CHAPTER

21 LEARNING MODELS

CHAPTER CONTENTS

Through Part 4, let us consider the *supervised learning* problem of approximating a function $y = f(\boldsymbol{x})$ from its input-output paired training samples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ (see Section 1.2.1). The input $\boldsymbol{x}$ is assumed to be a real-valued d-dimensional vector, and the output y is a real scalar in the case of *regression* and a categorical value $\{1, \ldots, c\}$ in the case of *classification*, where c denotes the number of classes. In this chapter, various *models* for supervised learning are introduced.

21.1 LINEAR-IN-PARAMETER MODEL

For simplicity, let us begin with a one-dimensional learning target function f. The simplest model for approximating f would be the *linear-in-input model* $\theta \times x$. Here, θ denotes a scalar parameter and the target function is approximated by learning the parameter θ. Although the linear-in-input model is mathematically easy to handle, it can only approximate a linear function (i.e., a straight line) and thus its expression power is limited (Fig. 21.1).

The *linear-in-parameter model* is an extension of the linear-in-input model that allows approximation of nonlinear functions:

$$f_{\boldsymbol{\theta}}(x) = \sum_{j=1}^{b} \theta_j \phi_j(x),$$

where $\phi_j(x)$ and θ_j are a *basis function* and its parameter, respectively, and b denotes the number of basis functions. The linear-in-parameter model may be compactly expressed as

$$f_{\boldsymbol{\theta}}(x) = \boldsymbol{\theta}^\top \boldsymbol{\phi}(x),$$

where

$$\boldsymbol{\phi}(x) = (\phi_1(x), \ldots, \phi_b(x))^\top,$$

An Introduction to Statistical Machine Learning. DOI: 10.1016/B978-0-12-802121-7.00032-7

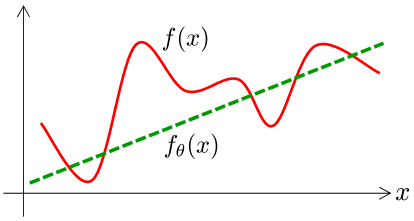


FIGURE 21.1

Linear-in-input model cannot approximate nonlinear functions.

$$\boldsymbol{\theta} = (\theta_1, \ldots, \theta_b)^\top,$$

and $^\top$ denotes the transpose. The linear-in-parameter is still linear in terms of parameters $\boldsymbol{\theta}$, but it can express nonlinear functions, e.g., by using *polynomial functions*

$$\boldsymbol{\phi}(x) = (1, x, x^2, \ldots, x^{b-1})^\top,$$

or *sinusoidal functions* for $b = 2m + 1$

$$\boldsymbol{\phi}(x) = (1, \sin x, \cos x, \sin 2x, \cos 2x, \ldots, \sin mx, \cos mx)^\top,$$

as basis functions.

The linear-in-parameter can be naturally extended to d-dimensional input vector $\boldsymbol{x} = (x^{(1)}, \ldots, x^{(d)})^\top$ as

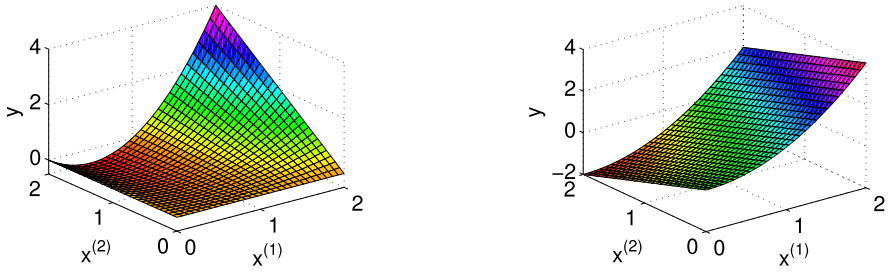
$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=1}^{b} \theta_j \phi_j(\boldsymbol{x}) = \boldsymbol{\theta}^\top \boldsymbol{\phi}(\boldsymbol{x}). \tag{21.1}$$

Below, methods for constructing multidimensional basis functions from one-dimensional basis functions are discussed.

