STATS205: Introduction to Nonparametric Statistics

Lecturer: Tengyu Ma
Scribe: Cam Burton and Matt Johnson

Lecture # 5
April 30th, 2021

5.1 Review and overview

In the previous lecture, we finished our discussion on natural cubic splines with different interpretations. In particular, we discussed interpreting splines as linear smoothers and how to use kernel estimation to estimate splines. We then began our discussion on nonparametric methods in higher dimensions, exploring the k-nearest neighbor algorithm and the kernel method.

In this lecture we will finish our discussion on the kernel method and begin talking about neural networks. We will explore their connection to the kernel method and their practical implementation.

5.2 More about kernel methods

5.2.1 Recap

Let us quickly review the kernel method from the last lecture. The basic principle of the kernel method is that given a set of data points

$$\left\{(x^{(1)}, y^{(1)}), \cdots, (x^{(n)}, y^{(n)})\right\}, \quad x^{(i)} \in \mathbb{R}^d, y^{(i)} \in \mathbb{R},$$

we look for a suitable feature map ϕ such that

$$\phi : x \mapsto \phi(x) \in \mathbb{R}^m.$$

Our interpretation of this feature map is that it is transforming the feature pairs in our dataset. If we run a linear regression or a logistic regression on the transformed dataset $(\phi(x^{(i)}), y^{(i)})$, then the algorithm only depends on the inner product (i.e. we don't need to know $\phi(x)$ or $\phi(z)$ explicitly). We only need to compute $\langle \phi(x), \phi(z) \rangle$. This is called the kernel function

$$K(x, z) := \langle \phi(x), \phi(z) \rangle. \tag{5.1}$$

If we can compute the kernel function directly, then we don't need to pay the computational overhead of computing the ϕ function/map explicitly. When the number of features is large, computing the feature map explicitly can be quite costly.

5.2.2 Another approach to kernel methods

An alternate way of understanding the kernel method is to view each feature as a function of x, that is

$$\phi(x)_k : x \to \mathbb{R},$$

where

$$\phi(x) = \begin{bmatrix} \phi(x)_1 \\ \vdots \\ \phi(x)_m \end{bmatrix}.$$

An example could be the second degree polynomial kernel such that $\phi(x)_{(ij)} = x_i x_j$. We can also view the linear prediction function of our features as a linear combination of these functions

$$\theta^T \phi(x) = \sum_{i=1}^{m} \theta_i \phi(x)_i \in \operatorname{span}\{\phi(x)_1, ..., \phi(x)_m\}.$$

The kernel method can be thought of as looking for a function in a linear span of functions.

5.2.3 Connection to splines

A cubic spline is a function in the span of a family of basis of cubic splines, that is our model $r(x)$ satisfies

$$r(x) \in \operatorname{span}\{h_1(x), ..., h_{n+4}(x)\}.$$

Equivalently, we can write

$$r(x) = \sum_{i=1}^{n+4} \beta_i h_i(x) = \beta^T \phi(x),$$

where $\phi(x)_i = h_i(x), i = 1, \cdots, n+4$, and thus

$$\phi : x \mapsto \phi(x) = \begin{bmatrix} h_1(x) \\ \vdots \\ h_{n+4}(x) \end{bmatrix}$$

is a feature map. Consequently, in our connection between kernels and splines, we can write out the kernel function for cubic splines as

$$K(x, z) = \langle \phi(x), \phi(z) \rangle = \sum_{i=1}^{n+4} h_i(x) h_i(z).$$

Empirically, our main design choice centers around our choosing a basis $h_i(x)$ such that $K(x, z)$ is efficiently computable. Previous bases we have used for splines have been good mathematically but are not necessarily the best choice when thinking about computability.

The kernel method with feature map ϕ is equivalent to a cubic spline $\hat{r}$ with no regularization, or a ridgeless kernel regression since

$$\operatorname*{argmin}_{\beta} \sum_{i=1}^{n} (y^{(i)} - \beta^T \phi(x^{(i)}))^2 \Leftrightarrow \operatorname*{argmin}_{\hat{r}} \sum_{i=1}^{n} (y^{(i)} - \hat{r}(x^{(i)}))^2.$$

However, this is underspecified because the number of parameters $(n+4) >$ number of data points n. Therefore, we look for the minimum norm solution or a regularization. An example of a regularized solution is the kernel ridge regression

$$\min_{\beta} \frac{1}{2} \sum_{i=1}^{n} (y^{(i)} - \beta^T \phi(x^{(i)}))^2 + \frac{\lambda}{2} \|\beta\|_2^2,$$

whose minimizer takes a simple form (cf. Homework 3),

$$\hat{\beta} = \Phi^\top (\Phi \Phi^\top + \lambda I)^{-1} y,$$

where

$$\Phi = \begin{bmatrix} \phi(x^{(1)})^\top \\ \vdots \\ \phi(x^{(n)})^\top \end{bmatrix} \in \mathbb{R}^{n\times m}.$$

To connect with splines, we see in in our previous lecture that in natural cubic splines we're using a similar but different regularizer $\beta^\top \Omega \beta$.

