

翻译并解释以下内容，翻译不能有遗漏，解释请通俗易懂且详尽，因为我的数学基础不太好，最后总结要点。

## 14 Kernels

### 14.1 Introduction

So far in this book, we have been assuming that each object that we wish to classify or cluster or process in anyway can be represented as a fixed-size feature vector, typically of the form  $x_i \in RD$ . However, for certain kinds of objects, it is not clear how to best represent them as fixed-sized feature vectors. For example, how do we represent a text document or protein sequence, which can be of variable length? or a molecular structure, which has complex 3d geometry? or an evolutionary tree, which has variable size and shape?

One approach to such problems is to define a generative model for the data, and use the inferred latent representation and/or the parameters of the model as features, and then to plug these features in to standard methods. For example, in Chapter 28, we discuss deep learning, which is essentially an unsupervised way to learn good feature representations.

Another approach is to assume that we have some way of measuring the similarity between objects, that doesn't require preprocessing them into feature vector format.

For example, when comparing strings, we can compute the edit distance between them. Let  $\kappa(x, x') \geq 0$  be some measure of similarity between objects  $x, x' \in X$ , where  $X$  is some abstract space; we will call  $\kappa$  a kernel function.

Note that the word "kernel" has several meanings; we will discuss a different interpretation in Section 14.7.1.

In this chapter, we will discuss several kinds of kernel functions. We then describe some algorithms that can be written purely in terms of kernel function computations. Such methods can be used when we don't have access to (or choose not to look at) the "inside" of the objects  $x$  that we are processing.

已思考 23s

**逐句准确翻译**

## 14 核方法 (Kernels)

### 14.1 引言 (Introduction)

到目前为止，在本书中我们一直假设：任何我们想要进行分类、聚类，或者以任何方式处理的对象，都可以表示成一个固定长度的特征向量，通常形如  $x_i \in \mathbb{R}^D$ 。然而，对于某些类型的对象，并不清楚应当如何把它们最好地表示成固定长度的特征向量。比如：我们如何表示一篇可以有可变长度的文本文档或蛋白质序列？又如何表示具有复杂三维几何结构的分子？或者如何表示大小与形状都可变的进化树？

解决这类问题的一种方法，是为数据定义一个**生成式模型**，并使用所推断的**潜在表示**和/或模型参数作为特征，然后把这些特征送入标准方法中。例如，在第28章我们会讨论深度学习，它本质上是一种无监督地学习良好特征表示的方法。

另一种方法是假设我们有某种**衡量对象间相似性**的方式，而且这种方式不需要把对象预处理成特征向量的格式。例如，在比较字符串时，我们可以计算它们之间的**编辑距离**。令  $\kappa(x, x') \geq 0$  表示对象  $x, x' \in X$  ( $X$  是某个抽象空间) 之间的一种相似性度量；我们称  $\kappa$  为**核函数 (kernel function)**。注意，“kernel”一词有多种含义；我们将在第14.7.1节讨论另一种解释。

在本章中，我们将讨论若干类型的核函数。随后，我们会描述一些只依赖于**核函数计算**就能书写的算法。当我们无法（或选择不）查看我们正在处理的对象  $x$  的“内部”结构时，这类方法就可以使用。

## 通俗而详尽的解释

### 这段话在说什么？

以前我们做机器学习时，常把每个样本都变成一个**固定长度**的数字列表（特征向量），再把这个列表丢给分类器、聚类器等算法处理。问题是：很多真实世界的数据**很难或不自然**地被塞进“固定长度列表”里，比如一篇文章（长短不一）、DNA/蛋白质序列（长度变化大）、一个分子的3D形状（结构复杂）、或一棵进化树（分叉数、深度都不同）。

### 两条常见思路：

#### 1. 学特征（生成式/表征学习路子）

- 给数据建一个**生成式模型**（它告诉你“数据可能是怎样生成的”），或用深度学习之类的方法自动学到一个**隐藏的低维表示**（潜在表示）。
- 然后把这些学到的表示/参数，当作“固定长度的特征”，再用常规算法（比如逻辑回归、SVM、KNN等）。
- 这就像先把原始复杂对象“翻译成”一个好用的向量，再进行后续任务。

## 2. 直接比相似（核方法路子）

- 与其强行把对象都变成同一种固定长度的向量，不如直接定义对象之间的相似程度。
- 例如，对字符串可以算**编辑距离**（改几步能把一个串变成另一个）。距离小=更相似。
- 进一步，我们用一个函数  $\kappa(x, x')$  来表示“**相似性**”（通常希望数值越大越相似），这个函数就叫**核函数 (kernel)**。
- 只要我们能算任意两件东西的“相似度”，很多算法就可以**只用这些相似度来完成学习，而不需要看对象的内部细节或把它们变成统一的向量**。

**小贴士：** 编辑“距离”是“差异”的度量，要变成“相似度”很常见的做法是套个函数，比如

$$\kappa(x, x') = \exp(-\gamma \cdot d(x, x')^2)$$

距离越小，指数越接近1，相似度越大；距离越大，相似度越接近0。

### 什么是核函数，直觉上怎么理解？

- 把所有样本摆成一个表，横纵轴都是样本，表格里填上“这俩有多像”，就得到一个**相似度矩阵**（也称Gram矩阵）。
- 核算法**可以只看这张“相似度表”，就做分类、回归、降维等工作。
- 这样，我们就**不必**把“文章/分子/树”这些复杂对象，先硬掰成同一长度的数字向量。

### 为什么这很有用？

- 它让我们能处理**结构各异、长度不一**的数据类型（文本、序列、图/树、三维形状……）。
- 它把“如何表示数据”这个难题，转化为“如何定义相似性”。只要相似性定义得合理，后续算法自然能利用。

### 需要注意的点（温和版）：

- 在很多经典核方法（如核SVM、核PCA）里，为了算法性质良好，核函数常被要求满足一种数学条件（**正定性**），简单理解就是：用它算出来的相似度矩阵应当“合理”（没有自相矛盾）。
- 实务中，常用的核（比如RBF/高斯核、线性核、多项式核）都满足这类条件；而像“编辑距离”属于**距离**，通常需要再做一步变换（例如上面的指数变换）才能当作核来用。

### 一句话抓核心：

与其费力把复杂对象做成统一的向量，不如定义“对象之间有多像”的规则（核函数）——很多强大的算法只需要你提供这种“相似度”，就能完成学习。

## 关键概念用更白话举例

- **固定长度特征向量：**

把一篇文章变成“100维数字”的表述（比如词频），每篇都必须是100个数。这叫固定长度。但真实文章长短不一、用词各异，强行定成100维可能会丢信息或很难统一。

- **生成式模型 / 深度学习学表征：**

想像一个“讲故事的模型”，它能说明文章可能是怎样写出来的；这个模型的“内心想法”（潜在变量）可能就是一个不错的、固定长度的表示。深度学习也会自动学到这样的表示。

- **核函数  $\kappa(x, x')$ ：**

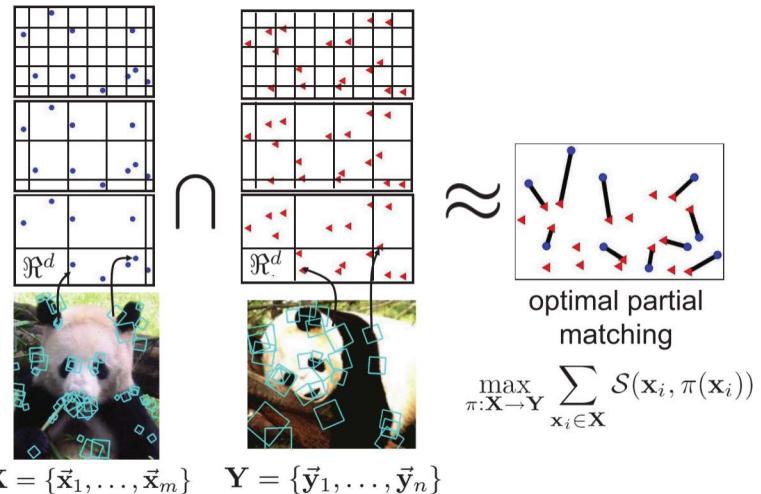
不看文章具体字词细节，只问：“A文和B文像不像？”你回答一个分数（比如0.82）。所有文档两两之间都回答一下，算法就能工作了。

- **只用相似度也能做算法：**

你给出“谁和谁像”，就能分群、分类。比如“把互相很像的一堆放一起”，或者“找一条边界，让相似度矩阵‘看起来’分成两类”。

## 要点总结

- **问题背景：**很多对象（文本、蛋白质序列、分子、树）难以自然地变成**固定长度向量**。
- **路子一（学特征）：**用**生成式模型或深度学习学到潜在表示/参数**，再当作特征。
- **路子二（核方法）：**直接定义对象间的**相似性**  $\kappa(x, x') \geq 0$ ，称为**核函数**，让算法只依赖“相似度”来学习。
- **好处：**能处理**不同长度/结构**的数据，无需深入解析对象内部或统一成同一向量格式。
- **提醒：**距离≠相似度；常把距离通过函数（如指数）转成相似度；经典核方法通常偏好**正定**的核。
- **本章走向：**先看多种核函数，再看**仅用核计算**就能实现的算法。



翻译并解释以下内容，翻译不能有遗漏，解释请通俗易懂且详尽，因为我的数学基础不太好，最后总结要点。

Figure 14.1 Illustration of a pyramid match kernel computed from two images.

## 14.2 Kernel functions

We define a kernel function to be a real-valued function of two arguments,  $\kappa(x, x') \in \mathbb{R}$ , for  $x, x' \in \mathcal{X}$ . Typically the function is symmetric (i.e.,  $\kappa(x, x') = \kappa(x', x)$ ), and non-negative (i.e.,  $\kappa(x, x') \geq 0$ ), so it can be interpreted as a measure of similarity, but this is not required. We give several examples below.

### 14.2.1 RBF kernels

The squared exponential kernel (SE kernel) or Gaussian kernel is defined by

$$\kappa(x, x') = \exp(-\frac{1}{2} \|x - x'\|^2) \quad (14.1)$$

If  $\Sigma$  is diagonal, this can be written as

$$\kappa(x, x') = \exp\left(-\frac{1}{2} \sum_{j=1}^D \sigma_j^2 (x_j - x'_j)^2\right) \quad (14.2)$$

We can interpret the  $\sigma_j$  as defining the characteristic length scale of dimension  $j$ . If  $\sigma_j = \infty$ , the corresponding dimension is ignored; hence this is known as the ARD kernel. If  $\Sigma$  is spherical, we get the isotropic kernel

$$\kappa(x, x') = \exp(-\|x - x'\|^2 / 2\sigma^2) \quad (14.3)$$

Here  $\sigma^2$  is known as the bandwidth. Equation 14.3 is an example of a radial basis function or RBF kernel, since it is only a function of  $\|x - x'\|$ .

### 14.2.2 Kernels for comparing documents

When performing document classification or retrieval, it is useful to have a way of comparing two documents,  $x_i$  and

$x_i$ . If we use a bag of words representation, where  $x_{ij}$  is the number of times word  $j$  occurs in document  $i$ , we can use the cosine similarity, which is defined by

$$\kappa(x_i, x_i) = \frac{x_i^T x_i}{\|x_i\|_2 \|x_i\|_2} \quad (14.4)$$

This quantity measures the cosine of the angle between  $x_i$  and  $x_i$  when interpreted as vectors. Since  $x_i$  is a count vector (and hence non-negative), the cosine similarity is between 0 and 1, where 0 means the vectors are orthogonal and therefore have no words in common.

Unfortunately, this simple method does not work very well, for two main reasons. First, if  $x_i$  has any word in common with  $x_i$ , it is deemed similar, even though some popular words, such as "the" or "and" occur in many documents, and are therefore not discriminative. (These are known as stop words.) Second, if a discriminative word occurs many times in a document, the similarity is artificially boosted, even though word usage tends to be bursty, meaning that once a word is used in a document it is very likely to be used again (see Section 3.5.5).

Fortunately, we can significantly improve performance using some simple preprocessing. The idea is to replace the word count vector with a new feature vector called the TF-IDF representation, which stands for "term frequency inverse document frequency". We define this as follows.

First, the term frequency is defined as a log-transform of the count:

$$tf(x_{ij}) = \log(1 + x_{ij}) \quad (14.5)$$

This reduces the impact of words that occur many times within one document. Second, the inverse document frequency is defined as

$$idf(j) = \log \frac{N}{1 + \sum_{i=1}^N I(x_{ij} > 0)} \quad (14.6)$$

where  $N$  is the total number of documents, and the denominator counts how many documents contain term  $j$ .

Finally, we define

$$tf-idf(x_i) = [tf(x_{ij}) \times idf(j)] \quad \forall j=1 \dots N \quad (14.7)$$

(There are several other ways to define the tf and idf terms, see (Manning et al. 2008) for details.) We then use this inside the cosine similarity measure. That is, our new kernel has the form

$$\kappa(x_i, x_i) = \frac{\varphi(x_i)^T \varphi(x_i)}{\|\varphi(x_i)\|_2 \|\varphi(x_i)\|_2} \quad (14.8)$$

where  $\varphi(x) = tf-idf(x)$ . This gives good results for information retrieval (Manning et al. 2008). A probabilistic interpretation of the tf-idf kernel is given in (Elkan 2005).

### 14.2.3 Mercer (positive definite) kernels

Some methods that we will study require that the kernel function satisfy the requirement that the Gram matrix, defined by

$$K = \left( \begin{array}{c|c} & \kappa(x_1, x_1) \cdots \kappa(x_1, x_N) \cdots \kappa(x_N, x_1) \cdots \kappa(x_N, x_N) \end{array} \right) \quad (14.9)$$

be positive definite for any set of inputs  $\{x_i\}_N$ . We call such a kernel a Mercer kernel, or positive definite kernel. It can be shown (Schoelkopf and Smola 2002) that the Gaussian kernel is a Mercer kernel as is the cosine similarity kernel (Sahami and Heilman 2006). The importance of Mercer kernels is the following result, known as Mercer's theorem. If the Gram matrix is positive definite, we can compute an eigenvector decomposition of it as follows

$$K = U \Lambda U^T \quad (14.10)$$

where  $\Lambda$  is a diagonal matrix of eigenvalues  $\lambda_i > 0$ . Now consider an element of  $K$ :

$$k_{ij} = (\Lambda^{-1/2} U^T)^T (\Lambda^{1/2} U)_{ij} \quad (14.11)$$

Let us define  $\varphi(x_i) = \Lambda^{1/2} U^T_{:,i}$ . Then we can write

$$k_{ij} = \varphi(x_i)^T \varphi(x_j) \quad (14.12)$$

Thus we see that the entries in the kernel matrix can be computed by performing an inner product of some feature vectors that are implicitly defined by the eigenvectors  $U$ . In general, if the kernel is Mercer, then there exists a function  $\varphi$  mapping  $x \in X$  to  $R^D$  such that

$$\kappa(x, x') = \varphi(x)^T \varphi(x') \quad (14.13)$$

where  $\varphi$  depends on the eigen functions of  $\kappa$  (so  $D$  is a potentially infinite dimensional space). For example, consider the (non-stationary) polynomial kernel  $\kappa(x, x') = (yx^T x' + r)^M$ , where  $r > 0$ . One can show that the corresponding feature vector  $\varphi(x)$  will contain all terms up to degree  $M$ . For example, if  $M = 2$ ,  $y = r = 1$  and  $x, x' \in R^2$ , we have

$$(1 + x^T x')^2 = (1 + x_1 x'_1 + x_2 x'_2)^2 \quad (14.14)$$

$$= 1 + 2x_1 x'_1 + 2x_2 x'_2 + (x_1 x'_1)^2 + (x_2 x'_2)^2 + 2x_1 x'_1 x_2 x'_2 \quad (14.15)$$

This can be written as  $\varphi(x)^T \varphi(x')$ , where

$$\varphi(x) = [1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2]^T \quad (14.16)$$

So using this kernel is equivalent to working in a 6 dimensional feature space. In the case of a Gaussian kernel, the feature map lives in an infinite dimensional space. In such a case, it is clearly infeasible to explicitly

represent the feature vectors. An example of a kernel that is not a Mercer kernel is the so-called sigmoid kernel, defined by

$$\kappa(x, x') = \tanh(yx^T x' + r) \quad (14.17)$$

(Note that this uses the tanh function even though it is called a sigmoid kernel.) This kernel was inspired by the multi-layer perceptron (see Section 16.5), but there is no real reason to use it. (For a true “neural net kernel”, which is positive definite, see Section 15.4.5.) In general, establishing that a kernel is a Mercer kernel is difficult, and requires techniques from functional analysis.

