}} & — \\ — & \boldsymbol{\phi}(x_2)^{\mathrm{T}} & — \\ & \vdots & \\ — & \boldsymbol{\phi}(x_M)^{\mathrm{T}} & — \end{bmatrix},$$

and

$$\boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix}.$$

Everything we have studied in lectures 3–5 can be applied to this least squares optimization problem. We can set $\widehat{\boldsymbol{\alpha}}_{\mathrm{ls}} = \boldsymbol{A}^{\dagger}\boldsymbol{y}$ (or use Tikhonov regularization, etc.) and use it as our set of regression coefficients.

Computation with kernel functions

Although, so far, we have mostly considered least squares in the case that $M \geq N$ (i.e., the number of samples is larger than the number of coefficients/features), we can also consider the "underdetermined" case where there are more features than samples. In this case, the matrix $\boldsymbol{A}$ has a nontrivial null space, so there is no unique solution to (1). However, recall that even in this case, $\widehat{\boldsymbol{\alpha}}_{\mathrm{ls}} = \boldsymbol{A}^{\dagger}\boldsymbol{y}$ gives us the solution with minimal $\|\boldsymbol{\alpha}\|_2^2$. In these notes, we will use Tikhonov regularization, which also gives us a unique solution by penalizing $\|\boldsymbol{\alpha}\|_2^2$ in the optimization problem.

Nonlinear least squares as presented above becomes difficult if the number N of features is huge. Doing computations with the resulting

extremely wide matrix $\boldsymbol{A}$ may be difficult. If fact, there are even interesting situations where we want to estimate a function with an *infinite* set of features! For example, consider a Fourier series

$$f(x) = \sum_{n=-\infty}^{\infty} b_n e^{j2\pi nx}$$

on the interval $[0, 1]$. In principle, there are infinitely many coefficients b_n we would need to estimate in order to do regression with a Fourier series.

To solve this problem, recall that in regression, we are less interested in the coefficients $\alpha_1, \ldots, \alpha_N$ than in the *function* $f(x) = \boldsymbol{\phi}(x)^{\mathrm{T}}\boldsymbol{\alpha}$ that they represent. Therefore, if $\widehat{\boldsymbol{\alpha}}$ is an estimated set of coefficients, we only need some practical way to compute $\widehat{\boldsymbol{\alpha}}^{\mathrm{T}}\boldsymbol{\phi}(x)$ for any x. Perhaps somewhat surprisingly, there are cases where we can still do this without having to enumerate all the values in $\widehat{\boldsymbol{\alpha}}$!

The key tool we will use to accomplish this "trick" is the *kernel function* associated with the feature map $\boldsymbol{\phi}(x)$, defined as

$$k(x, x') = \boldsymbol{\phi}(x)^{\mathrm{T}}\boldsymbol{\phi}(x').$$

What is interesting about many kernels is that we can often compute $k(x, x')$ without having to ever compute $\boldsymbol{\phi}(x)$ and $\boldsymbol{\phi}(x')$. Here are a few examples:

- **Polynomials.** For $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^P$, one can check

  $$k(\boldsymbol{x}, \boldsymbol{x}') = (1 + \boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}')^D$$

  corresponds to a $\boldsymbol{\phi}(\boldsymbol{x})$ that includes all the polynomials of degree up to D in P variables.[2] If D and P are moderately large,

[2] Consider, for example, $D = P = 2$, where all polynomials with degree up to 2 in 2 variables (say, x and y) are a linear combination of 1, x, y, x^2, y^2, and xy. We will work out this or another easy example in class.

enumerating all the polynomial basis functions would be quite impractical!

- **Finite (truncated) Fourier series on an interval.** If[3]

$$\boldsymbol{\phi}(x) = \begin{bmatrix} e^{-j2\pi Nx} \\ \vdots \\ e^{-j2\pi x} \\ 1 \\ e^{j2\pi x} \\ \vdots \\ e^{j2\pi Nx} \end{bmatrix},$$

then you can check that

$$k(x, x') = \frac{\sin((2N+1)\pi(x-x'))}{\sin(\pi(x-x'))}$$

(this is the "Dirichlet kernel," which you may recall from ECE 2026). Note that even if N is extremely large, the kernel $k(x, x')$ is relatively easy to compute.