The *multiplicative model* expresses multidimensional basis functions by the product of one-dimensional basis functions as

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j_1=1}^{b'} \cdots \sum_{j_d=1}^{b'} \theta_{j_1,\ldots,j_d} \phi_{j_1}(x^{(1)}) \cdots \phi_{j_d}(x^{(d)}),$$

where b' denotes the number of parameters in each dimension. Since all possible combinations of one-dimensional basis functions are considered in the multiplicative model, it can express complicated functions, as illustrated in Fig. 21.2(a). However,



(a) Multiplicative model (b) Additive model

FIGURE 21.2

Multidimensional basis functions. The multiplicative model is expressive, but the number of parameters grows exponentially in input dimensionality. On the other hand, in the additive model, the number of parameters grows only linearly in input dimensionality, but its expression power is limited.

the total number of parameters is $(b')^d$, which grows exponentially as input dimensionality d increases. For example, when $b' = 10$ and $d = 100$, the total number of parameters is given by

$$10^{100} = 1\underbrace{000\cdots000}_{100},$$

which is an astronomical number having 100 zeros after the first one and cannot be handled in computers. Such exponential growth in input dimensionality is often referred to as the *curse of dimensionality*.

The *additive model* expresses multidimensional basis functions by the sum of one-dimensional basis functions as

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{k=1}^{d} \sum_{j=1}^{b'} \theta_{k,j} \phi_j(x^{(k)}).$$

In the additive model, the total number of parameters is $b'd$, which grows only linearly as input dimensionality d increases. For example, when $b' = 10$ and $d = 100$, the total number of parameters is $10 \times 100 = 1000$ and thus it can be easily handled by standard computers. However, since only the sum of one-dimensional basis functions is considered in the additive model, it cannot express complicated functions, as illustrated in Fig. 21.2(b).

21.2 KERNEL MODEL

In the linear-in-parameter model introduced above, basis functions are fixed to, e.g., polynomial functions or sinusoidal functions without regard to training samples

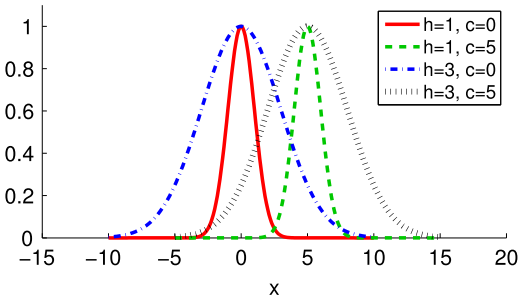


FIGURE 21.3

Gaussian kernel with bandwidth h and center c.

$\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$. In this section, the *kernel model* is introduced, which uses training input samples $\{\boldsymbol{x}_i\}_{i=1}^n$ for basis function design.

Let us consider a bivariate function called the *kernel function* $K(\cdot,\cdot)$. The kernel model is defined as the linear combination of $\{K(\boldsymbol{x},\boldsymbol{x}_j)\}_{j=1}^n$:

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=1}^{n} \theta_j K(\boldsymbol{x},\boldsymbol{x}_j). \tag{21.2}$$

As a kernel function, the *Gaussian kernel* would be the most popular choice:

$$K(\boldsymbol{x},\boldsymbol{c}) = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{c}\|^2}{2h^2}\right),$$

where $\|\cdot\|$ denotes the ℓ_2-norm:

$$\|\boldsymbol{x}\| = \sqrt{\boldsymbol{x}^\top \boldsymbol{x}}.$$

h and $\boldsymbol{c}$ are, respectively, called the *Gaussian bandwidth* and the *Gaussian center* (see Fig. 21.3).

In the Gaussian kernel model, Gaussian functions are located at training input samples $\{\boldsymbol{x}_i\}_{i=1}^n$ and their height $\{\theta_i\}_{i=1}^n$ is learned (Fig. 21.4). Since the Gaussian function is *local* around its center, the Gaussian kernel model can approximate the learning target function only in the vicinity of training input samples (Fig. 21.5). This is quite different from the multiplicative model which tries to approximate the learning target function over the entire input space.