However, one can show that it is possible to build up new Mercer kernels from simpler ones using a set of standard rules. For example, if  $\kappa_1$  and  $\kappa_2$  are both Mercer, so is  $\kappa(x, x') = \kappa_1(x, x') + \kappa_2(x, x')$ . See e.g., (Schoelkopf and Smola 2002) for details.

#### 14.2.4 Linear kernels

Deriving the feature vector implied by a kernel is in general quite difficult, and only possible if the kernel is Mercer. However, deriving a kernel from a feature vector is easy: we just use

$$\kappa(x, x') = \varphi(x)^T \varphi(x') = \varphi(x), \varphi(x') \quad (14.18)$$

If  $\varphi(x) = x$ , we get the linear kernel, defined by

$$\kappa(x, x') = x^T x' \quad (14.19)$$

This is useful if the original data is already high dimensional, and if the original features are individually informative, e.g., a bag of words representation where the vocabulary size is large, or the expression level of many genes. In such a case, the decision boundary is likely to be representable as a linear combination of the original features, so it is not necessary to work in some other feature space. Of course, not all high dimensional problems are linearly separable. For example, images are high dimensional, but individual pixels are not very informative, so image classification typically requires non-linear kernels (see e.g., Section 14.2.7).

#### 14.2.5 Matern kernels

The Matern kernel, which is commonly used in Gaussian process regression (see Section 15.2), has the following form

$$\kappa(r) = 2^{1-v} \Gamma(v) \sqrt{2v} r^v K_v \sqrt{2v} r \quad (14.20)$$

where  $r = \|x - x'\|$ ,  $v > 0$ , and  $K_v$  is a modified Bessel function. As  $v \rightarrow \infty$ , this approaches the SE kernel. If

$v = 1/2$ , the kernel simplifies to

$$\kappa(r) = \exp(-r/l) \quad (14.21)$$

If  $D = 1$ , and we use this kernel to define a Gaussian process (see Chapter 15), we get the Ornstein-Uhlenbeck process, which describes the velocity of a particle undergoing Brownian motion (the corresponding function is continuous but not differentiable, and hence is very “jagged”).

#### 14.2.6 String kernels

The real power of kernels arises when the inputs are structured objects. As an example, we now describe one way of comparing two variable length strings using a string kernel. We follow the presentation of (Rasmussen and Williams 2006, p100) and (Hastie et al. 2009, p668). Consider two strings  $x$ , and  $x'$  of lengths  $D$ ,  $D'$ , each defined over the alphabet  $A$ . For example, consider two amino acid sequences, defined over the 20 letter alphabet  $A = \{A, R, N, D, C, E, Q, G, H, I, L, K, M, F, P, S, T, W, Y, V\}$ . Let  $x$  be the following sequence of length 110

IPTSLALVKETLALLSTHRTLLIANETLRIPVPVHKKNHQLCTEEIFQG  
IGTLESQTVQGGTV  
ERLFKNLSLIKYYIDGQKKKCGEERRRVNQFLDYLQEFLGVMNT  
EWI

and let  $x'$  be the following sequence of length 153

PHRRDLCRSIWLARKIRSDLTALTESYVKHQGLWSELTEAERLQ  
ENLQAYRTFHVLIA  
RLLEDQQVHFTPTEGDFHQAIHTLLQVAFAFAYQIEELMILLEYKI  
PRNEADGMLFEKK

LWGLKVLQELSQWTVRSIHDLRFISSHQTGIP

These strings have the substring LQE in common. We can define the similarity of two strings to be the number of substrings they have in common. More formally and more generally, let us say that  $s$  is a substring of  $x$  if we can write  $x = usv$  for some (possibly empty) strings  $u$ ,  $s$  and  $v$ . Now let  $\varphi_s(x)$  denote the number of times that substring  $s$  appears in string  $x$ . We define the kernel between two strings  $x$  and  $x'$  as

$$\kappa(x, x') = \sum_{s \in A^*} w_s \varphi_s(x) \varphi_s(x') \quad (14.22)$$

where  $w_s \geq 0$  and  $A^*$  is the set of all strings (of any length) from the alphabet  $A$  (this is known as the Kleene star operator). This is a Mercer kernel, and be computed in  $O(|x| + |x'|)$  time (for certain settings of the weights  $\{w_s\}$ ) using suffix trees (Leslie et al. 2003; Vishwanathan

and Smola 2003; Shawe-Taylor and Cristianini 2004). There are various cases of interest. If we set  $ws = 0$  for  $|s| > 1$  we get a bag-of-characters kernel. This defines  $\varphi(x)$  to be the number of times each character in A occurs in x. If we require s to be bordered by white-space, we get a bag-of-words kernel, where  $\varphi(x)$  counts how many times each possible word occurs. Note that this is a very sparse vector, since most words will not be present. If we only consider strings of a fixed length k, we get the k-spectrum kernel. This has been used to classify proteins into SCOP superfamilies (Leslie et al. 2003). For example if k = 3, we have  $\varphi LQE(x) = 1$  and  $\varphi LQE(x) = 2$  for the two strings above. Various extensions are possible. For example, we can allow character mismatches (Leslie et al. 2003). And we can generalize string kernels to compare trees, as described in (Collins and Duffy 2002). This is useful for classifying (or ranking) parse trees, evolutionary trees, etc.

#### 14.2.7 Pyramid match kernels

In computer vision, it is common to create a bag-of-words representation of an image by computing a feature vector (often using SIFT (Lowe 1999)) from a variety of points in the image, commonly chosen by an interest point detector. The feature vectors at the chosen places are then vector-quantized to create a bag of discrete symbols. One way to compare two variable-sized bags of this kind is to use a pyramid match kernel (Grauman and Darrell 2007). The basic idea is illustrated in Figure 14.1. Each feature set is mapped to a multi-resolution histogram. These are then compared using weighted histogram intersection. It turns out that this provides a good approximation to the similarity measure one would obtain by performing an optimal bipartite match at the finest spatial resolution, and then summing up pairwise similarities between matched points. However, the histogram method is faster and is more robust to missing and unequal numbers of points. This is a Mercer kernel.

#### 14.2.8 Kernels derived from probabilistic generative models

Suppose we have a probabilistic generative model of feature vectors,  $p(x|\theta)$ . Then there are several ways we can use this model to define kernel functions, and thereby make the model suitable for discriminative tasks. We sketch two approaches below.

#### 14.2.8.1 Probability product kernels

One approach is to define a kernel as follows:

$$\kappa(x_i, x_j) = p(x|x_i)p(x|x_j)p_{\theta}(x) \quad (14.23)$$

where  $p > 0$ , and  $p(x|x_i)$  is often approximated by  $p(x|\hat{\theta}(x_i))$ , where  $\hat{\theta}(x_i)$  is a parameter estimate computed using a single data vector. This is called a probability product kernel (Jebara et al. 2004). Although it seems strange to fit a model to a single data point, it is important to bear in mind that the fitted model is only being used to see how similar two objects are. In particular, if we fit the model to  $x_i$  and then the model thinks  $x_j$  is likely, this means that  $x_i$  and  $x_j$  are similar. For example, suppose  $p(x|\theta) = N(\mu, \sigma^2 I)$ , where  $\sigma^2$  is fixed. If  $\mu = x_i$ , and we use  $\hat{\mu}(x_i) = x_i$  and  $\hat{\mu}(x_j) = x_j$ , we find (Jebara et al. 2004, p825) that

$$\kappa(x_i, x_j) = 1/(4\pi\sigma^2 D/2) \exp(-1/4\sigma^2 \|x_i - x_j\|^2) \quad (14.24)$$

which is (up to a constant factor) the RBF kernel. It turns out that one can compute Equation 14.23 for a variety of generative models, including ones with latent variables, such as HMMs. This provides one way to define kernels on variable length sequences. Furthermore, this technique works even if the sequences are of real-valued vectors, unlike the string kernel in Section 14.2.6. See (Jebara et al. 2004) for further details.

#### 14.2.8.2 Fisher kernels

A more efficient way to use generative models to define kernels is to use a Fisher kernel (Jaakkola and Haussler 1998) which is defined as follows:

$$\kappa(x, x') = g(x)^T F^{-1} g(x') \quad (14.25)$$

where  $g$  is the gradient of the log likelihood, or score vector, evaluated at the MLE  $\hat{\theta}$

$$g(x) = \nabla_{\theta} \log p(x|\theta) \quad (\hat{\theta}) \quad (14.26)$$

and  $F$  is the Fisher information matrix, which is essentially the Hessian:

$$F = \nabla \nabla \log p(x|\theta) \quad (\hat{\theta}) \quad (14.27)$$

Note that  $\hat{\theta}$  is a function of all the data, so the similarity of  $x$  and  $x'$  is computed in the context of all the data as well. Also, note that we only have to fit one model. The intuition behind the Fisher kernel is the following: let  $g(x)$  be the direction (in parameter space) in which  $x$  would like the parameters to move (from  $\hat{\theta}$ ) so as to maximize its own likelihood; call this the directional gradient. Then we say that two vectors  $x$  and  $x'$  are similar if their directional

gradients are similar wrt the the geometry encoded by the curvature of the likelihood function (see Section 7.5.3). Interestingly, it was shown in (Saunders et al. 2003) that the string kernel of Section 14.2.6 is equivalent to the Fisher kernel derived from an L'th order Markov chain (see Section 17.2). Also, it was shown in (Elkan 2005) that a kernel defined by the inner product of TF-IDF vectors (Section 14.2.2) is approximately equal to the Fisher kernel for a certain generative model of text based on the compound Dirichlet multinomial model (Section 3.5.5).

已思考 10s

下面先给出逐字逐句的准确中文翻译（含公式与编号），随后是通俗详尽的解释，最后给出要点总结。我会把原文的符号（如  $x'$ ）统一写成带撇号的形式，保证意思与公式不变。

---

## 逐句翻译

图 14.1 由两幅图像计算得到的金字塔匹配核（pyramid match kernel）的示意图。

## 14.2 核函数 (Kernel functions)

我们把核函数定义为一个关于两个自变量的实值函数， $\kappa(x, x') \in \mathbb{R}$ ，其中  $x, x' \in X$ 。通常该函数是对称的（即  $\kappa(x, x') = \kappa(x', x)$ ），且非负（即  $\kappa(x, x') \geq 0$ ），因此它可以被解释为一种相似性度量，但这并不是必须的。下面我们给出若干例子。

### 14.2.1 RBF 核

**平方指数核** (SE kernel) 或**高斯核**定义为

$$\kappa(x, x') = \exp(-\frac{1}{2}(x - x')^T \Sigma^{-1} (x - x')) \quad (14.1)$$

如果  $\Sigma$  是对角矩阵，它可以写成

$$\kappa(x, x') = \exp(-\frac{1}{2} \sum_{j=1}^D \frac{1}{\sigma_j^2} (x_j - x'_j)^2) \quad (14.2)$$

我们可以把  $\sigma_j$  理解为第  $j$  个维度的**特征长度尺度**。如果  $\sigma_j = \infty$ ，与之对应的维度会被忽略；因此这被称为**ARD 核**。如果  $\Sigma$  是球形的，我们得到**各向同性核**

$$\kappa(x, x') = \exp(-\frac{\|x - x'\|^2}{2\sigma^2}) \quad (14.3)$$

这里  $\sigma^2$  被称为带宽。式 (14.3) 是径向基函数 (RBF) 核的一个例子，因为它只依赖于  $\|x - x'\|$ 。

## 14.2.2 比较文档的核

在进行文档分类或检索时，拥有一种比较两篇文档  $x_i$  和  $x'_i$  的方法是有用的。如果我们使用词袋 (bag of words) 表示，其中  $x_{ij}$  是词  $j$  在文档  $i$  中出现的次数，我们可以使用余弦相似度，其定义为

$$\kappa(x_i, x'_i) = \frac{x_i^T x'_i}{\|x_i\|_2 \|x'_i\|_2} \quad (14.4)$$

当把  $x_i, x'_i$  视为向量时，该量测量它们之间夹角的余弦。由于  $x_i$  是计数向量（因此非负），余弦相似度介于 0 与 1 之间，其中 0 表示向量正交、因此没有任何共同词。不幸的是，这种简单方法并不好用，主要有两个原因。第一，只要  $x_i$  与  $x'_i$  有任何共同词，它们就被判定为相似，即使一些高频词（如 “the” 或 “and”）在许多文档中都会出现，因此并没有区分能力（这些称为停用词）。第二，如果一个具有区分力的词在一篇文档中出现了很多次，相似度会被人为放大，尽管词的使用往往具有“爆发性”（bursty）——一旦在文档中使用了某个词，它很可能被反复使用（见 3.5.5 节）。幸运的是，通过一些简单的预处理，我们可以显著提升性能。思想是用一个称为 TF-IDF（词频—逆文档频率）的新特征向量替换原始词频向量。我们如下定义它。首先，词频定义为计数的对数变换：

$$\text{tf}(x_{ij}) = \log(1 + x_{ij}) \quad (14.5)$$

这减弱了在同一文档中多次出现的词的影响。第二，逆文档频率定义为

$$\text{idf}(j) = \log \frac{N}{1 + \sum_{i=1}^N \mathbf{I}(x_{ij} > 0)} \quad (14.6)$$

其中  $N$  是文档总数，分母统计包含词  $j$  的文档数。最后我们定义

$$\text{tf-idf}(x_i) = [\text{tf}(x_{ij}) \times \text{idf}(j)]_{j=1}^V \quad (14.7)$$

（关于 tf 与 idf 的其他定义方式有若干，详见 Manning 等 (2008)）。随后，我们把它用在余弦相似度中。也就是说，我们新的核具有如下形式

$$\kappa(x_i, x'_i) = \frac{\phi(x_i)^T \phi(x'_i)}{\|\phi(x_i)\|_2 \|\phi(x'_i)\|_2} \quad (14.8)$$