- **Infinite Fourier series.** More generally, if we have the

[3] As soon as complex numbers are involved, the notation becomes more delicate, since we need to use complex conjugates in certain places. We will ignore this detail for simplicity's sake, but keep it in mind if you start working things out. See the Wikipedia page on inner product spaces for more details.

infinite Fourier feature map

$$\boldsymbol{\phi}(x) = \begin{bmatrix} \vdots \\ \lambda_{-2} e^{-j2\pi \cdot 2x} \\ \lambda_{-1} e^{-j2\pi \cdot x} \\ \lambda_0 \\ \lambda_1 e^{j2\pi \cdot x} \\ \lambda_2 e^{j2\pi \cdot 2x} \\ \vdots \end{bmatrix}$$

for some set of coefficients λ_n, the kernel will be

$$k(x, x') = \sum_{n=-\infty}^{\infty} \lambda_n^2 e^{j2\pi n(x-x')},$$

which has a closed-form expression for some interesting choices of the λ_n's.[4]

To see how to use the kernel, recall the Tikhonov regularization formulas (now with $\boldsymbol{\alpha}$ instead of $\boldsymbol{x}$):

$$\widehat{\boldsymbol{\alpha}}_{\text{tik}} = (\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} + \delta \mathbf{I})^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y} = \boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}} + \delta \mathbf{I})^{-1}\boldsymbol{y}.$$

Before, we primarily considered the first formula (involving $\boldsymbol{A}^T\boldsymbol{A}$), but now we will use the second, involving $\boldsymbol{A}\boldsymbol{A}^T$. To see why, note that $\boldsymbol{A}^T\boldsymbol{A}$ is an $N \times N$ matrix, and in the present context, N is extremely large (or infinite). However, $\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}$ is an $M \times M$ matrix; M is simply the number of samples (x_m, y_m) we have, so it should be reasonable in size (or at least not infinite).

Furthermore, the entries of $\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}$ are simply $(\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}})_{ij} = \boldsymbol{\phi}(x_i)^{\mathrm{T}}\boldsymbol{\phi}(x_j) = k(x_i, x_j)$. Therefore, the kernel $k(x, x')$ gives us a practical means to

[4] For those who have taken ECE 2026, note that this is essentially the discrete-time Fourier transform (DTFT) of the signal $x[n] = \lambda_n^2$.

compute

$$\widehat{\boldsymbol{z}} = \begin{bmatrix} \widehat{z}_1 \\ \vdots \\ \widehat{z}_M \end{bmatrix} = (\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}} + \delta \mathbf{I})^{-1}\boldsymbol{y},$$

which is *part* of the Tikhonov regularization formula.

How do we go from here to an estimated regression function $\widehat{f}(x)$? A naïve approach would be to compute $\widehat{\boldsymbol{\alpha}}_{\text{tik}} = \boldsymbol{A}^{\mathrm{T}}\widehat{\boldsymbol{z}}$ and then set $f(x) = \widehat{\boldsymbol{\alpha}}_{\text{tik}}^{\mathrm{T}}\boldsymbol{\phi}(x)$. However, as we have already discussed, this is computationally challenging (or impossible). Instead, note that we can calculate, for any point x and any vector $\boldsymbol{z} \in \mathbb{R}^M$,