In the kernel model, the number of parameters is given by the number of training samples, n, which is independent of the dimensionality d of the input variable $\boldsymbol{x}$. Thus, even when d is large, the kernel model can be easily handled in standard computers, as long as n is not too large. Even if n is very large, only a subset $\{\boldsymbol{c}_j\}_{j=1}^b$

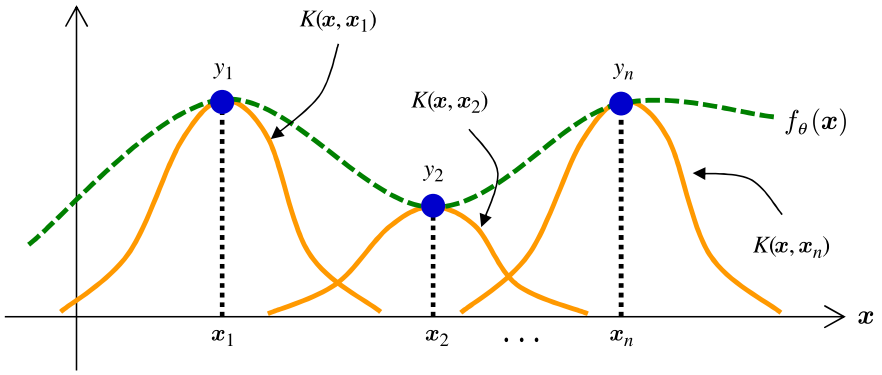


FIGURE 21.4

One-dimensional Gaussian kernel model. Gaussian functions are located at training input samples $\{\boldsymbol{x}_i\}_{i=1}^n$ and their height $\{\theta_i\}_{i=1}^n$ is learned.

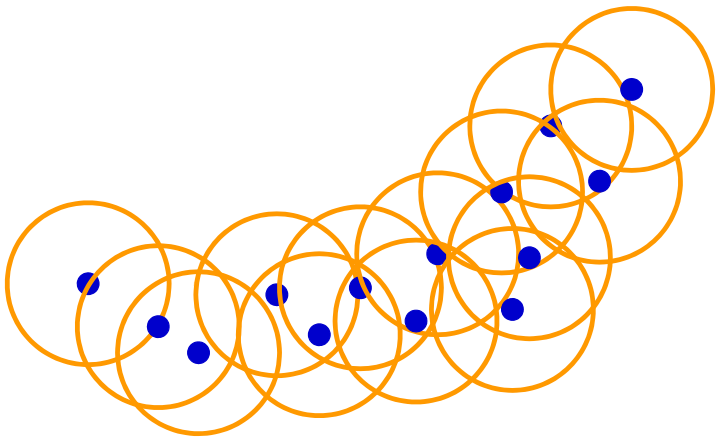


FIGURE 21.5

Two-dimensional Gaussian kernel model. The curse of dimensionality is mitigated by only approximating the learning target function in the vicinity of training input samples.

of training input samples $\{\boldsymbol{x}_i\}_{i=1}^n$ may be used as kernel centers for reducing the computation costs:

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=1}^{b} \theta_j K(\boldsymbol{x}, \boldsymbol{c}_j).$$

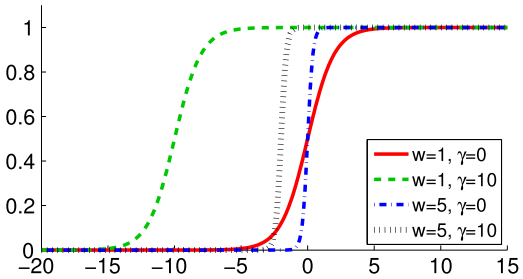


FIGURE 21.6

Sigmoidal function.

Since kernel model (21.2) is linear in terms of the parameter $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n)^\top$, it can also be regarded as a linear-in-parameter model. However, basis functions $\{K(\boldsymbol{x}, \boldsymbol{x}_j)\}_{j=1}^n$ depend on training input samples $\{\boldsymbol{x}_i\}_{i=1}^n$ and the number of basis function grows as the number of training samples increases. For this reason, in statistics, the kernel model is categorized as a *nonparametric model* and is clearly differentiated from linear-in-parameter models which are parametric. However, within the scope of Part 4, the kernel model may be treated almost in the same way as the linear-in-parameter models.

Another important advantage of the kernel model is that it can be easily extended to *nonvectorial* $\boldsymbol{x}$ such as a sequence (with different length) and a tree (with different depth) [46]. More specifically, since input $\boldsymbol{x}$ only appears in the kernel function $K(\boldsymbol{x}, \boldsymbol{x}')$ in kernel model (21.2), the expression of $\boldsymbol{x}$ itself does not matter as long as the kernel function $K(\boldsymbol{x}, \boldsymbol{x}')$ for two input objects $\boldsymbol{x}$ and $\boldsymbol{x}'$ is defined.