其中  $\phi(x) = \text{tf-idf}(x)$ 。这在信息检索中给出了很好的结果 (Manning 等, 2008)。对 TF-IDF 核的概率解释见 Elkan (2005)。

## 14.2.3 Mercer (正定) 核

我们将要研究的一些方法要求核函数满足这样一个条件：由

$$K = \begin{pmatrix} \kappa(x_1, x_1) & \cdots & \kappa(x_1, x_N) \\ \vdots & \ddots & \vdots \\ \kappa(x_N, x_1) & \cdots & \kappa(x_N, x_N) \end{pmatrix} \quad (14.9)$$

定义的Gram 矩阵对任意输入集合  $\{x_i\}_{i=1}^N$  都是**正定**的。我们称这样的核为 Mercer 核或**正定核**。可以证明 (Schölkopf 和 Smola, 2002)，高斯核是 Mercer 核，余弦相似度核也是 (Sahami 和 Heilman, 2006)。Mercer 核的重要性体现在如下结果，即**Mercer 定理**。如果 Gram 矩阵是正定的，我们可以对其做特征分解：

$$K = U^T \Lambda U \quad (14.10)$$

其中  $\Lambda$  是特征值对角矩阵， $\lambda_i > 0$ 。现在考虑  $K$  的一个元素：

$$k_{ij} = (\Lambda^{1/2} U_{:,i})^T (\Lambda^{1/2} U_{:,j}) \quad (14.11)$$

令  $\phi(x_i) = \Lambda^{1/2} U_{:,i}$ 。则可写作

$$k_{ij} = \phi(x_i)^T \phi(x_j) \quad (14.12)$$

因此我们看到，核矩阵中的各项可以通过对某些特征向量做内积得到，而这些特征向量由特征向量矩阵  $U$  **隐式定义**。一般来说，如果核是 Mercer 的，那么存在一个函数  $\phi$  将  $x \in X$  映射到  $\mathbb{R}^D$ ，使得

$$\kappa(x, x') = \phi(x)^T \phi(x') \quad (14.13)$$

其中  $\phi$  依赖于  $\kappa$  的本征函数（因此  $D$  可能是**无限维**空间）。例如，考虑（非平稳的）**多项式核**  $\kappa(x, x') = (\gamma x^T x' + r)^M$ ，其中  $r > 0$ 。可证明相应的特征向量  $\phi(x)$  会包含所有不超过  $M$  次的项。比如，当  $M = 2$ ,  $\gamma = r = 1$ ,  $x, x' \in \mathbb{R}^2$  时，

$$(1 + x^T x')^2 = (1 + x_1 x'_1 + x_2 x'_2)^2 \quad (14.14)$$

$$= 1 + 2x_1 x'_1 + 2x_2 x'_2 + (x_1 x'_1)^2 + (x_2 x'_2)^2 + 2x_1 x'_1 x_2 x'_2 \quad (14.15)$$

这可以写为  $\phi(x)^T \phi(x')$ ，其中

$$\phi(x) = [1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2]^T \quad (14.16)$$

因此使用该核等价于在一个**6 维**特征空间中工作。对于高斯核，其特征映射位于**无限维**空间。在这种情况下，显式表示特征向量显然不可行。一个**不是** Mercer 核的例子是所谓的**sigmoid 核**，其定义为

$$\kappa(x, x') = \tanh(\gamma x^T x' + r) \quad (14.17)$$

(注意它使用的是  $\tanh$  函数，尽管名字叫 sigmoid 核。) 该核的灵感来自多层感知机 (见 16.5 节)，但并没有真正的理由使用它。（若想要一个真正**正定**的“神经网络核”，见

15.4.5 节。) 一般而言, 要证明某个核是 Mercer 核是困难的, 需要泛函分析中的技术。然而, 可以证明可用一组标准规则由简单核构造新的 Mercer 核。例如, 若  $\kappa_1$  与  $\kappa_2$  都是 Mercer 核, 则  $\kappa(x, x') = \kappa_1(x, x') + \kappa_2(x, x')$  也是。详情见 Schölkopf 与 Smola (2002)。

#### 14.2.4 线性核

从核推出隐含的特征向量通常相当困难, 而且只有当核是 Mercer 核时才可能。然而, 从特征向量构造核很容易: 只需使用

$$\kappa(x, x') = \phi(x)^T \phi(x') = \langle \phi(x), \phi(x') \rangle \quad (14.18)$$

若取  $\phi(x) = x$ , 得到**线性核**, 定义为

$$\kappa(x, x') = x^T x' \quad (14.19)$$

当原始数据已经是**高维**, 且原始特征**单个就有信息量**时, 这很有用, 例如词汇量很大的词袋表示, 或许多基因的表达水平。在这种情况下, 决策边界很可能可以表示为原始特征的线性组合, 因此没必要在其他特征空间中工作。当然, 并非所有高维问题都是线性可分的。例如, 图像是高维的, 但单个像素并不十分有信息量, 因此图像分类通常需要**非线性核** (见 14.2.7 节等)。

#### 14.2.5 Matérn 核

Matérn 核在高斯过程回归中被广泛使用 (见 15.2 节), 其形式为

$$\kappa(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} (\sqrt{2\nu} r)^\nu K_\nu(\sqrt{2\nu} r) \quad (14.20)$$

其中  $r = \|x - x'\|$ ,  $\nu > 0$ ,  $l > 0$  (原文用  $\ell$ ),  $K_\nu$  是修正的 Bessel 函数。当  $\nu \rightarrow \infty$  时, 它趋近于 SE 核。若  $\nu = \frac{1}{2}$ , 该核化简为

$$\kappa(r) = \exp(-r/l) \quad (14.21)$$

若  $D = 1$ , 并用该核定义一个高斯过程 (见第 15 章), 我们得到**Ornstein–Uhlenbeck 过程**, 它描述做 Brownian 运动的粒子的速度 (相应函数是连续但不可导的, 因此非常“锯齿状”)。

#### 14.2.6 字符串核

当输入是结构化对象时, 核方法的真正威力就体现出来了。作为示例, 我们现在描述一种使用**字符串核**比较两个可变长度字符串的方法。我们遵循 Rasmussen 与 Williams (2006, 第 100 页) 以及 Hastie 等 (2009, 第 668 页) 的表述。考虑两个字符串  $x$  与  $x'$ , 其长度分别为  $D, D'$ , 定义在字母表 A 上。例如, 考虑两段氨基酸序列, 定义在 20 个字母的字母表  $A = \{A, R, N, D, C, E, Q, G, H, I, L, K, M, F, P, S, T, W, Y, V\}$  上。令  $x$  为下

面长度为 110 的序列

(此处原文给出了一段具体的 110 长度序列)

并令  $x'$  为下面长度为 153 的序列

(此处原文给出了一段具体的 153 长度序列)

这两个字符串具有公共子串 LQE。我们可以把两字符串的相似性定义为它们拥有的**公共子串的数量**。更形式化更一般地，若存在（可能为空的）字符串  $u, s, v$ ，使得  $x = usv$ ，我们就说  $s$  是  $x$  的一个子串。现在令  $\phi_s(x)$  表示子串  $s$  在字符串  $x$  中出现的次数。我们把两个字符串  $x$  与  $x'$  之间的核定义为

$$\kappa(x, x') = \sum_{s \in A^*} w_s \phi_s(x) \phi_s(x') \quad (14.22)$$

其中  $w_s \geq 0$ ，而  $A^*$  是由字母表  $A$  组成的**所有（任意长度）字符串**的集合（称为 Kleene 星）。这确实是一个 Mercer 核，并且（在某些权重  $\{w_s\}$  的设定下）可用**后缀树**在  $O(|x| + |x'|)$  时间内计算（Leslie 等，2003；Vishwanathan 与 Smola，2003；Shawe-Taylor 与 Cristianini，2004）。有多种有趣的特例。若对  $|s| > 1$  取  $w_s = 0$ ，我们得到**字符袋核**。它令  $\phi(x)$  为字母表中每个字符在  $x$  中出现的次数。若要求  $s$  被空白字符包围，我们得到**词袋核**，其中  $\phi(x)$  统计每个可能的单词出现了多少次。注意这会是一个非常稀疏的向量，因为大多数单词不会出现。若只考虑固定长度  $k$  的字符串，我们得到**k-谱**

**(spectrum) 核**。它已被用于把蛋白质分类到 SCOP 超家族中（Leslie 等，2003）。例如若  $k = 3$ ，对上面的两段字符串有  $\phi_{LQE}(x) = 1$ 、 $\phi_{LQE}(x') = 2$ 。可以做各种扩展。例如，我们可以允许字符**错配**（Leslie 等，2003）。我们还可以把字符串核推广到**比较树**（Collins 与 Duffy，2002），这对分类（或排序）句法树、进化树等是有用的。

### 14.2.7 金字塔匹配核

在计算机视觉中，常见做法是：对图像中的多个位置计算特征向量（通常使用 SIFT（Lowe，1999）），这些位置通常由**兴趣点检测器**选出；然后把这些位置的特征向量做**量化**，得到离散符号的“词袋”。比较两组这种**可变大小**的“袋子”的一种方法是使用**金字塔匹配核**（Grauman 与 Darrell，2007）。其基本思想如图 14.1 所示。把每个特征集合作映射到一个**多分辨率直方图**；然后用**带权的直方图交**来比较它们。结果表明，这对在最精细的空间分辨率下执行**最优二分图匹配**并随后对匹配点的两两相似度求和所得到的相似性是一种良好的近似。然而，直方图方法更快，并且对**缺失点**和**点数不相等**更鲁棒。这是一个 Mercer 核。

### 14.2.8 从概率生成模型导出的核

设我们有一个关于特征向量的**概率生成模型**  $p(x | \theta)$ 。那么有若干方法可以利用该模型来定义核函数，从而使模型适用于判别式任务。下面我们勾勒两种方法。

#### 14.2.8.1 概率乘积核 (Probability product kernels)

一种方法是定义如下的核：

$$\kappa(x_i, x_j) = \int p(x | x_i)^\rho p(x | x_j)^\rho dx \quad (14.23)$$

其中  $\rho > 0$ , 且  $p(x | x_i)$  常用  $p(x | \hat{\theta}(x_i))$  近似, 这里的  $\hat{\theta}(x_i)$  是使用**单个**数据向量估计得到的参数。这被称为**概率乘积核** (Jebara 等, 2004)。尽管给单个样本拟合一个模型看起来有点奇怪, 但要牢记: 拟合的模型仅用于判断两个对象有多相似。具体地说, 如果我们把模型拟合到  $x_i$ , 而模型认为  $x_j$  很可能 (出现), 这就意味着  $x_i$  与  $x_j$  相似。举例来说, 若  $p(x | \theta) = N(\mu, \sigma^2 I)$ , 且  $\sigma^2$  固定; 当  $\rho = 1$  且取  $\hat{\mu}(x_i) = x_i$ ,  $\hat{\mu}(x_j) = x_j$  时, 可以得到 (Jebara 等, 2004, 第 825 页)

$$\kappa(x_i, x_j) = \frac{1}{(4\pi\sigma^2)^{D/2}} \exp\left(-\frac{\|x_i - x_j\|^2}{4\sigma^2}\right) \quad (14.24)$$

这 (在一个常数因子之外) 就是 RBF 核。事实证明, 可以对多种生成模型 (包括带**潜变量**的, 如 HMM) 计算式 (14.23)。这为在**可变长度序列**上定义核提供了一种方法。此外, 该技术即使在序列的元素是**实值向量**时也有效, 这与 14.2.6 节中的字符串核不同。更多细节见 Jebara 等 (2004)。

#### 14.2.8.2 Fisher 核

一种更高效地使用生成模型来定义核的方法是使用 **Fisher 核** (Jaakkola 与 Haussler, 1998), 其定义为

$$\kappa(x, x') = g(x)^T F^{-1} g(x') \quad (14.25)$$

其中  $g$  是在极大似然估计  $\hat{\theta}$  处评估的**对数似然梯度** (或**得分向量**) :

$$g(x) = \nabla_{\theta} \log p(x | \theta) \Big|_{\hat{\theta}} \quad (14.26)$$

而  $F$  是**Fisher 信息矩阵**, 本质上是 Hessian:

$$F = \nabla \nabla \log p(x | \theta) \Big|_{\hat{\theta}} \quad (14.27)$$

注意  $\hat{\theta}$  是**所有数据**的函数, 因此  $x$  与  $x'$  的相似性也是在所有数据的背景下计算的。另外要注意我们只需拟合**一个**模型。Fisher 核的直觉是: 令  $g(x)$  表示这样一个方向 (在参数空间中): 从  $\hat{\theta}$  出发,  $x$  希望 (为了最大化自身似然) 把参数往这个方向“推”——称为**方向梯度**。那么, 如果两个向量  $x$  与  $x'$  的方向梯度在由似然函数曲率 (见 7.5.3 节) 所编码的几何意义上相似, 我们就认为它们相似。有趣的是, Saunders 等 (2003) 表明, 14.2.6 节的字符串核等价于从  $L$  阶马尔可夫链 (见 17.2 节) 导出的 Fisher 核。Elkan (2005) 还表明, 由 TF-IDF 向量内积 (14.2.2 节) 定义的核, 与基于**复合狄利克雷多项式模型** (3.5.5 节) 的一种文本生成模型的 Fisher 核近似相等。

# 通俗详尽的解释

## 图 14.1 在说什么？

两张熊猫图片 → 在每张图上提取许多局部特征点（蓝点、红三角）。把特征空间用“网格金字塔”（粗→细的多层次网格）划成箱子，把特征落到这些箱子里，形成多分辨率直方图。两图的直方图做“带权交集”（粗层给小权、细层给大权），得到一个分数。这个分数近似于“把两图特征一一匹配的最优部分匹配”所得的相似度，但算直方图要快得多，也不怕两边点数不一致或有缺失。因此这就是**金字塔匹配核**。

---

## 1) 什么是“核”？

把  $\kappa(x, x')$  想成“**相似度打分器**”。它输入两个对象（不一定是普通向量，甚至可以是字符串、树、图像特征集合），输出一个数，数越大越像。很多算法（如 SVM、核 PCA、谱聚类）只要看成对的相似度表（Gram 矩阵）就能工作。

## 2) RBF / 高斯核 (14.2.1)

- 看两点距离：近 → 分数接近 1；远 → 分数接近 0。
- $\Sigma$  控制各方向的“尺度”。ARD 情况下每维都有自己的  $\sigma_j$ ： $\sigma_j$  大表示“这一维没那么重要”。
- **带宽**  $\sigma$ ：太小 → 只把非常近的点当相似，容易过拟合；太大 → 大家都差不多，容易欠拟合。

## 3) 文档核与 TF-IDF (14.2.2)

- 直接用词频做**余弦相似度**会被停用词和“同一文档内重复很多次的词”误导。
- **TF-IDF** 两招：
  - tf:  $\log(1 + \text{词频})$  抑制“同文档内重复”。
  - idf:  $\log \frac{N}{1 + \text{含该词的文档数}}$  降低“到处都出现的词”的权重。
- 再用余弦相似度比 TF-IDF 向量，就能更合理地度量两篇文档的相似性。