$$\begin{aligned}
(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{z})^{\mathrm{T}}\boldsymbol{\phi}(x) &= \boldsymbol{z}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{\phi}(x)) \\
&= \boldsymbol{z}^{\mathrm{T}} \begin{bmatrix} — & \boldsymbol{\phi}(x_1)^{\mathrm{T}} & — \\ — & \boldsymbol{\phi}(x_2)^{\mathrm{T}} & — \\ & \vdots & \\ — & \boldsymbol{\phi}(x_M)^{\mathrm{T}} & — \end{bmatrix} \boldsymbol{\phi}(x) \\
&= \boldsymbol{z}^{\mathrm{T}} \begin{bmatrix} \boldsymbol{\phi}(x_1)^{\mathrm{T}}\boldsymbol{\phi}(x) \\ \boldsymbol{\phi}(x_2)^{\mathrm{T}}\boldsymbol{\phi}(x) \\ \vdots \\ \boldsymbol{\phi}(x_N)^{\mathrm{T}}\boldsymbol{\phi}(x) \end{bmatrix} \\
&= \sum_{m=1}^{M} z_m k(x_m, x).
\end{aligned}$$

where $\boldsymbol{z} = \begin{bmatrix} z_1 & \cdots & z_M \end{bmatrix}^{\mathrm{T}}$. Therefore, we can express our regression function estimate $\widehat{f}(x)$ as

$$\widehat{f}(x) = \widehat{\boldsymbol{\alpha}}_{\text{tik}}^{\mathrm{T}}\boldsymbol{\phi}(x) = (\boldsymbol{A}^{\mathrm{T}}\widehat{\boldsymbol{z}})^{\mathrm{T}}\boldsymbol{\phi}(x) = \sum_{m=1}^{M} \widehat{z}_m k(x_m, x).$$

Thus we can calculate the Tikhonov regularized regression estimate $\widehat{f}$ using only the data $((x_m, y_m), m = 1, \ldots, M)$ and the kernel function

$k(x, x')$! Furthermore, computationally, the most difficult part is to invert an $M \times M$ matrix.

An extended Fourier series example

Consider the Fourier feature map

$$\boldsymbol{\phi}(x) = \begin{bmatrix} \vdots \\ \lambda_{-2} e^{-j2\pi \cdot 2x} \\ \lambda_{-1} e^{-j2\pi \cdot x} \\ \lambda_0 \\ \lambda_1 e^{j2\pi \cdot x} \\ \lambda_2 e^{j2\pi \cdot 2x} \\ \vdots \end{bmatrix},$$

where

$$\lambda_n = a^{-|n|/2}, \ n = \ldots, -2, -1, 0, 1, 2, \ldots$$

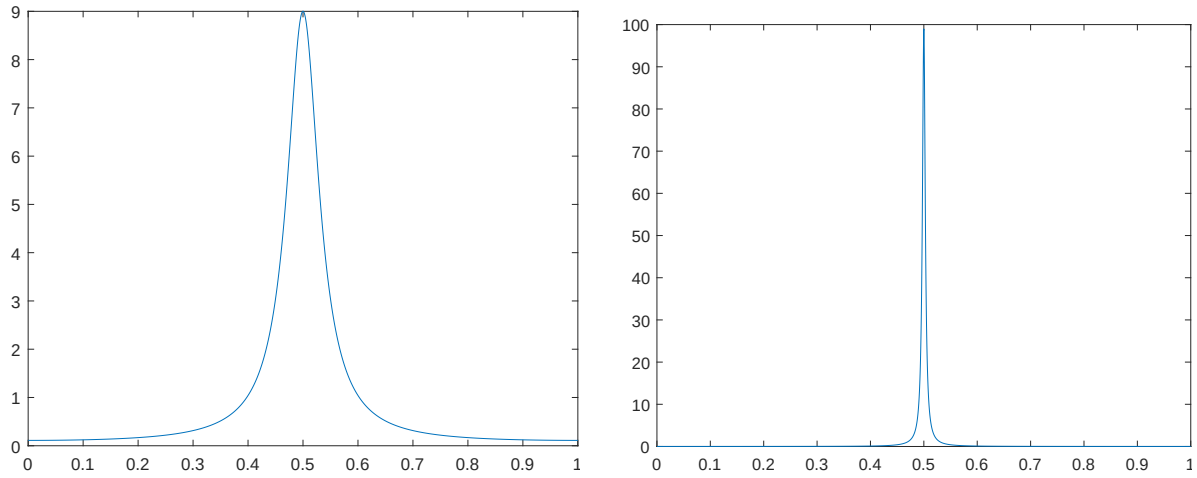
for some real number $a \in (0, 1)$.