5.2.4 Connection to nearest neighbor methods

Another application of the kernel method is to perform a nearest neighbor strategy in feature space. We run nearest neighbor on $\{(\phi(x^{(1)}), y^{(1)}), ..., (\phi(x^{(n)}), y^{(n)})\}$, and the ℓ_2 squared distance metric is then

$$\begin{aligned} d(x,z) &= \|\phi(x) - \phi(z)\|_2^2 \\ &= \langle \phi(x)^T - \phi(z)^T, \phi(x) - \phi(z) \rangle \\ &= \phi(x)^T\phi(x) - 2\phi(x)^T\phi(z) + \phi(z)^T\phi(z) \\ &= K(x,x) - 2K(x,z) + K(z,z) \end{aligned}$$

We see again that we don't need to compute the features ϕ explicitly.

5.3 Neural networks

5.3.1 A glimpse into deep learning theory

While Neural Networks are not typically studied in most classic statistics classes, in recent years they have revolutionized the field of machine learning and thus have become an increasingly interesting topic in the field of statistics. The result we will show in this lecture is that a one-dimensional neural network is a fully non-parametric method which is somewhat similar to what we have already discussed in cubic splines. We will primarily discuss the following two things.

1. We will use a neural net to represent features and then find those features, this allows for more dynamic and better features than those computed in the kernel method.

2. We will also show that for one dimensional two layer wide neural network, it is equivalent to a linear spline.

5.3.2 Fully-connected two layer neural networks

A neural network can be thought of as a method to learn the features ϕ in a non-parametric model. If we think about neural networks in the specific case where the input is only 1-dimensional, and we have two layers, then we can see that it is really a type of linear spline. To begin, we introduce some basic notations.

Definition 5.1 (Transformation of fully-connected neural network in each layer)**.** We denote by the input of i-th layer of a neural network by $h_{i-1} \in \mathbb{R}^d$ and its output by $h_i \in \mathbb{R}^m$. The weighted matrix parameters are denoted by $W \in \mathbb{R}^{m\times d}$. Let σ be the activation function $\mathbb{R} \to \mathbb{R}$. Examples of activation functions include

$$\begin{aligned}
\mathrm{ReLU}(x) &:= \max\{x, 0\},\\
\mathrm{Sigmoid}(x) &:= \frac{1}{1+e^{-x}},\\
\mathrm{Softplus}(x) &:= \log\left(\frac{1}{1+e^{x}}\right).
\end{aligned}$$

Then the output vector can be written as $h_i = \sigma(Wh_{i-1})$, where σ here is understood to be applied elementwise.

For a fully-connected two layer neural networks, we thus can write

$$\hat{y} = a^\top \sigma(Wx),$$

where $x := h_0$ is the input of the network and $a \in \mathbb{R}^m$. If we view $\sigma(Wx)$ as a feature map $\phi(x)$ which depends on W (hence we might more accurately write this feature map as $\phi_W(x)$), then this is similar to a kernel method. If we fix W, then this is exactly a the kernel method with $K(x,z) = \langle \phi_W(x), \phi_W(z)\rangle$. In neural networks, the difference is that we train both a and W. If we don't train W, then we essentially have a kernel method.

5.3.3 Deep neural networks

With the above notations, we can formally define a fully-connected deep neural network with r layers, parameters $W_1, \cdots, W_r, a$ and input vector $x := h_0$ as

$$\begin{aligned}
\text{First layer:}\quad & h_1 = \sigma(W_1, x)\\
\text{Second layer:}\quad & h_2 = \sigma(W_2, h_1)\\
& \vdots\\
\text{Output:}\quad & \hat{y} = a^T h_r
\end{aligned}$$

Often times, h_r is called "the features". $h_r = \sigma(W_r\sigma(W_{r-1}\dots)) \to \phi_{W_1\dots W_r}(x)$ where $\phi_{W_1\dots W_r}(x)$ is referred to as the feature extractor or the feature map. The key difference is that $\phi_{W_1\dots W_r}(x)$ is learned. More broadly, any sequence of parameterized computations is called a neural network. For example, the residual neural network is

$$\begin{aligned}
\text{First layer:}\quad & h_1 = \sigma(W_1, x)\\
\text{Second layer:}\quad & h_2 = h_1 + \sigma(W_2, h_1)\\
& \vdots\\
r\text{-th layer:}\quad & h_r = h_{r-1} + \sigma(W_r, h_{r-1})\\
\text{Output:}\quad & \hat{y} = a^T h_r
\end{aligned}$$