## 4) Mercer/正定核与“隐式特征空间” (14.2.3)

- 若对所有样本算出的 Gram 矩阵  $K$  都是**正定**，这个核就是 **Mercer 核**。
- Mercer 定理告诉我们：存在某个（也许是**无限维**的）特征映射  $\phi(\cdot)$ ，使得  $\kappa(x, x') = \langle \phi(x), \phi(x') \rangle$ 。  
也就是说，**算核 = 在某个特征空间里做内积**。
- 多项式核例子展示：用核等价于在更高维里加入一堆交叉项与平方项；不用真的把特征展开，也能享受“高维”的好处。
- Sigmoid 核并不总是正定，因此不太推荐；而高斯、余弦等是正定的。

- 正定核可以通过加法、乘法、与正函数的点乘等规则拼装出来。

## 5) 线性核 (14.2.4)

- 直接取  $\phi(x) = x$ , 核就是  $x^T x'$ 。
- 当原始特征本身就很强 (如高维稀疏词袋、很多基因表达值), 线性模型就很好; 图像像素这种“单维弱信息”的高维数据通常需要非线性核。

## 6) Matérn 核 (14.2.5)

- 一个“家族”核, 用参数  $\nu$  调节平滑度:  $\nu$  越大, 函数越平滑, 极限就是 SE 核;  $\nu = \frac{1}{2}$  时是  $\exp(-r/l)$ 。
- 在 1 维、 $\nu = \frac{1}{2}$  时, 对应著名的 Ornstein–Uhlenbeck 过程 (连续但不可导, 看起来“锯齿”)。

## 7) 字符串核 (14.2.6)

- 想法: 两个串越像, **共享的子串越多**。
- 为所有可能子串  $s$  计数  $\phi_s(x)$ , 用带权和  $\sum_s w_s \phi_s(x) \phi_s(x')$ 。
- 选不同的  $w_s$  或限制  $s$  的长度, 就得到不同变体: 字符袋、词袋、k-谱核 (常用  $k = 3$  来比蛋白质片段)。
- 可用后缀树在线性时间里算出来; 还能允许错配, 并推广到**树结构** (比较句法树、进化树)。

## 8) 金字塔匹配核 (14.2.7)

- 用于**可变大小的特征集合** (如图像里任意多的局部特征)。
- 把特征落进多层网格直方图 → 计算带权直方图交集 → 近似“最优匹配”的相似度, 但速度更快、对点数不等更稳健。
- 这是一个**正定核**。

## 9) 从生成模型造核 (14.2.8)

### A. 概率乘积核

- 用两“类条件分布”的乘积积分来度量相似: 如果由  $x_i$  拟合的模型也“觉得”  $x_j$  很可能, 那二者就相似。
- 对高斯模型推导可得一个 RBF 形式, 因而直觉上合理。还能用于 HMM 等带隐变量的模型 → 适合**可变长度的实值序列**。

### B. Fisher 核

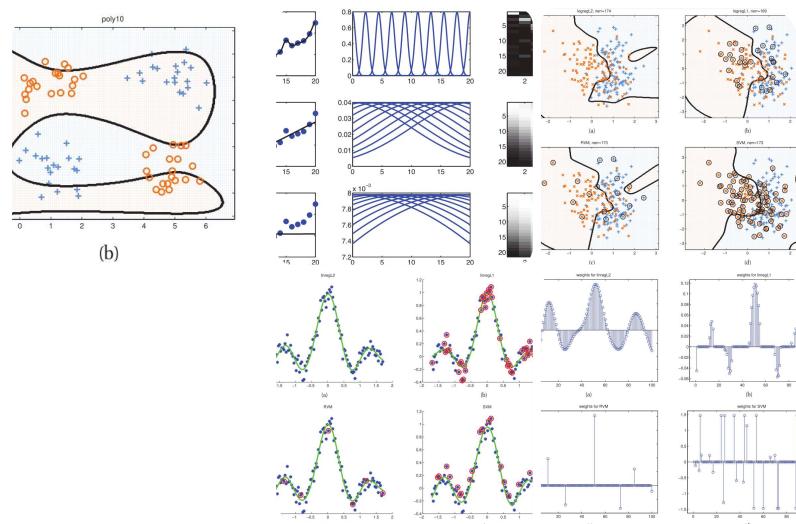
- 只拟合一个生成模型。
- 看每个样本  $x$  的“得分向量”  $g(x) = \nabla_{\theta} \log p(x | \theta)|_{\theta}$ : 它告诉我们“若只为让  $x$  更可能, 参数应往哪儿动”。

- 用  $g(x)^T F^{-1} g(x')$  比较两个样本 ( $F$  体现了模型曲率的“几何”)。
- 有理论联系：k-谱/字符串核等价于某些马尔可夫链的 Fisher 核；TF-IDF 内积核近似等价于某文本生成模型的 Fisher 核。

## 要点总结

- 核 = 相似度函数：**  $\kappa(x, x')$  给出对象间“像不像”的分数；很多算法只需成对相似度。
- 经典核：** 高斯/RBF（尺度可各向、ARD）、线性、多项式、Matérn、字符串核、金字塔匹配核等。
- TF-IDF + 余弦：** 相比裸词频更稳健地比较文档。
- Mercer/正定核很关键：** 保证存在隐式特征映射  $\phi$ ，从而“在高维里做内积而不显式展开”；高斯、余弦等是，sigmoid 核一般不是。
- 金字塔匹配核：** 多分辨率直方图交近似最优点对匹配，适合可变大小的图像特征集合。
- 生成模型造核：**
  - 概率乘积核：用“两个模型都认同的样本”来衡量相似，能得到 RBF 形式并适配 HMM 等。
  - Fisher 核：用得分向量与 Fisher 信息的几何来比相似，只需拟合一个模型，与字符串核、TF-IDF 内积核存在对应关系。

如果你想，我也可以把这些核在几个小示例（文本、图像点集、时间序列）里如何具体计算，做成一步步的小练习。



翻译并解释以下内容，翻译不能有遗漏，解释请通俗易懂且详尽，因为我的数学基础不太好，最后总结要点。

Figure 14.2 (a) xor truth table. (b) Fitting a linear logistic regression classifier using degree 10 polynomial expansion. (c) Same model, but using an RBF kernel with centroids specified by the 4 black crosses.

Figure 14.3 RBF basis in 1d. Left column: fitted function. Middle column: basis functions evaluated on a grid. Right column: design matrix. Top to bottom we show different bandwidths:  $\tau = 0.1, \tau = 0.5, \tau = 50$ .

Figure 14.4 Example of non-linear binary classification using an RBF kernel with bandwidth  $\sigma = 0.3$ . (a) L2VM with  $\lambda = 5$ . (b) L1VM with  $\lambda = 1$ . (c) RVM. (d) SVM with  $C = 1/\lambda$  chosen by cross validation. Black circles denote the support vectors.

Figure 14.5 Example of kernel based regression on the noisy sinc function using an RBF kernel with bandwidth  $\sigma = 0.3$ . (a) L2VM with  $\lambda = 0.5$ . (b) L1VM with  $\lambda = 0.5$ . (c) RVM. (d) SVM regression with  $C = 1/\lambda$  chosen by cross validation, and  $= 0.1$  (the default for SVMlight). Red circles denote the retained training exemplars.

Figure 14.6 Coefficient vectors of length  $N = 100$  for the models in Figure 14.6.

### 14.3 Using kernels inside GLMs

In this section, we discuss one simple way to use kernels for classification and regression. We will see other approaches later.

#### 14.3.1 Kernel machines

We define a kernel machine to be a GLM where the input feature vector has the form

$$\varphi(x) = [\kappa(x, \mu_1), \dots, \kappa(x, \mu_K)] \quad (14.28)$$

where  $\mu_k \in X$  are a set of  $K$  centroids. If  $\kappa$  is an RBF kernel, this is called an RBF network. We discuss ways to choose the  $\mu_k$  parameters below. We will call Equation 14.28 a kernelised feature vector. Note that in this approach, the kernel need not be a Mercer kernel. We can use the kernelized feature vector for logistic regression by defining  $p(y|x, \theta) = \text{Ber}(w^T \varphi(x))$ . This provides a simple way to define a non-linear decision boundary. As an example, consider the data coming from the exclusive or or xor function. This is a binary valued function of two binary

inputs. Its truth table is shown in Figure 14.2(a). In Figure 14.2(b), we have show some data labeled by the xor function, but we have jittered the points to make the picture clearer. We see we cannot separate the data even using a degree 10 polynomial.

However, using an RBF kernel and just 4 prototypes easily solves the problem as shown in Figure 14.2(c). We can also use the kernelized feature vector inside a linear regression model by defining  $p(y|x, \theta) = N(w^T \phi(x), \sigma^2)$ . For example, Figure 14.3 shows a 1d data set fit with  $K = 10$  uniformly spaced RBF prototypes, but with the bandwidth ranging from small to large. Small values lead to very wiggly functions, since the predicted function value will only be non-zero for points  $x$  that are close to one of the prototypes  $\mu_k$ . If the bandwidth is very large, the design matrix reduces to a constant matrix of 1's, since each point is equally close to every prototype; hence the corresponding function is just a straight line.

**14.3.2 L1VMs, RVMs, and other sparse vector machines**  
The main issue with kernel machines is: how do we choose the centroids  $\mu_k$ ? If the input is low-dimensional Euclidean space, we can uniformly tile the space occupied by the data with prototypes, as we did in Figure 14.2(c). However, this approach breaks down in higher numbers of dimensions because of the curse of dimensionality. If  $\mu_k \in RD$ , we can try to perform numerical optimization of these parameters (see e.g., (Haykin 1998)), or we can use MCMC inference, (see e.g., (Andrieu et al. 2001; Kohn et al. 2001)), but the resulting objective function / posterior is highly multimodal. Furthermore, these techniques is hard to extend to structured input spaces, where kernels are most useful. Another approach is to find clusters in the data and then to assign one prototype per cluster center (many clustering algorithms just need a similarity metric as input). However, the regions of space that have high density are not necessarily the ones where the prototypes are most useful for representing the output, that is, clustering is an unsupervised task that may not yield a representation that is useful for prediction. Furthermore, there is the need to pick the number of clusters. A simpler approach is to make each example  $x_i$  be a prototype, so we get

$$\phi(x) = [\kappa(x, x_1), \dots, \kappa(x, x_N)] \quad (14.29)$$

Now we see  $D = N$ , so we have as many parameters as data points. However, we can use any of the sparsity-promoting priors for  $w$  discussed in Chapter 13 to efficiently select a subset of the training exemplars. We call this a sparse vector machine. The most natural choice is to use  $L_1$  regularization (Krishnapuram et al. 2005).

(Note that in the multi-class case, it is necessary to use group lasso, since each exemplar is associated with  $C$  weights, one per class.) We call this L1VM, which stands for “ $L_1$ -regularized vector machine”. By analogy, we define the use of an  $L_2$  regularizer to be a L2VM or “ $L_2$ -regularized vector machine”; this of course will not be sparse.

We can get even greater sparsity by using ARD/SBL, resulting in a method called the relevance vector machine or RVM (Tipping 2001). One can fit this model using generic ARD/SBL algorithms, although in practice the most common method is the greedy algorithm in (Tipping and Faul 2003) (this is the algorithm implemented in Mike Tipping’s code, which is bundled with PMTK). Another very popular approach to creating a sparse kernel machine is to use a support vector machine or SVM. This will be discussed in detail in Section 14.5. Rather than using a sparsity-promoting prior, it essentially modifies the likelihood term, which is rather unnatural from a Bayesian point of view. Nevertheless, the effect is similar, as we will see. In Figure 14.4, we compare L2VM, L1VM, RVM and an SVM using the same RBF kernel on a binary classification problem in 2d. For simplicity,  $\lambda$  was chosen by hand for L2VM and L1VM; for RVMs, the parameters are estimated using empirical Bayes; and for the SVM, we use CV to pick  $C = 1/\lambda$ , since SVM performance is very sensitive to this parameter (see Section 14.5.3). We see that all the methods give similar performance. However, RVM is the sparsest (and hence fastest at test time), then L1VM, and then SVM. RVM is also the fastest to train, since CV for an SVM is slow. (This is despite the fact that the RVM code is in Matlab and the SVM code is in C.) This result is fairly typical. In Figure 14.5, we compare L2VM, L1VM, RVM and an SVM using an RBF kernel on a 1d regression problem. Again, we see that predictions are quite similar, but RVM is the sparsest, then L2VM, then SVM. This is further illustrated in Figure 14.6.

已思考 31s

下面先给出逐字逐句的准确中文翻译（含图注与公式编号），随后是通俗详尽的解释，最后给出要点总结。

---

## 逐句翻译

**图 14.2**

- (a) XOR 的真值表。
- (b) 用 10 次多项式展开来拟合一个线性的逻辑回归分类器。
- (c) 同样的模型，但使用一个 RBF 核，并把 4 个黑色叉号的位置作为中心 (centroids)。

**图 14.3** 1 维中的 RBF 基。左列：拟合得到的函数；中列：在一张网格上评估得到的各个基函数；右列：设计矩阵。自上而下展示不同带宽： $\tau = 0.1, \tau = 0.5, \tau = 50$ 。

**图 14.4** 使用带宽  $\sigma = 0.3$  的 RBF 核进行非线性二分类的示例。

- (a) L2VM,  $\lambda = 5$ 。
- (b) L1VM,  $\lambda = 1$ 。
- (c) RVM。
- (d) SVM (用交叉验证选  $C = 1/\lambda$ )。黑色圆圈表示支持向量。

**图 14.5** 在带噪 sinc 函数上使用带宽  $\sigma = 0.3$  的 RBF 核做基于核的回归示例。

- (a) L2VM,  $\lambda = 0.5$ 。
- (b) L1VM,  $\lambda = 0.5$ 。
- (c) RVM。
- (d) SVM 回归 (用交叉验证选  $C = 1/\lambda$ ，并取  $\varepsilon = 0.1$ ，这是 SVMlight 的默认值)。红色圆圈表示被保留的训练样本。

**图 14.6** 针对图 14.6 中各模型的长度为  $N = 100$  的系数向量。（注：原文此处很可能本应指“图 14.5 中的模型”。）

---

## 14.3 在广义线性模型 (GLM) 中使用核

在本节，我们讨论一种把核用于分类和回归的简单方法。稍后还会看到其他做法。

### 14.3.1 核机器 (Kernel machines)

我们把核机器定义为这样一种 GLM：其输入特征向量的形式为

$$\phi(x) = [\kappa(x, \mu_1), \dots, \kappa(x, \mu_K)] \quad (14.28)$$

其中  $\mu_k \in X$  是  $K$  个中心 (centroids) 构成的集合。若  $\kappa$  是 RBF 核，这就叫作 **RBF 网络**。下面将讨论如何选择  $\mu_k$ 。我们把式 (14.28) 称为**核化的特征向量**。注意在这种做法里，**核不必是 Mercer 核**。