Given samples $(x_m, y_m), m = 1, \ldots, M$ with $x_m \in [0, 1]$, we want to find a function $\widehat{f}(x) = \widehat{\boldsymbol{\alpha}}^{\mathrm{T}} \boldsymbol{\phi}(x)$ that fits the data.

What does Tikhonov regularization do in this setting? Note that for any $\boldsymbol{\alpha}$, $(\boldsymbol{A}\boldsymbol{\alpha})_m = \boldsymbol{\alpha}^{\mathrm{T}} \boldsymbol{\phi}(x_m)$. Therefore, the Tikhonov regularization optimization program is

$$\underset{\boldsymbol{\alpha}}{\text{minimize}} \ \sum_{m=1}^{M} (y_m - \boldsymbol{\alpha}^{\mathrm{T}} \boldsymbol{\phi}(x_m))^2 + \delta \sum_{n=-\infty}^{\infty} \alpha_n^2.$$

The regularization term penalizes all the α_n equally, but because the coefficients λ_n decay exponentially, the regularization penalizes sinusoidal components of $\boldsymbol{\alpha}^{\mathrm{T}} \boldsymbol{\phi}(x)$ more and more at higher (positive



(a) $a = 0.8$ (b) $a = 0.98$

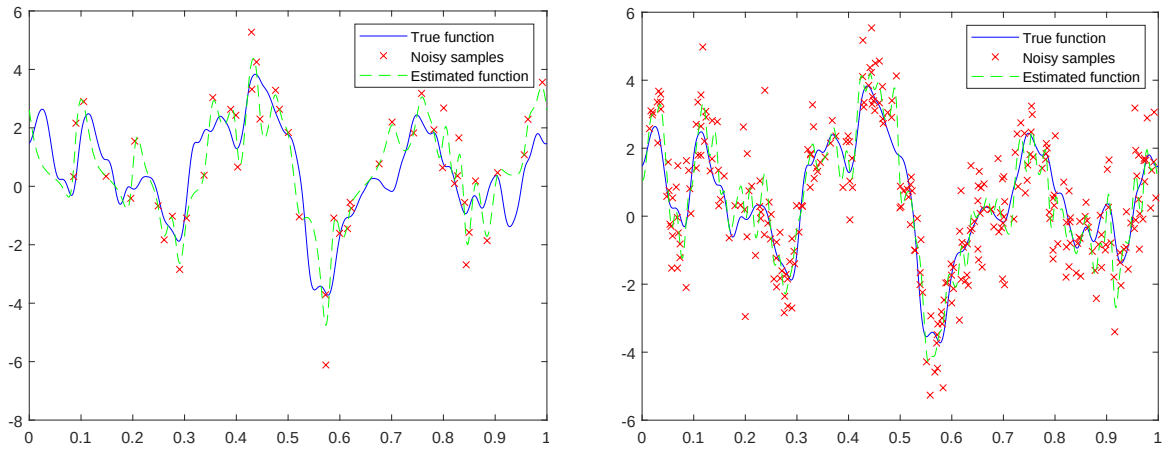
Figure 1: $k(x, 0.5)$ for the Fourier series kernel with $\lambda_n = a^{-|n|/2}$ for different values of a.

and negative) frequencies. Thus Tikhonov regularization promotes functions f that are "smooth" in the sense that most of their energy is in lower frequencies. The degree of penalization depends on the parameter a, as we will see more clearly in a moment.