5.3.4 Equivalence to linear splines

We will work with infinitely wide neural networks to make the connection to linear splines. We don't really need an infinite width, but we do need a really large width. First, we introduce some notations. We call an input $x \in \mathbb{R}$ and its associated output $y \in \mathbb{R}$. We will call our model $h_\theta(x) = \sum_{i=1}^{m} a_i[w_i x + b_i]_+ + c$ where $a_i \in \mathbb{R}$, $w_i \in \mathbb{R}$, $x \in \mathbb{R}$, $b_i \in \mathbb{R}$, $c \in \mathbb{R}$. Our activation function in this neural network is simply the $\text{ReLU}(x) = \max\{t, 0\}$ function which we denoted by $[\dots]_+$. And as an aside

$$\begin{aligned}
&a^T\sigma(Wx) = \sum_{i=1}^{m} a_i(\sigma(Wx))_i = \sum_{i=1}^{m} a_i\sigma(Wx)_i),\\
&W = \begin{bmatrix} w_1^T \\ \vdots \\ w_m^T \end{bmatrix} \to Wx = \begin{bmatrix} w_1^T x \\ \vdots \\ w_m^T x \end{bmatrix},\\
&(Wx)_i = w_i^T x,\\
&a^T\sigma(Wx) = \sum_{i=1}^{m} a_i\sigma(w_i^T x).
\end{aligned}$$

We will denote our parameters by $\theta = (m, a, w, b, c)$ where $m \in \mathbb{N}$, $a \in \mathbb{R}^m$, $w \in \mathbb{R}^m$, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}$.

Our regularizer will be the l_2 norm of the weights parameter:

$$C(\theta) = \frac{1}{2}\left(\|a\|_2^2 + \|w\|_2^2\right) = \frac{1}{2}\sum_{i=1}^{m}(a_i^2 + w_i^2) \tag{5.2}$$

And we now define our regularized training objective:

$$\inf_\theta [L(h_\theta) + \lambda C(\theta)] \tag{5.3}$$

Where $L(h_\theta)$ can be any loss function that is continuous in θ, and $C(\theta)$ is our regularizer. An example of a loss function is

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - h_\theta(x^{(i)}))^2. \tag{5.4}$$

5.3.5 Simplification: m goes to infinity

With some abuse of notation, we will work with the following neural network

$$\begin{aligned}
&h_\theta(x) = \sum_{i=1}^{\infty} a_i[w_i + b_i]_+,\\
&a = (a_i, \dots, a_k, \dots) \in \mathbb{R}^\infty, w = (w_i, \dots, w_k, \dots) \in \mathbb{R}^\infty,\\
&b = (b_i, \dots) \in \mathbb{R}^\infty,\\
&C(\theta) = \frac{1}{2}\sum_{i=1}^{\infty} a_i^2 + \sum_{i=1}^{\infty} w_i^2.
\end{aligned}$$

Theorem 5.2. *Define the nonparametric complexity measure*

$$\bar{R}(f) = \max\left\{\int_{-\infty}^{+\infty} |f^{''}(x)|dx, |f^{'}(-\infty) + f^{'}(+\infty)|\right\}. \tag{5.5}$$

The first term is related to the continuity of the function, and the second term pertains to the slope. Recall that for cubic splines, the penalization was $\int r^{''}(x)^2dx$. For a nonparametric penalized regression, our goal is to find the minimizer to

$$\inf_f L(f) + \lambda\bar{R}(f). \tag{5.6}$$

For a parameterized neural network, we are trying to find the minimizer to

$$\inf_\theta L(h_\theta) + \lambda C(\theta). \tag{5.7}$$

We claim that these methods are doing the same thing. Specifically we claim that

$$\inf_f L(f) + \lambda\bar{R}(f) = \inf_\theta L(h_\theta) + \lambda C(\theta),$$

and

$$f^* = h^*_\theta,$$

where f^* and θ^* are the minimizers of the above two problems respectively.

In other words, on the one hand we have a non-parametric approach, i.e. a penalized regression with a complexity measure $\bar{R}(f)$. On the other hand we have a parameterized regression which comes from neural networks. We claim that these are doing the exact same thing.