我们可以把核化特征向量用于逻辑回归，定义

$$p(y | x, \theta) = \text{Ber}(w^\top \phi(x)).$$

这提供了一种**定义非线性决策边界**的简单方式。

举例来说，考虑来自“异或” (exclusive or, XOR) 函数的数据。这是一个对两个二进制输入的二值函数。它的真值表如图 14.2(a)。在图 14.2(b) 中，我们展示了一些按 XOR 标注的数据点，但为了图像清晰，对这些点做了轻微扰动。我们看到，即便使用 10 次多项式，也**无法把这些数据分开**。

然而，使用 RBF 核并只用 4 个原型点 (prototypes) 就能轻松解决该问题，如图 14.2(c) 所示。

我们也可以在线性回归模型中使用核化特征向量，定义

$$p(y | x, \theta) = N(w^\top \phi(x), \sigma^2).$$

例如，图 14.3 展示了一个 1 维数据集，用  $K = 10$  个**均匀放置**的 RBF 原型来拟合，但把带宽从小到大变化。带宽取很小时，会得到非常“抖”的函数，因为预测值只在  $x$  接近某个原型  $\mu_k$  时才不为零；若带宽非常大，设计矩阵会退化成元素全为 1 的常数矩阵，因为每个点到每个原型都“同样近”；因此对应的函数就是一条**直线**。

### 14.3.2 L1VM、RVM 以及其他稀疏向量机

核机器的主要问题是：**如何选择中心  $\mu_k$ ？**

若输入是低维欧氏空间，可以像图 14.2(c) 那样，用原型**均匀铺满**数据占据的空间。但是在高维里，这种方法会因**维度灾难**而失效。

若  $\mu_k \in \mathbb{R}^D$ ，可以尝试对这些参数做数值优化 (如 Haykin 1998)，或用 MCMC 推断 (如 Andrieu 等 2001; Kohn 等 2001)，但得到的目标函数/后验往往**高度多峰**。此外，这些技术很难推广到**结构化输入空间** (核在这类空间最有用)。

另一种办法是先对数据做**聚类**，每个簇中心放一个原型 (许多聚类算法只需一个相似度度量)。但是，空间中**密度高的**区域未必就是对表示输出最有用的区域；也就是说，聚类是**无监督的**，它学到的表示未必有利于**预测**。此外还面临如何选择簇数。

更简单的一招：让**每个训练样本  $x_i$  都当一个原型**，于是

$$\phi(x) = [\kappa(x, x_1), \dots, \kappa(x, x_N)] \quad (14.29)$$

这时看到  $D = N$ , 参数个数与样本数一样多。不过我们可以用第 13 章讨论的促稀疏先验/正则来高效地从训练样本中**挑一个子集**。我们把这种方法称为**稀疏向量机**。

最自然的是用  $\ell_1$  正则 (Krishnapuram 等 2005)。(注意在多分类时需要**组套索 (group lasso)**，因为每个样本对应  $C$  个权重，每类一个。) 我们把它称为 L1VM ( $\ell_1$ -regularized vector machine)。类似地，如果用  $\ell_2$  正则，就得到 L2VM ( $\ell_2$ -regularized vector machine)；当然它一般**不稀疏**。

用 ARD/SBL (自动相关性确定/稀疏贝叶斯学习) 可以得到更强的稀疏性，这就是**相关向量机 RVM** (Tipping 2001)。该模型可以用通用的 ARD/SBL 算法拟合，但实践中最常用的是 Tipping 与 Faul (2003) 的贪心算法 (PMTK 附带的 Tipping 代码实现了它)。

另一个非常流行的稀疏核机器是**支持向量机 SVM** (14.5 节详述)。它不是用促稀疏先验，而是**改造了似然/损失** (从贝叶斯视角看不太自然)，但效果类似。

在图 14.4 中，我们在一个二维二分类问题上，用**同一** RBF 核比较了 L2VM、L1VM、RVM 与 SVM。为简便起见，L2VM 与 L1VM 的  $\lambda$  手工设定；RVM 用经验贝叶斯估计参数；SVM 通过交叉验证选择  $C = 1/\lambda$ ，因为 SVM 性能对该参数很敏感 (见 14.5.3)。可以看到，各方法的表现**相近**。不过**稀疏度上**：RVM 最稀疏 (因而**测试时最快**)，其次 L1VM，再次 SVM。训练速度上，RVM 也更快，因为 SVM 的交叉验证较慢。(尽管 RVM 代码用 Matlab、SVM 代码用 C。) 这个结果相当常见。

在图 14.5 中，我们在一个 1 维回归任务上 (带噪 sinc) 用 RBF 核比较 L2VM、L1VM、RVM 与 SVM。再次看到，预测曲线很相似，但稀疏度次序为：RVM 最稀疏，其次 L2VM，再次 SVM。这一点在图 14.6 的系数向量中进一步得到体现。

## 通俗而详尽的解释

### 1) “核机器”的直觉

- 把每个中心  $\mu_k$  想成一只“传感器/探测器”。
- 传感器对“离它近”的输入  $x$  反应强 (RBF 核值大)，离得远反应弱。
- $\phi(x)$  就是所有传感器的读数；模型再对这些读数做线性加权  $w^\top \phi(x) \rightarrow$  就能画出**弯弯曲曲**的决策边界或函数曲线。

### 2) 带宽 ( $\tau$ 或 $\sigma$ ) 像“探测半径”

- 小带宽：每个传感器只“听得见”身边很近的点 → 曲线**很抖**，容易过拟合；设计矩阵接近**对角** (只对本地起作用)。
- 大带宽：每个传感器对所有点反应都差不多 → 设计矩阵几乎全 1 → 曲线接近**直线/常数**，容易欠拟合。

- 图 14.3 左中右三列正对应“拟合曲线 / 基函数形状 / 设计矩阵热度图”。

### 3) XOR 为什么线性/多项式难, RBF 容易?

- XOR 的四个角“同类不相邻、异类相邻”，任何**单条直线**都分不开。
- 多项式把原空间弯折，但在**有限阶数**下仍可能不够灵活（图 14.2(b)）。
- 用 RBF 只需在四个角各放一个原型，模型就能“局部发声”，轻松分开（图 14.2(c)）。

### 4) 怎么选中心 $\mu_k$

- 规则摆点（网格/均匀铺）：低维可行，高维爆炸。
- 优化或 MCMC：目标多峰，计算难，结构化输入更难。
- 聚类当中心：无监督，不一定对**预测最有利**，还要定簇数。
- 最简单**：直接把**每个训练样本**都当中心（式 14.29），再用稀疏技术自动挑出**少数关键样本**。

### 5) 谁更稀疏/更快? (图 14.4、14.5、14.6)

- L1VM:  $\ell_1$  正则会把很多权重压到 0 → 稀疏。
- RVM: 用 ARD/SBL 的贝叶斯做法，通常比  $\ell_1$  还稀疏（留下更少“相关向量”）。
- SVM: 通过“合页损失”只保留**支持向量**，也有稀疏性，但一般比 RVM、L1VM 稀疏度弱；而且需要调  $C$ ，常要交叉验证。
- L2VM:  $\ell_2$  正则不促稀疏，通常会保留许多小权重。
- 稀疏意味着：**预测阶段更快**（只对保留下的少数样本/中心计算核），也更易解释（哪些样本“在起作用”一目了然）。
- 图 14.6 的“系数杆状图”直观看到：RVM 留下的非零点最少，其次 L1VM（或 L2VM，视任务而定），SVM 次之或介于其间。

### 6) GLM 里用核，为什么不需要 Mercer?

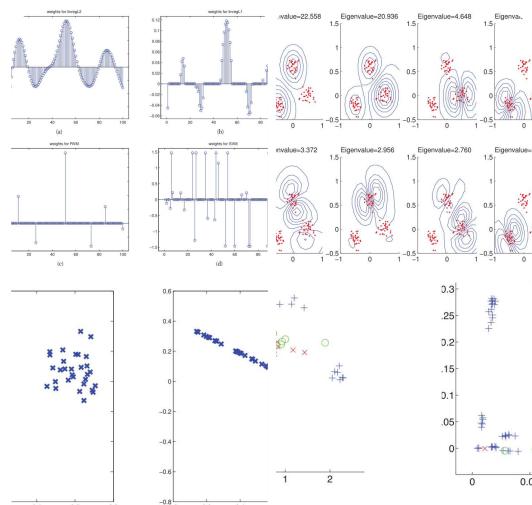
- 这里我们**显式构造**了一个**有限维**的特征向量  $\phi(x)$ （长度  $K$  或  $N$ ），在这一步中只把  $\kappa$  当作“相似度函数/基函数”。
- 只要能计算  $\kappa(x, \mu_k)$ ，就能训练线性/逻辑回归等 GLM；不涉及无限维隐式映射，因此**不强制要正定**（Mercer）。

## 要点总结

- 核机器 = GLM + 核化特征**:  $\phi(x) = [\kappa(x, \mu_k)]$ 。 $\mu_k$  是“中心/原型”，RBF 核下就是 RBF 网络。
- 带宽控制平滑度**: 小带宽 → 曲线抖、可能过拟合；大带宽 → 近似直线、可能欠拟合。

- **XOR 例子：**多项式特征也难分；RBF 只要 4 个原型就能分开。
- **选中心之痛点：**网格铺点高维失效；优化/MCMC 难；聚类未必有利于预测。
- **实用招数：**把每个训练样本当中心（式 14.29），再用**稀疏化**挑关键点 → 得到**稀疏向量机**。
- **几种方法对比：**
  - L2VM:  $\ell_2$  正则，不稀疏；
  - L1VM:  $\ell_1$  正则，较稀疏；
  - RVM: ARD/SBL，通常**最稀疏**、训练也快；
  - SVM: 靠损失函数产生稀疏（支持向量），性能相近但需调  $C$ 。
- **效果与速度：**分类/回归预测常相近；但稀疏度通常 RVM > L1VM(或 L2VM) > SVM，稀疏度高 → 预测更快、更易解释。

如果你愿意，我可以用一小段代码演示：给同一数据集，改变 RBF 带宽/ $\lambda$ ，画出对应的拟合曲线与非零系数位置，对比 L1VM、RVM、SVM 的“稀疏程度”。



翻译并解释以下内容，翻译不能有遗漏，解释请通俗易懂且详尽，因为我的数学基础不太好，最后总结要点。

Figure 14.6 Coefficient vectors of length  $N = 100$  for the models in Figure 14.6.

Figure 14.7 Visualization of the first 8 kernel principal component basis functions derived from some 2d data.

We use an RBF kernel with  $\sigma^2 = 0.1$ .

Figure 14.8 2d visualization of some 2d data. (a) PCA projection. (b) Kernel PCA projection.

Figure 14.9 2d representation of 12 dimensional oil flow

data. The different colors/symbols represent the 3 phases of oil flow. (a) PCA. (b) Kernel PCA with Gaussian kernel. Compare to Figure 15.10(b). From Figure 1 of (Lawrence 2005).

## 14.4 The kernel trick

Rather than defining our feature vector in terms of kernels,  $\phi(x) = [\kappa(x, x_1), \dots, \kappa(x, x_N)]$ , we can instead work with the original feature vectors  $x$ , but modify the algorithm so that it replaces all inner products of the form  $x, x$  with a call to the kernel function,  $\kappa(x, x)$ . This is called the kernel trick. It turns out that many algorithms can be kernelized in this way. We give some examples below. Note that we require that the kernel be a Mercer kernel for this trick to work.

### 14.4.1 Kernelized nearest neighbor classification

Recall that in a 1NN classifier (Section 1.4.2), we just need to compute the Euclidean distance of a test vector to all the training points, find the closest one, and look up its label. This can be kernelized by observing that

$$\|x_i - x_j\|^2 = x_i^T x_j + x_j^T x_i - 2 x_i^T x_j \quad (14.30)$$

This allows us to apply the nearest neighbor classifier to structured data objects.

### 14.4.2 Kernelized K-medoids clustering

K-means clustering (Section 11.4.2.5) uses Euclidean distance to measure dissimilarity, which is not always appropriate for structured objects. We now describe how to develop a kernelized version of the algorithm. The first step is to replace the K-means algorithm with the K-medoids algorithm. This is similar to K-means, but instead of representing each cluster's centroid by the mean of all data vectors assigned to this cluster, we make each centroid be one of the data vectors themselves. Thus we always deal with integer indexes, rather than data objects. We assign objects to their closest centroids as before. When we update the centroids, we look at each object that belongs to the cluster, and measure the sum of its distances to all the others in the same cluster; we then pick the one which has the smallest such sum:

$$m_k = \operatorname{argmin}_{i: z_i=k} \sum_{j: z_j=k} d(i, j) \quad (14.31)$$

where

$$d(i, j) = \|x_i - x_j\|^2 \quad (14.32)$$

This takes  $O(n^2 k)$  work per cluster, whereas K-means

takes  $O(nkD)$  to update each cluster. The pseudo-code is given in Algorithm 5. This method can be modified to derive a classifier, by computing the nearest medoid for each class. This is known as nearest medoid classification (Hastie et al. 2009, p671). This algorithm can be kernelized by using Equation 14.30 to replace the distance computation,  $d(i, i')$ .

#### 14.4.3 Kernelized ridge regression

Applying the kernel trick to distance-based methods was straightforward. It is not so obvious how to apply it to parametric models such as ridge regression. However, it can be done, as we now explain. This will serve as a good "warm up" for studying SVMs.

##### 14.4.3.1 The primal problem

Let  $x \in RD$  be some feature vector, and  $X$  be the corresponding  $N \times D$  design matrix. We want to minimize  $J(w) = (y - Xw)^T(y - Xw) + \lambda\|w\|^2$  (14.33)

The optimal solution is given by

$$w = (XTX + \lambda I)^{-1}XTy = (\sum_i x_i x_i^T + \lambda I)^{-1}XTy \quad (14.34)$$

##### 14.4.3.2 The dual problem

Equation 14.34 is not yet in the form of inner products.

However, using the matrix inversion lemma (Equation 4.107) we rewrite the ridge estimate as follows

$$w = XT(X^T X + \lambda I)^{-1}y \quad (14.35)$$

which takes  $O(N^3 + N^2D)$  time to compute. This can be advantageous if  $D$  is large. Furthermore, we see that we can partially kernelize this, by replacing  $X^T X$  with the Gram matrix  $K$ . But what about the leading  $X^T$  term? Let us define the following dual variables:

$$\alpha = (K + \lambda I)^{-1}y \quad (14.36)$$

Then we can rewrite the primal variables as follows

$$w = XT\alpha = \sum_i \alpha_i x_i \quad (14.37)$$

This tells us that the solution vector is just a linear sum of the  $N$  training vectors. When we plug this in at test time to compute the predictive mean, we get

$$\hat{f}(x) = w^T x = \sum_i \alpha_i x^T x = \sum_i \alpha_i k(x, x_i) \quad (14.38)$$

So we have successfully kernelized ridge regression by changing from primal to dual variables. This technique can be applied to many other linear models, such as logistic regression.

##### 14.4.3.3 Computational cost

The cost of computing the dual variables  $\alpha$  is  $O(N^3)$ , whereas the cost of computing the primal variables  $w$  is

$O(D^3)$ . Hence the kernel method can be useful in high dimensional settings, even if we only use a linear kernel (c.f., the SVD trick in Equation 7.44). However, prediction using the dual variables takes  $O(ND)$  time, while prediction using the primal variables only takes  $O(D)$  time. We can speedup prediction by making  $\alpha$  sparse, as we discuss in Section 14.5.