To see how to compute the regression estimate $\widehat{f}$, we first calculate the kernel. If we treat λ_n^2 as a discrete-time signal and apply a DTFT, we can show that the kernel function is

$$k(x, x') = \sum_{n=-\infty}^{\infty} \lambda_n^2 e^{j2\pi n(x-x')} = \frac{1 - a^2}{1 - 2a\cos(2\pi(x - x')) + a^2}.$$

Figure 1 show the kernel function for two values of a. Note that for a closer to 1, the kernel is very "spiky" (and therefore able to represent rougher functions), while for smaller a, the kernel is much more gentle (and therefore can only represent smoother functions). This makes sense given the fact that decreasing a penalizes large frequencies more.



(a) $M = 50$ (b) $M = 300$

Figure 2: Regression example with different sample sizes M. $a = 0.9$, and $\delta = 1$. The noise has a Gaussian distribution with zero mean and variance $\sigma^2 = 1$.

We can then do regression with the kernel $k(x, x')$. We write the "kernel matrix" as

$$\boldsymbol{K} = \boldsymbol{A}\boldsymbol{A}^{\mathrm{T}} = \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & \cdots & k(x_1, x_M) \\ k(x_2, x_1) & k(x_2, x_2) & \cdots & k(x_2, x_M) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_M, x_1) & k(x_M, x_2) & \cdots & k(x_M, x_M) \end{bmatrix}$$

Then, recall that

$$\widehat{f}(x) = \sum_{m=1}^{M} \widehat{z}_m k(x, x_m),$$

where

$$\widehat{\boldsymbol{z}} = \begin{bmatrix} \widehat{z}_1 \\ \vdots \\ \widehat{z}_M \end{bmatrix} = (\boldsymbol{K} + \delta \mathbf{I})^{-1} \boldsymbol{y}.$$

Figure 2 show an example with two different sample sizes. Note that the larger sample size leads to a more accurate estimate.

Beyond explicit feature maps

Kernels become most interesting when, unlike in the example of the previous section, we *don't have* a handy expression for the "feature map" that generates the kernel.[5]

We showed how to get a kernel $k(x, x')$ from a feature map $\boldsymbol{\phi}(x)$. Since we never actually use $\boldsymbol{\phi}(x)$ in our formulas, could we use a kernel $k(x, x')$ without having to think about what feature map generates it?

It turns out that kernels generated by feature maps are special in the following sense: for *any* positive integer M and set of points $x_1, \ldots, x_M$, the kernel matrix $\boldsymbol{K}$ on those points is positive semidefinite, that is,

$$\boldsymbol{z}^{\mathrm{T}} \boldsymbol{K} \boldsymbol{z} \geq 0 \text{ for all } \boldsymbol{z} \in \mathbb{R}^M.$$

You can verify this by recalling that $\boldsymbol{K} = \boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}$, so

$$\boldsymbol{z}^{\mathrm{T}} \boldsymbol{K} \boldsymbol{z} = \boldsymbol{z}^{\mathrm{T}} \boldsymbol{A}\boldsymbol{A}^{\mathrm{T}} \boldsymbol{z} = \|\boldsymbol{A}^{\mathrm{T}} \boldsymbol{z}\|_2^2 \geq 0.$$

We call any symmetric function $k(x, x')$ that satisfies this property a *positive semidefinite* (PSD) *kernel*. It turns out that *any* PSD kernel has a corresponding feature map $\boldsymbol{\phi}(x)$ that satisfies $k(x, x') = \boldsymbol{\phi}(x)^{\mathrm{T}} \boldsymbol{\phi}(x')$.

The best choice of kernel depends on what kind of function one is expecting to recover. One common choice is the Gaussian function

$$k(x, x') = e^{-(x-x')^2/r^2},$$

where $r > 0$ is a scaling parameter. This works best with extremely smooth functions, since the Gaussian kernel is very smooth.

[5] In fact, if we know the features exactly, we are often still better off computationally if we just use a truncated feature map.

Another choice (sometimes called the exponential kernel) is $k(x, x') = e^{-|x-x'|/r}$. This kernel is not even differentiable, so it can recover very "rough" functions.

In both cases, the feature maps are infinite-dimensional and complicated to describe.