Machine learning with kernel functions is called the *kernel method* and has been studied extensively [89].

21.3 HIERARCHICAL MODEL

A model that is nonlinear in terms of parameters is referred to as a *nonlinear model*. Among nonlinear models, a three-layer *hierarchical model* is a popular choice:

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=1}^{b} \alpha_j \phi(\boldsymbol{x}; \boldsymbol{\beta}_j),$$

where $\phi(\boldsymbol{x}; \boldsymbol{\beta})$ is a basis function parameterized by $\boldsymbol{\beta}$. A hierarchical model is linear in terms of parameter $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_b)^\top$, as in Eq. (21.1). However, in terms of the basis parameters $\{\boldsymbol{\beta}_j\}_{j=1}^b$, the hierarchical model is nonlinear.

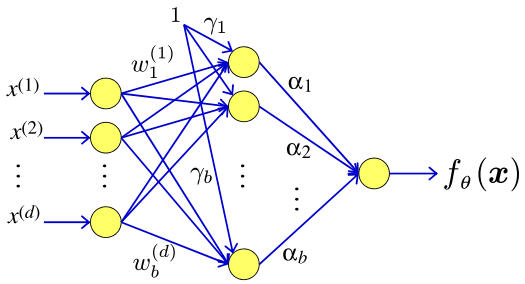


FIGURE 21.7

Hierarchical model as a three-layered network.

As a basis function, the *sigmoidal function* (see Fig. 21.6),

$$\phi(\boldsymbol{x};\boldsymbol{\beta}) = \frac{1}{1+\exp(-\boldsymbol{x}^\top \boldsymbol{w} - \gamma)}, \quad \boldsymbol{\beta} = (\boldsymbol{w}^\top, \gamma)^\top,$$

and the *Gaussian function* (see Fig. 21.3),

$$\phi(\boldsymbol{x};\boldsymbol{\beta}) = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{c}\|^2}{2h^2}\right), \quad \boldsymbol{\beta} = (\boldsymbol{c}^\top, h)^\top,$$

are standard choices. The sigmoidal function is motivated by the activity of neurons in human's brain. For this reason, a hierarchical model is also called a *neural network* and a basis function is called an *activation function*. To reduce the computation cost, the *rectified linear function*,

$$\phi(\boldsymbol{x};\boldsymbol{\beta}) = \max(\boldsymbol{x}^\top \boldsymbol{w} + \gamma, 0), \quad \boldsymbol{\beta} = (\boldsymbol{w}^\top, \gamma)^\top,$$

is also popularly used, although it is not differentiable at zero.

On the other hand, the Gaussian function is essentially the same as the Gaussian kernel introduced in Section 21.2, but the Gaussian bandwidth and center are also learned in the hierarchical model. For this reason, a neural network model with the Gaussian activation function is expected to be more expressive than the Gaussian kernel model.

A hierarchical model can be expressed by a three-layered network, as illustrated in Fig. 21.7. A notable characteristic of the neural network model is that a mapping between a parameter and a function is not necessarily one-to-one. In particular, different parameter values can yield the same function. For example, a neural network model with $b = 2$,

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \alpha_1 \phi(\boldsymbol{x};\boldsymbol{w}_1,\gamma_1) + \alpha_2 \phi(\boldsymbol{x};\boldsymbol{w}_2,\gamma_2)$$

for $\boldsymbol{w}_1 = \boldsymbol{w}_2 = \boldsymbol{w}$ and $\gamma_1 = \gamma_2 = \gamma$, represents the same function if $\alpha_1 + \alpha_2$ is a constant:

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = (\alpha_1 + \alpha_2)\phi(\boldsymbol{x}; \boldsymbol{w}, \gamma).$$

This causes the *Fisher information matrix* (see Section 13.3) to be singular, which makes it impossible to apply standard statistical machine learning theory. To cope with this problem, *Bayesian inference* was shown to be promising [119].

Due to the nonlinearity of the neural network model with respect to parameters, parameter learning is usually carried out by a gradient descent method called the *error back-propagation* algorithm [85]. However, only a local optimal solution may be found by the gradient method (see Section 15.3). To cope with this problem, running the gradient-based learning procedure multiple times from different initial solutions is effective. Furthermore, *pretraining* of each layer by an unsupervised learning method is demonstrated to be useful in practice [55].