#### 14.4.4 Kernel PCA

In Section 12.2, we saw how we could compute a low-dimensional linear embedding of some data using PCA. This required finding the eigenvectors of the sample covariance matrix  $S = \frac{1}{N} \sum_{i=1}^N x_i x_i^T = \frac{1}{N} X X^T$ .

However, we can also compute PCA by finding the eigenvectors of the inner product matrix  $X X^T$ , as we show below. This will allow us to produce a nonlinear embedding, using the kernel trick, a method known as kernel PCA (Schoelkopf et al. 1998). First, let  $U$  be an orthogonal matrix containing the eigenvectors of  $X X^T$  with corresponding eigenvalues in  $\Lambda$ . By definition we have  $(X X^T)U = U\Lambda$ . Pre-multiplying by  $X^T$  gives

$$(X X^T)(X^T U) = (X^T U)\Lambda \quad (14.39)$$

from which we see that the eigenvectors of  $X X^T$  (and hence of  $S$ ) are  $V = X^T U$ , with eigenvalues given by  $\Lambda$  as before. However, these eigenvectors are not normalized, since  $\|v_j\|^2 = u_j^T X X^T u_j = \lambda_j u_j^T u_j = \lambda_j$ . So the normalized eigenvectors are given by  $V_{PCA} = X^T U \Lambda^{-1/2}$ . This is a useful trick for regular PCA if  $D > N$ , since  $X X^T$  has size  $D \times D$ , whereas  $X X^T$  has size  $N \times N$ . It will also allow us to use the kernel trick, as we now show. Now let  $K = X X^T$  be the Gram matrix. Recall from Mercer's theorem that the use of a kernel implies some underlying feature space, so we are implicitly replacing  $x_i$  with  $\varphi(x_i) = \varphi_i$ . Let  $\Phi$  be the corresponding (notional) design matrix, and  $S_\varphi = \frac{1}{N} \sum_{i=1}^N \varphi_i \varphi_i^T$  be the corresponding (notional) covariance matrix in feature space. The eigenvectors are given by  $V_{KPCA} = \Phi^T U \Lambda^{-1/2}$ , where  $U$  and  $\Lambda$  contain the eigenvectors and eigenvalues of  $K$ . Of course, we can't actually compute  $V_{KPCA}$ , since  $\varphi_i$  is potentially infinite dimensional.

However, we can compute the projection of a test vector  $x^*$  onto the feature space as follows:

$$\varphi^T * V_{KPCA} = \varphi^T * \Phi U \Lambda^{-1/2} = k^T * U \Lambda^{-1/2} \quad (14.40)$$

where  $k^* = [k(x^*, x_1), \dots, k(x^*, x_N)]$ .

There is one final detail to worry about. So far, we have

assumed the projected data has zero mean, which is not the case in general. We cannot simply subtract off the mean in feature space. However, there is a trick we can use. Define the centered feature vector as  $\tilde{\varphi}^T i = \varphi(x_i) - \frac{1}{N} \sum_{j=1}^N \varphi(x_j)$ . The Gram matrix of the centered feature vectors is given by

$$\tilde{K}_{ij} = \tilde{\varphi}^T i \tilde{\varphi}^T j \quad (14.41)$$

$$= \varphi^T i \varphi^T j - \frac{1}{N} \sum_{k=1}^N \varphi^T i \varphi^T k - \frac{1}{N} \sum_{k=1}^N \varphi^T j \varphi^T k + \frac{1}{N^2} \sum_{k=1}^N \sum_{l=1}^M \varphi^T k \varphi^T l \quad (14.42)$$

$$= \kappa(x_i, x_j) - \frac{1}{N} \sum_{k=1}^N \kappa(x_i, x_k) - \frac{1}{N} \sum_{k=1}^N \kappa(x_j, x_k) + \frac{1}{N^2} \sum_{k=1}^N \sum_{l=1}^M \kappa(x_k, x_l) \quad (14.43)$$

This can be expressed in matrix notation as follows:

$$\tilde{K} = H K H \quad (14.44)$$

where  $H = I - \frac{1}{N} J N J^T$  is the centering matrix.

Whereas linear PCA is limited to using  $L \leq D$  components, in kPCA, we can use up to  $N$  components, since the rank of  $\Phi$  is  $N \times D^*$ , where  $D^*$  is the (potentially infinite) dimensionality of embedded feature vectors. Figure 14.7 gives an example of the method applied to some  $D = 2$  dimensional data using an RBF kernel. We project points in the unit grid onto the first 8 components and visualize the corresponding surfaces using a contour plot. We see that the first two components separate the three clusters, and following components split the clusters. Although the features learned by kPCA can be useful for classification (Schoelkopf et al. 1998), they are not necessarily so useful for data visualization. For example, Figure 14.8 shows the projection of the data from Figure 14.7 onto the first 2 principal bases computed using PCA and kPCA. Obviously PCA perfectly represents the data. kPCA represents each cluster by a different line. Of course, there is no need to project 2d data back into 2d. So let us consider a different data set. We will use a 12 dimensional data set representing the three known phases of flow in an oil pipeline. (This data, which is widely used to compare data visualization methods, is synthetic, and comes from (Bishop and James 1993).) We project this into 2d using PCA and kPCA (with an RBF kernel). The results are shown in Figure 14.9. If we perform nearest neighbor classification in the low-dimensional space, kPCA makes 13 errors and PCA makes 20 (Lawrence 2005). Nevertheless, the kPCA projection is rather unnatural. In Section 15.5, we will discuss how to make kernelized

versions of probabilistic PCA. Note that there is a close connection between kernel PCA and a technique known as multidimensional scaling or MDS. This method finds a low-dimensional embedding such that Euclidean distance in the embedding space approximates the original dissimilarity matrix. See e.g., (Williams 2002) for details.

已思考 21s

下面先给出准确中文翻译（含图注与公式编号），随后是通俗详尽的解释，最后给出要点总结。

---

## 逐句翻译

**图 14.6** 针对图 14.6 中各模型的、长度为  $N = 100$  的系数向量。（注：原文此处很可能 是“针对图 14.5 的模型”，但我保持原文表述。）

**图 14.7** 由某些二维数据得到的前 8 个核主成分基函数的可视化。我们使用方差  $\sigma^2 = 0.1$  的 RBF 核。

**图 14.8** 对某些二维数据的二维可视化。(a) PCA 投影。(b) 核 PCA 投影。

**图 14.9** 12 维油流数据的二维表示。不同的颜色/符号表示油流的 3 个相位。(a) PCA。(b) 采用高斯核的核 PCA。与图 15.10(b) 对比。改绘自 Lawrence (2005) 图 1。

---

## 14.4 核技巧 (The kernel trick)

与其把特征向量定义为由核构成的

$$\phi(x) = [\kappa(x, x_1), \dots, \kappa(x, x_N)],$$

我们也可以直接在原始特征向量  $x$  上工作，但修改算法：把所有形如  $\langle x, x' \rangle$  的内积统统替换为一次核函数调用  $\kappa(x, x')$ 。这称为核技巧。事实证明许多算法都可这样“核化”。下面给出若干例子。注意：为了该技巧有效，我们需要核是 Mercer (正定) 核。

### 14.4.1 核化的一近邻分类 (1NN)

回忆 1NN 分类器 (1.4.2 节)：对测试向量与全部训练点计算欧氏距离，找最近者并读取其标签。可利用恒等式

$$\|x_i - x'_i\|_2^2 = \langle x_i, x_i \rangle + \langle x'_i, x'_i \rangle - 2\langle x_i, x'_i \rangle \quad (14.30)$$

把三个内积都换成核值，即可把最近邻分类应用到结构化对象上。

### 14.4.2 核化的 K-medoids 聚类

K-means (11.4.2.5 节) 用欧氏距离衡量不相似性，对结构化对象往往不合适。我们先把 K-means 换成 K-medoids：与 K-means 类似，但每个簇的“中心”不是均值，而是该簇中某个样本本身。这样始终只处理样本索引而非对象。分配步骤同前；更新中心时，在簇内对候选样本  $i$  计算它与同簇所有其它样本距离之和，并选择和最小者：

$$m_k = \arg \min_{i:z_i=k} \sum_{i':z_{i'}=k} d(i, i') \quad (14.31)$$

其中

$$d(i, i') = \|x_i - x_{i'}\|_2^2. \quad (14.32)$$

该更新每个簇需  $O(n_k^2)$  工作量，而 K-means 更新每簇是  $O(n_k D)$ 。伪代码见算法 5。本方法还能改成分类器：对每个类别求其最近 medoid，称为最近 medoid 分类 (Hastie 等, 2009, 第 671 页)。利用式 (14.30) 把距离换成核计算即可“核化”该算法。

### 14.4.3 核化的岭回归 (ridge regression)

把核技巧用于基于距离的方法很直接；用于参数模型（如岭回归）不那么明显，但也可以，如下所示——这也是学习 SVM 的“热身”。

#### 14.4.3.1 原始问题 (primal)

设  $x \in \mathbb{R}^D$  为特征向量， $X$  为  $N \times D$  的设计矩阵。最小化

$$J(w) = (y - Xw)^T (y - Xw) + \lambda \|w\|^2. \quad (14.33)$$

最优解

$$w = (X^T X + \lambda I_D)^{-1} X^T y = \left( \sum_i x_i x_i^T + \lambda I_D \right)^{-1} X^T y. \quad (14.34)$$

#### 14.4.3.2 对偶问题 (dual)

式 (14.34) 还没写成“内积形式”。用矩阵求逆引理 (式 4.107) 可改写为

$$w = X^T (X X^T + \lambda I_N)^{-1} y, \quad (14.35)$$

计算量  $O(N^3 + N^2 D)$ ，当  $D$  大时有利。进一步，把  $X X^T$  换成 Gram 矩阵  $K$  可部分核化。但前面的  $X^T$  怎么办？定义对偶变量

$$\alpha = (K + \lambda I_N)^{-1} y, \quad (14.36)$$

则原始变量

$N$ 

$$w = X^T \alpha = \sum_{i=1}^N \alpha_i x_i. \quad (14.37)$$

也就是说，解向量是训练样本的线性组合。测试时的预测均值

$$\hat{f}(x) = w^T x = \sum_{i=1}^N \alpha_i \langle x_i, x \rangle = \sum_{i=1}^N \alpha_i \kappa(x, x_i). \quad (14.38)$$

这样通过从**原始变量转为对偶变量**，我们把岭回归成功“核化”了。该技巧也可用于很多线性模型，如逻辑回归。

#### 14.4.3.3 计算代价

求  $\alpha$  的代价是  $O(N^3)$ ，而求  $w$  是  $O(D^3)$ 。因此即便只用**线性核**，在高维场景下对偶/核方法也很有用（参见 7.44 节的 SVD 技巧）。不过，用对偶变量预测每次需  $O(ND)$ ，而用原始变量只需  $O(D)$ 。若让  $\alpha$  稀疏（见 14.5 节），可显著加速预测。

#### 14.4.4 核 PCA (kPCA)

12.2 节中我们用 PCA 求低维**线性**嵌入，需要求协方差矩阵

$$S = \frac{1}{N} \sum_{i=1}^N x_i x_i^T = (1/N) X^T X$$

的特征向量。不过，也可通过求**内积矩阵**  $XX^T$  的特征向量来做 PCA，这将允许我们用核技巧得到**非线性**嵌入——即**核 PCA** (Schölkopf 等, 1998)。

令  $U$  为  $XX^T$  的特征向量矩阵，对应特征值在  $\Lambda$  中。按定义

$$(XX^T)U = U\Lambda.$$

左乘  $X^T$  得

$$(X^T X)(X^T U) = (X^T U)\Lambda, \quad (14.39)$$

可见  $X^T X$  (从而  $S$ ) 的特征向量是  $V = X^T U$ ，特征值仍为  $\Lambda$ 。但这些向量未归一化，

因为

$$\|v_j\|^2 = u_j^T X X^T u_j = \lambda_j u_j^T u_j = \lambda_j.$$

归一化后为

$$V_{\text{pca}} = X^T U \Lambda^{-1/2}.$$

当  $D > N$  时这对常规 PCA 很有用，因为  $X^T X$  是  $D \times D$ ，而  $XX^T$  仅  $N \times N$ 。这也让我们可用核技巧：

令  $K = XX^T$  为 Gram 矩阵。由 Mercer 定理，使用核意味着存在某个底层特征空间；用  $\phi(x_i)$  取代  $x_i$ ，记  $\phi_i$ 。令  $\Phi$  为相应（概念上的）设计矩阵， $S_\phi = \frac{1}{N} \sum_i \phi_i \phi_i^T$  为特征空间中的协方差矩阵。核 PCA 的“特征向量”为

$$V_{\text{kPCA}} = \Phi^T U \Lambda^{-1/2},$$

其中  $U, \Lambda$  来自  $K$  的特征分解。当然我们无法显式算  $V_{\text{kPCA}}$ ，因为  $\phi_i$  可能是无限维。不过，测试点  $x_{\backslash *}$  在该空间的投影可算为

$$\phi_{\backslash *}^T V_{\text{kPCA}} = \phi_{\backslash *}^T \Phi U \Lambda^{-1/2} = k_{\backslash *}^T U \Lambda^{-1/2}, \quad (14.40)$$

其中  $k_{\backslash *} = [\kappa(x_{\backslash *}, x_1), \dots, \kappa(x_{\backslash *}, x_N)]$ 。

还有一个细节：到目前为止我们默认投影后的数据零均值，而一般并非如此。不能直接在特征空间里减均值，但可用技巧：把居中后的特征向量定义为

$\tilde{\phi}_i = \phi(x_i) - \frac{1}{N} \sum_{j=1}^N \phi(x_j)$ 。它们的 Gram 矩阵为

$$\tilde{K}_{ij} = \tilde{\phi}_i^T \tilde{\phi}_j \quad (14.41)$$

$$= \kappa(x_i, x_j) - \frac{1}{N} \sum_k \kappa(x_i, x_k) - \frac{1}{N} \sum_k \kappa(x_j, x_k) + \frac{1}{N^2} \sum_k \sum_{\ell} \kappa(x_k, x_{\ell}) \quad (14.43)$$

记号形式：

$$\tilde{K} = HKH, \quad H = I - \frac{1}{N} \mathbf{1}_N \mathbf{1}_N^T. \quad (14.44)$$

线性 PCA 的主成分数  $L \leq D$ ；而在 kPCA 中，可用到  $N$  个分量，因为  $\Phi$  的秩是  $N \times D^{1*}$  ( $D^{1*}$  是嵌入特征的维数，可能无限)。

图 14.7：用 RBF 核把二维数据做 kPCA，把单位网格上的点投到前 8 个分量，再用等高线显示对应“表面”。前两个分量把 3 个簇分开，后续分量在簇内进一步细分。

kPCA 学到的特征有时有利于分类 (Schölkopf 等, 1998)，但对可视化未必理想。图 14.8：把图 14.7 的数据投到 PCA 与 kPCA 的前 2 个主轴。PCA 能“完美”表示数据；kPCA 则把每个簇拉成不同的直线。