How do we interpret this? What does minimizing $L(f) + \lambda\bar{R}(f)$ really do? First, let's consider

$$\text{minimize } \bar{R}(f) \text{ s.t } L(f) = \sum_{i=1}^{n}(y^{(i)} - f(x^{(i)}))^2 = 0.$$

As this corresponds to the case where $\lambda \to 0$. The above is minimized when f is a linear spline that fits the data exactly and $L(f) = 0$.

From equation (5.5), we see that $\bar{R}(f)$ consists of two terms. We know that $f^{''}(x) = \infty$ at the data points (since this is where the slope instantaneously changes from one linear line to another), and $f^{''}(x) = 0$ otherwise. Hence, we can model $f^{''}(x)$ by the dirac delta function $\{\delta(t)|t = \text{datapoint}\}$. For dirac delta functions, we know that $\int_{-\infty}^{+\infty}\delta(t)dt = 1$, hence $\int |f^{''}(x)|dx$ in equation (5.5) is actually quite small. We won't go into the formality of proving this, but take it as true that minimizing $\bar{R}$ gives we a linear spline since the penalization from the second order derivatives is actually quite small.

To represent a linear spline with n knots, we only need $n+1$ pieces. We can therefore represent a linear spline with a neural net of at most $n+1$ terms. Analogously, in penalized regression we started with all possible solutions $r(x)$, but after we realized that the solution has a structure like a cubic spline, we then reduced our infinitely large solution space to an $n+4$ dimensional space ($n+4$ neurons/width of the neural net); this simplification makes the optimization of the problem a lot easier.

5.3.6 Outline of the proof

The proof follows mainly from the following two steps.

- Step 1: Show $\exists\ \bar{R}(f)$ such that $\min L(f) + \lambda \bar{R}(f) = \min L(\theta) + \lambda C(\theta)$;
- Step 2: Derive the formula for $\bar{R}(f)$.

In this lecture we show the first step. We begin by looking for a representation of $f(x)$ by a neural network with minimum complexity.

$$\bar{R}(f) \overset{\Delta}{=} \min C(\theta) \text{ s.t. } f(x) = h_\theta(x) \tag{5.8}$$

Why do we know that such a neural network exists? For any piecewise linear function with a finite number of pieces, we know that there exists a $h_\theta(x)$ that represents $f(x)$, since neural networks are piecewise linear for a finite number of neurons. A uniformly continuous function $f(x)$ can be approximated by a 2-layer neural network with finite width, and it can be exactly represented by a two-layer neural network with infinite width. This is done by taking finer and finer approximations of our function, and then taking the limit as the number of approximations (width of our neural network) $\to \infty$.

Hence, we wish to prove that

$$\min L(f) + \lambda \bar{R}(f) = \min L(\theta) + \lambda C(\theta),$$

where $\bar{R}(f) = \min C(\theta)$ s.t. $f = h_\theta$. Let θ^* be the minimizer of the $\min L(\theta) + \lambda C(\theta)$, thus

$$L(h_{\theta^*}) + \lambda C(\theta^*) = \min L(\theta) + \lambda C(\theta).$$

Take $f = h_{\theta^*}$, then we have $L(f) = L(h_{\theta^*})$ and $\bar{R}(f) = \min C(\theta) = C(\theta^*);$ where $h_\theta = f$. Combining these statements implies that $L(f) + \lambda \bar{R}(f) \leq L(h_{\theta^*}) + \lambda C(\theta^*) = \min L(\theta) + \lambda C(\theta)$, which suggests

$$\min L(f) + \lambda \bar{R}(f) \leq \min L(\theta) + \lambda C(\theta). \tag{5.9}$$

On the other direction, let f^* be the minimizer of $\min L(f) + \lambda \bar{R}(f)$. By the argument above, we can construct θ such that $\min L(f) + \lambda \bar{R}(f) \geq L(h_\theta) + \lambda C(\theta) \geq \min L(\theta) + \lambda C(\theta)$. Take θ to be the minimizer of $\min C(\theta) = \bar{R}(f^*)$ s.t. $f^* = h_\theta$. This is the minimum complexity network that can represent f. This means that $C(\theta) = \bar{R}(f)$ and

$$\begin{aligned}
\min L(f) + \lambda \bar{R}(f) &= L(f^*) + \lambda \bar{R}(f^*) \geq L(f^*) + \lambda C(\theta) \\
&= L(h_\theta) + \lambda C(\theta) \\
&\geq \min L(h_\theta) + \lambda C(\theta) \\
&= \min L(\theta) + \lambda C(\theta).
\end{aligned} \tag{5.10}$$

Taken collectively, we conclude that $\min L(f) + \lambda \bar{R}(f) = \min L(\theta) + \lambda C(\theta)$. The second step is left for the next lecture.