再看不同数据集：12 维的油管三相流 (Bishop & James, 1993，常用于可视化方法比较)。用 PCA 与 RBF-kPCA 投到二维 (图 14.9)。若在低维空间里做最近邻分类，kPCA 出错 13 次、PCA 出错 20 次 (Lawrence, 2005)；不过 kPCA 的投影看起来较“不自然”。第 15.5 节会讨论概率 PCA 的核化版本。kPCA 与\*\*多维尺度法 (MDS)\*\*关系密切：MDS 在低维中寻找欧氏距离尽量近似原始“不相似度矩阵”的嵌入 (见 Williams, 2002)。

## 通俗而详尽的解释

### 1) “核技巧”到底是什么?

把“在某个（也许无限维）特征空间做线性算法”这件事，等价地写成只用**内积**的形式；再把内积换成**核函数**。你不用显式造出高维特征  $\phi(x)$ ，只要能算  $\kappa(x, x')$  就行。前提：核得是**正定的** (Mercer)，这样得到的 Gram 矩阵才“像内积”。

### 2) 为什么 1NN、K-medoids 也能核化?

因为它们只用到了**距离**，而距离平方可以由**三个内积组合成式** (14.30)。把这些内积换成核值，就能在“字符串、树、图像片段集合”等结构化对象上做最近邻或 medoids 聚类/分类。

### 3) 岭回归的“对偶视角”= 代表定理的味道

推到式 (14.37) 后会发现：最优的  $w$  总能写成训练样本的线性组合。这意味着预测  $\hat{f}(x) = \sum_i \alpha_i \kappa(x, x_i)$ ；也就是说，**只需核值**，不用  $w$  或  $\phi(x)$ 。这与很多核方法的“代表定理”直觉一致：解落在训练点张成的子空间里。

### 4) 计算量的权衡

- 训练：原始 (primal) 解  $O(D^3)$ ；对偶 (dual) 解  $O(N^3)$ 。高维  $D$  大时，对偶更划算。
- 预测：对偶每次要与**所有训练点**做核计算 ( $O(N)$  次内积)，而原始只要  $O(D)$ 。
- 因此需要**稀疏化** (让很多  $\alpha_i = 0$ ，只保留少数“支持/相关向量”) 来提速——这也是 L1VM/RVM/SVM 的意义。

### 5) 核 PCA 的直觉

- 常规 PCA：找能最大化方差的线性方向。
- 核 PCA：先把数据**隐式映射**到高维再做 PCA，于是得到**非线性的**低维表示。
- 实现关键：把所有对  $\phi(x)$  的操作改写为对 Gram 矩阵  $K$  的操作，并且要先把  $K$  居中 (式 14.44)。
- 效果：对某些数据，kPCA 能更好地分离类簇、提升下游最近邻分类；但投影形态可能“不直观”。

### 6) 图注里的信息在说啥?

- **图 14.6：**不同方法的权重/系数图能直观看到谁更**稀疏** (RVM 往往最稀疏，其次 L1，再是 SVM/L2)。
- **图 14.7–14.9：**展示 kPCA 的基函数形状、与 PCA 的可视化差异、以及在油流数据上的分类误差比较。

## 要点总结

- **核技巧：**把算法改写成只依赖内积，再以  $\kappa(x, x')$  替换内积；要求核为Mercer。

- **可核化的对象**: 1NN、K-medoids、岭回归（及其它线性模型）、PCA → kPCA。
- **岭回归对偶化**:  $\alpha = (K + \lambda I)^{-1}y$ ,  $\hat{f}(x) = \sum_i \alpha_i \kappa(x, x_i)$ 。训练在高维更省，预测需稀疏化提速。
- **kPCA 实现要点**: 用  $K$  的特征分解；测试点投影  $k_{\text{test}}^T U \Lambda^{-1/2}$ ；居中  $K$  用  $\tilde{K} = HKH^T$ 。
- **效果**: kPCA 能提供非线性嵌入，常利于分类，但可视化形态不一定“自然”。
- **图示信息**: RVM/L1VM/SVM 等方法在系数稀疏度与投影表现上的不同，体现了核方法在“表达力—计算—可解释性”之间的权衡。

这一小节主要在讲什么？我不太懂。把核化用到这些方法上优势是什么

已思考 17s

好问题！这小节（14.4 一整段）讲的是\*\*“核技巧（kernel trick）”：

**把很多只用到内积或距离的算法，改写成不直接看原始特征、而是把内积  $\langle x, x' \rangle$  全部换成核函数  $\kappa(x, x')$**  来算。这样等价于在一个（可能无限维的）特征空间里做线性方法，但我们**不用真的构造那些高维特征**。

具体他们示范了 4 个“核化”的例子：

1. **一近邻 (1NN)**：用恒等式  $\|x - x'\|^2 = \langle x, x \rangle + \langle x', x' \rangle - 2\langle x, x' \rangle$ ，把三个内积换成核值，就能对字符串/树/图像集合等**结构化对象**做最近邻。
2. **K-medoids 聚类**：更新簇中心要用样本间距离，仍通过上式把距离写成核值，从而在**结构化对象**上做聚类，并可做“最近 medoid 分类”。
3. **岭回归**：把原始解（用  $X^T X$ ）换成**对偶解**，得到  $\hat{f}(x) = \sum_i \alpha_i \kappa(x, x_i)$ 。这就是把线性回归“核化”的标准套路（很多线性模型都能这样做）。
4. **PCA→核 PCA**：把协方差的特征分解换成 Gram 矩阵  $K$  的特征分解（并先居中  $K$ ），就得到**非线性的低维嵌入**。文中图示对比了 PCA 与 kPCA 的不同效果。

## 核化带来的优势（为啥要这么干）

### 1) 自动获得非线性表达力

- 用核替换内积，相当于在高维（甚至无限维）做线性方法  $\Rightarrow$  在原空间就是**非线性的**决策边界/回归函数/降维映射。

- 例：线性或有限阶多项式分不开的 XOR，用 RBF 核轻松分开。

## 2) 直接处理“结构化”或“可变长”的数据

- 只要能定义一个合适的相似度/核，就能把最近邻、聚类、回归、降维等方法用在字符串、树、图像特征集合、序列等上面，而不必先把它们硬凑成固定长度向量。

## 3) 免去显式造高维特征

- 不用真的构造  $\phi(x)$  (可能无限维)，只要算核值  $\kappa(x, x')$ ；既省事又稳定。
- 还能把领域知识直接放进核：比如编辑距离变核、图像的金字塔匹配核等。

## 4) 在高维 ( $D$ 很大) 时训练更划算

- 岭回归等线性模型的对偶形式把训练复杂度从  $O(D^3)$  变成  $O(N^3)$ 。当  $D \gg N$  时更有利（文本、基因表达等）。
- 即便只用线性核，也可用对偶解省计算。

## 5) 一套统一的“插件式”改造法

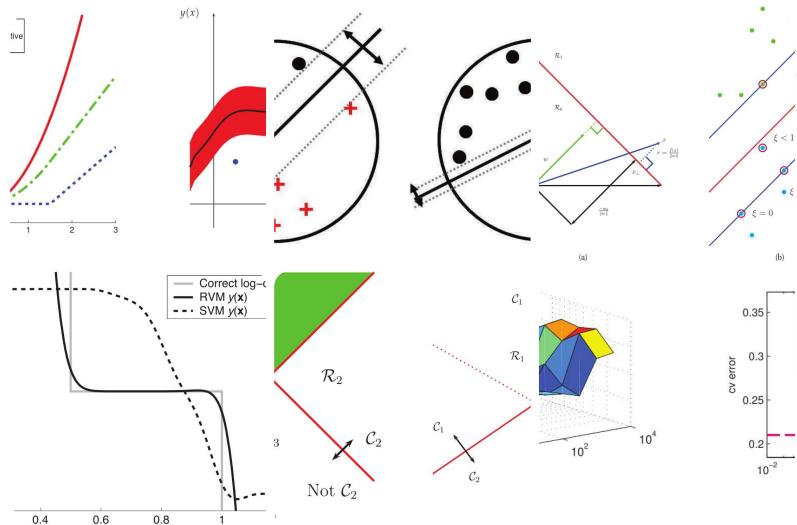
- 许多只依赖内积/距离的算法都能“一键核化”：最近邻、K-medoids、岭回归、逻辑回归（对偶）、PCA 等，形成统一范式，工程上好复用。

## 但也要知道的取舍

- 核必须正定 (Mercer)**：否则“像内积”的数学性质不保。
- 核与超参需要调**（如 RBF 的带宽  $\sigma$ ），不当会过/欠拟合。
- 规模瓶颈**：核方法通常需要存  $N \times N$  的 Gram 矩阵（内存  $O(N^2)$ ，训练  $O(N^3)$ ）。
- 预测代价**：对偶形式预测要与所有训练点算核 ( $O(N)$ )。常借助 **稀疏化**（SVM 支持向量、RVM/ $\ell_1$  只留少数样本）来提速。
- 可视化未必更“直观”**：kPCA 的嵌入对分类可能更好，但形状不一定像 PCA 那样易读。

## 一句话版

这小节教你一个通用法：**把算法里的内积换成核函数**，就能在不造高维特征的情况下，让传统的线性/距离方法获得**非线性能力**，并能处理**结构化数据**；代价是要选好核与超参，且计算通常随样本数  $N$  呈二/三次方增长，需要用稀疏化或近似来规模化。



翻译并解释以下内容，翻译不能有遗漏，解释请通俗易懂且详尽，因为我的数学基础不太好，最后总结要点。

Figure 14.10 (a) Illustration of Huber and  $\ell_1$ -insensitive loss functions, where  $\kappa=1.5$ . Figure generated by huberLossDemo. (b) Illustration of the  $\ell_1$ -tube used in SVM regression. Points above the tube have  $\xi_i > 0$  and  $\xi_{*i} = 0$ . Points below the tube have  $\xi_i = 0$  and  $\xi_{*i} > 0$ . Points inside the tube have  $\xi_i = \xi_{*i} = 0$ . Based on Figure 7.7 of (Bishop 2006a).

Figure 14.11 Illustration of the large margin principle. Left: a separating hyper-plane with large margin. Right: a separating hyper-plane with small margin.

Figure 14.12 (a) Illustration of the geometry of a linear decision boundary in 2d. A point  $x$  is classified as belonging in decision region  $R_1$  if  $f(x) > 0$ , otherwise it belongs in decision region  $R_2$ ; here  $f(x)$  is known as a discriminant function. The decision boundary is the set of points such that  $f(x)=0$ .  $w$  is a vector which is perpendicular to the decision boundary. The term  $w_0$  controls the distance of the decision boundary from the origin. The signed distance of  $x$  from its orthogonal projection onto the decision boundary,  $x^\perp$ , is given by  $f(x)/\|w\|$ . Based on Figure 4.1 of (Bishop 2006a). (b) Illustration of the soft margin principle. Points with circles around them are support vectors. We also indicate the value of the corresponding slack variables. Based on Figure 7.3 of (Bishop 2006a).

Figure 14.13 Log-odds vs x for 3 different methods. Based on Figure 10 of (Tipping 2001).

Figure 14.14 (a) The one-versus-rest approach. The green region is predicted to be both class 1 and class 2. (b) The one-versus-one approach. The label of the green region is ambiguous. Based on Figure 4.2 of (Bishop 2006a).

Figure 14.15 (a) A cross validation estimate of the 0-1 error for an SVM classifier with RBF kernel with different precisions  $\gamma = 1/(2\sigma^2)$  and different regularizer  $\lambda = 1/C$ , applied to a synthetic data set drawn from a mixture of 2 Gaussians. (b) A slice through this surface for  $\gamma = 5$ . The red dotted line is the Bayes optimal error, computed using Bayes rule applied to the model used to generate the data. Based on Figure 12.6 of (Hastie et al. 2009). Figure generated by `svmCgammaDemo`.

## 14.5 Support vector machines (SVMs)

In Section 14.3.2, we saw one way to derive a sparse kernel machine, namely by using a GLM with kernel basis functions, plus a sparsity-promoting prior such as L1 or ARD. An alternative approach is to change the objective function from negative log likelihood to some other loss function, as we discussed in Section 6.5.5. In particular, consider the L2 regularized empirical risk function

$$J(w, \lambda) = N \sum_{i=1}^N L(y_i, \hat{y}_i) + \lambda \|w\|^2 \quad (14.45)$$

where  $\hat{y}_i = w^T x_i + w_0$ . (So far this is in the original feature space; we introduce kernels in a moment.) If  $L$  is quadratic loss, this is equivalent to ridge regression, and if  $L$  is the log-loss defined in Equation 6.73, this is equivalent to logistic regression. In the ridge regression case, we know that the solution to this has the form  $\hat{w} = (X^T X + \lambda I)^{-1} X^T y$ , and plug-in predictions take the form  $\hat{w}_0 + \hat{w}^T x$ . As we saw in Section 14.4.3, we can rewrite these equations in a way that only involves inner products of the form  $x^T x$ , which we can replace by calls to a kernel function,  $\kappa(x, x')$ . This is kernelized, but not sparse. However, if we replace the quadratic/ log-loss with some other loss function, to be explained below, we can ensure that the solution is sparse, so that predictions only depend on a subset of the training data, known as support vectors. This combination of the kernel trick plus a modified loss function is known as a support vector machine or SVM. This technique was originally designed for binary classification, but can be extended to regression and multi-class classification as we explain below. Note that SVMs are very unnatural from a

probabilistic point of view. First, they encode sparsity in the loss function rather than the prior. Second, they encode kernels by using an algorithmic trick, rather than being an explicit part of the model. Finally, SVMs do not result in probabilistic outputs, which causes various difficulties, especially in the multi-class classification setting (see Section 14.5.2.4 for details). It is possible to obtain sparse, probabilistic, multi-class kernel-based classifiers, which work as well or better than SVMs, using techniques such as the L1VM or RVM, discussed in Section 14.3.2. However, we include a discussion of SVMs, despite their non-probabilistic nature, for two main reasons. First, they are very popular and widely used, so all students of machine learning should know about them. Second, they have some computational advantages over probabilistic methods in the structured output case; see Section 19.7.

#### 14.5.1 SVMs for regression

The problem with kernelized ridge regression is that the solution vector  $w$  depends on all the training inputs. We now seek a method to produce a sparse estimate. Vapnik (Vapnik et al. 1997) proposed a variant of the Huber loss function (Section 7.4) called the epsilon insensitive loss function, defined by

$$L(y, \hat{y}) = 0 \text{ if } |y - \hat{y}| < \epsilon \text{ otherwise} \quad (14.46)$$

This means that any point lying inside an  $\epsilon$ -tube around the prediction is not penalized, as in Figure 14.10. The corresponding objective function is usually written in the following form

$$J = C \sum_{i=1}^n L(y_i, \hat{y}_i) + \frac{1}{2} \|w\|^2 \quad (14.47)$$

where  $\hat{y}_i = f(x_i) = w^T x_i + w_0$  and  $C = 1/\lambda$  is a regularization constant. This objective is convex and unconstrained, but not differentiable, because of the absolute value function in the loss term. As in Section 13.4, where we discussed the lasso problem, there are several possible algorithms we could use. One popular approach is to formulate the problem as a constrained optimization problem. In particular, we introduce slack variables to represent the degree to which each point lies outside the tube:

$$y_i \leq f(x_i) + \xi_i \quad (14.48)$$

$$y_i \geq f(x_i) - \xi_i \quad (14.49)$$

Given this, we can rewrite the objective as follows:

$$J = C \sum_{i=1}^n (\xi_i + \xi_i^-) + \frac{1}{2} \|w\|^2 \quad (14.50)$$

quadratic function of  $w$ , and must be minimized subject to the linear constraints in Equations 14.48-14.49, as well as the positivity constraints  $\xi_+ - i \geq 0$  and  $\xi_- - i \geq 0$ . This is a standard quadratic program in  $2N + D + 1$  variables. One can show (see e.g., (Schoelkopf and Smola 2002)) that the optimal solution has the form

$$\hat{w} = i \alpha_i x_i \quad (14.51)$$

where  $\alpha_i \geq 0$ . Furthermore, it turns out that the  $\alpha$  vector is sparse, because we don't care about errors which are smaller than  $\epsilon$ . The  $x_i$  for which  $\alpha_i > 0$  are called the support vectors; these are points for which the errors lie on or outside the tube. Once the model is trained, we can then make predictions using

$$\hat{y}(x) = \hat{w}_0 + \hat{w}^T x \quad (14.52)$$

Plugging in the definition of  $\hat{w}$  we get

$$\hat{y}(x) = \hat{w}_0 + i \alpha_i x_i^T x \quad (14.53)$$

Finally, we can replace  $x^T x$  with  $\kappa(x_i, x)$  to get a kernelized solution:

$$\hat{y}(x) = \hat{w}_0 + i \alpha_i \kappa(x_i, x) \quad (14.54)$$

#### 14.5.2 SVMs for classification

We now discuss how to apply SVMs to classification. We first focus on the binary case, and then discuss the multi-class case in Section 14.5.2.4.

##### 14.5.2.1 Hinge loss

In Section 6.5.5, we showed that the negative log likelihood of a logistic regression model,

$$L_{\text{NLL}}(y, \eta) = -\log p(y|x, w) = \log(1 + e^{-y\eta}) \quad (14.55)$$

was a convex upper bound on the 0-1 risk of a binary classifier, where  $\eta = f(x) = w^T x + w_0$  is the log odds ratio, and we have assumed the labels are  $y \in \{1, -1\}$  rather than  $\{0, 1\}$ . In this section, we replace the NLL loss with the hinge loss, defined as

$$L_{\text{hinge}}(y, \eta) = \max(0, 1 - y\eta) = (1 - y\eta)_+ \quad (14.56)$$

Here  $\eta = f(x)$  is our "confidence" in choosing label  $y = 1$ ; however, it need not have any probabilistic semantics. See Figure 6.7 for a plot. We see that the function looks like a door hinge, hence its name. The overall objective has the form

$$\min_w w_0 \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N (1 - y_i f(x_i))_+ \quad (14.57)$$

Once again, this is non-differentiable, because of the max term. However, by introducing slack variables  $\xi_i$ , one can show that this is equivalent to solving

$$\min_w w_0, \xi \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \xi_i \text{ s.t. } \xi_i \geq 0, y_i(x^T w + w_0) \geq$$

$$1 - \xi_i, i = 1: N \quad (14.58)$$

This is a quadratic program in  $N + D + 1$  variables, subject to  $O(N)$  constraints. We can eliminate the primal variables  $w$ ,  $w_0$  and  $\xi_i$ , and just solve the  $N$  dual variables, which correspond to the Lagrange multipliers for the constraints. Standard solvers take  $O(N^3)$  time. However, specialized algorithms, which avoid the use of generic QP solvers, have been developed for this problem, such as the sequential minimal optimization or SMO algorithm (Platt 1998). In practice this can take  $O(N^2)$ . However, even this can be too slow if  $N$  is large. In such settings, it is common to use linear SVMs, which take  $O(N)$  time to train (Joachims 2006; Bottou et al. 2007).

One can show that the solution has the form

$$\hat{w} = \sum_i \alpha_i x_i \quad (14.59)$$

where  $\alpha_i = \lambda_i y_i$  and where  $\alpha$  is sparse (because of the hinge loss). The  $x_i$  for which  $\alpha_i > 0$  are called support vectors; these are points which are either incorrectly classified, or are classified correctly but are on or inside the margin (we discuss margins below). See Figure 14.12(b) for an illustration. At test time, prediction is done using

$$\hat{y}(x) = \text{sgn}(f(x)) = \text{sgn}(\hat{w}^T x + b) \quad (14.60)$$

Using Equation 14.59 and the kernel trick we have

$$\hat{y}(x) = \text{sgn}(\hat{w}^T x + \sum_i \alpha_i k(x_i, x)) \quad (14.61)$$

This takes  $O(sD)$  time to compute, where  $s \leq N$  is the number of support vectors. This depends on the sparsity level, and hence on the regularizer  $C$ .

#### 14.5.2.2 The large margin principle

In this section, we derive Equation 14.58 from a completely different perspective. Recall that our goal is to derive a discriminant function  $f(x)$  which will be linear in the feature space implied by the choice of kernel.

Consider a point  $x$  in this induced space. Referring to Figure 14.12(a), we see that

$$x = x_\perp + r w / \|w\| \quad (14.62)$$

where  $r$  is the distance of  $x$  from the decision boundary whose normal vector is  $w$ , and  $x_\perp$  is the orthogonal projection of  $x$  onto this boundary. Hence

$$f(x) = w^T x + w_0 = (w^T x_\perp + w_0) + r w^T w / \|w\|^2 \quad (14.63)$$

Now  $f(x_\perp) = 0$  so  $0 = w^T x_\perp + w_0$ . Hence  $f(x) = r w^T w / \|w\|^2$ , and  $r = f(x) / \|w\|$ . We would like to make this distance  $r = f(x) / \|w\|$  as large as possible, for reasons illustrated in Figure 14.11. In particular, there might be many lines that

perfectly separate the training data (especially if we work in a high dimensional feature space), but intuitively, the best one to pick is the one that maximizes the margin, i.e., the perpendicular distance to the closest point. In addition, we want to ensure each point is on the correct side of the boundary, hence we want  $f(x_i)y_i > 0$ . So our objective becomes

$$\max_{w,w_0} N \min_{i=1}^N y_i(w^T x_i + w_0) \|w\| \quad (14.64)$$

Note that by rescaling the parameters using  $w \rightarrow kw$  and  $w_0 \rightarrow kw_0$ , we do not change the distance of any point to the boundary, since the  $k$  factor cancels out when we divide by  $\|w\|$ . Therefore let us define the scale factor such that  $y_i = 1$  for the point that is closest to the decision boundary. We therefore want to optimize

$$\min_{w,w_0} 1/2 \|w\|^2 \text{ s.t. } y_i(w^T x_i + w_0) \geq 1, i=1:N \quad (14.65)$$

(The fact of  $1/2$  is added for convenience and doesn't affect the optimal parameters.) The constraint says that we want all points to be on the correct side of the decision boundary with a margin of at least 1. For this reason, we say that an SVM is an example of a large margin classifier. If the data is not linearly separable (even after using the kernel trick), there will be no feasible solution in which  $y_i \geq 1$  for all  $i$ . We therefore introduce slack variables  $\xi_i \geq 0$  such that  $\xi_i = 0$  if the point is on or inside the correct margin boundary, and  $\xi_i = |y_i - f_i|$  otherwise. If  $0 < \xi_i \leq 1$  the point lies inside the margin, but on the correct side of the decision boundary. If  $\xi_i > 1$ , the point lies on the wrong side of the decision boundary. See Figure 14.12(b). We replace the hard constraints that  $y_i \geq 0$  with the soft margin constraints that  $y_i \geq 1 - \xi_i$ . The new objective becomes

$$\min_{w,w_0,\xi} 1/2 \|w\|^2 + C \sum_{i=1}^N \xi_i \text{ s.t. } \xi_i \geq 0, y_i(w^T x_i + w_0) \geq 1 - \xi_i \quad (14.66)$$

which is the same as Equation 14.58. Since  $\xi_i > 1$  means point  $i$  is misclassified, we can interpret  $\xi_i$  as an upper bound on the number of misclassified points. The parameter  $C$  is a regularization parameter that controls the number of errors we are willing to tolerate on the training set. It is common to define this using  $C = 1/(vN)$ , where  $0 < v \leq 1$  controls the fraction of misclassified points that we allow during the training phase. This is called a  $v$ -SVM classifier. This is usually set using cross-validation (see Section 14.5.3).

#### 14.5.2.3 Probabilistic output

An SVM classifier produces a hard-labeling,  $\hat{y}(x) = \text{sign}(f(x))$ . However, we often want a measure of confidence in our prediction. One heuristic approach is to interpret  $f(x)$  as the log-odds ratio,  $\log \frac{p(y=1|x)}{p(y=0|x)}$ . We can then convert the output of an SVM to a probability using

$$p(y=1|x, \theta) = \sigma(a f(x) + b) \quad (14.67)$$

where  $a, b$  can be estimated by maximum likelihood on a separate validation set. (Using the training set to estimate  $a$  and  $b$  leads to severe overfitting.) This technique was first proposed in (Platt 2000). However, the resulting probabilities are not particularly well calibrated, since there is nothing in the SVM training procedure that justifies interpreting  $f(x)$  as a log-odds ratio. To illustrate this, consider an example from (Tipping 2001). Suppose we have 1d data where  $p(x|y=0) = \text{Unif}(0, 1)$  and  $p(x|y=1) = \text{Unif}(0.5, 1.5)$ . Since the class-conditional distributions overlap in the middle, the log-odds of class 1 over class 0 should be zero in [0.5, 1.0], and infinite outside this region. We sampled 1000 points from the model, and then fit an RVM and an SVM with a Gaussian kernel of width 0.1. Both models can perfectly capture the decision boundary, and achieve a generalization error of 25%, which is Bayes optimal in this problem. The probabilistic output from the RVM is a good approximation to the true log-odds, but this is not the case for the SVM, as shown in Figure 14.13.

#### 14.5.2.4 SVMs for multi-class classification

In Section 8.3.7, we saw how we could “upgrade” a binary logistic regression model to the multiclass case, by replacing the sigmoid function with the softmax, and the Bernoulli distribution with the multinomial. Upgrading an SVM to the multi-class case is not so easy, since the outputs are not on a calibrated scale and hence are hard to compare to each other. The obvious approach is to use a one-versus-the-rest approach (also called one-vs-all), in which we train  $C$  binary classifiers,  $f_c(x)$ , where the data from class  $c$  is treated as positive, and the data from all the other classes is treated as negative. However, this can result in regions of input space which are ambiguously labeled, as shown in Figure 14.14(a). A common alternative is to pick  $\hat{y}(x) = \arg \max_c f_c(x)$ . However, this

technique may not work either, since there is no guarantee that the different fc functions have comparable magnitudes. In addition, each binary subproblem is likely to suffer from the class imbalance problem. To see this, suppose we have 10 equally represented classes. When training f1, we will have 10% positive examples and 90% negative examples, which can hurt performance. It is possible to devise ways to train all C classifiers simultaneously (Weston and Watkins 1999), but the resulting method takes  $O(C^2N^2)$  time, instead of the usual  $O(CN^2)$  time.

Another approach is to use the one-versus-one or OVO approach, also called all pairs, in which we train  $C(C-1)/2$  classifiers to discriminate all pairs  $f_{c,c}$ . We then classify a point into the class which has the highest number of votes. However, this can also result in ambiguities, as shown in Figure 14.14(b). Also, it takes  $O(C^2N^2)$  time to train and  $O(C^2Nsv)$  to test each data point, where  $Nsv$  is the number of support vectors.<sup>2</sup> See also (Allwein et al. 2000) for an approach based on error-correcting output codes. It is worth remembering that all of these difficulties, and the plethora of heuristics that have been proposed to fix them, fundamentally arise because SVMs do not model uncertainty using probabilities, so their output scores are not comparable across classes.

#### 14.5.3 Choosing C

SVMs for both classification and regression require that you specify the kernel function and the parameter C. Typically C is chosen by cross-validation. Note, however, that C interacts quite strongly with the kernel parameters. For example, suppose we are using an RBF kernel with precision  $\gamma = 1/2\sigma^2$ . If  $\gamma = 5$ , corresponding to narrow kernels, we need heavy regularization, and hence small C (so  $\lambda = 1/C$  is big). If  $\gamma = 1$ , a larger value of C should be used. So we see that  $\gamma$  and C are tightly coupled. This is illustrated in Figure 14.15, which shows the CV estimate of the 0-1 risk as a function of C and  $\gamma$ . The authors of libsvm recommend (Hsu et al. 2009) using CV over a 2d grid with values  $C \in \{2^{-5}, 2^{-3}, \dots, 2^{15}\}$  and  $\gamma \in \{2^{-15}, 2^{-13}, \dots, 2^3\}$ . In addition, it is important to standardize the data first, for a spherical Gaussian kernel to make sense. To choose C efficiently, one can develop a path following algorithm in the spirit of lars (Section 13.3.4). The basic idea is to start

with  $\lambda$  large, so that the margin  $1/\|w(\lambda)\|$  is wide, and hence all points are inside of it and have  $\alpha_i = 1$ . By slowly decreasing  $\lambda$ , a small set of points will move from inside the margin to outside, and their  $\alpha_i$  values will change from 1 to 0, as they cease to be support vectors. When  $\lambda$  is maximal, the function is completely smoothed, and no support vectors remain. See (Hastie et al. 2004) for the details.

#### 14.5.4 Summary of key points

Summarizing the above discussion, we recognize that SVM classifiers involve three key ingredients: the kernel trick, sparsity, and the large margin principle. The kernel trick is necessary to prevent underfitting, i.e., to ensure that the feature vector is sufficiently rich that a linear classifier can separate the data. (Recall from Section 14.2.3 that any Mercer kernel can be viewed as implicitly defining a potentially high dimensional feature vector.) If the original features are already high dimensional (as in many gene expression and text classification problems), it suffices to use a linear kernel,  $\kappa(x, x') = x^T x'$ , which is equivalent to working with the original features.

The sparsity and large margin principles are necessary to prevent overfitting, i.e., to ensure that we do not use all the basis functions. These two ideas are closely related to each other, and both arise (in this case) from the use of the hinge loss function. However, there are other methods of achieving sparsity (such as L1), and also other methods of maximizing the margin (such as boosting). A deeper discussion of this point takes us outside of the scope of this book. See e.g., (Hastie et al. 2009) for more information.

#### 14.5.5 A probabilistic interpretation of SVMs

In Section 14.3, we saw how to use kernels inside GLMs to derive probabilistic classifiers, such as the L1VM and RVM. And in Section 15.3, we will discuss Gaussian process classifiers, which also use kernels. However, all of these approaches use a logistic or probit likelihood, as opposed to the hinge loss used by SVMs. It is natural to wonder if one can interpret the SVM more directly as a probabilistic model. To do so, we must interpret  $Cg(m)$  as a negative log likelihood, where  $g(m) = (1 - m)_+$ , where  $m = yf(x)$  is the margin. Hence  $p(y=1|f)=\exp(-Cg(f))$  and  $p(y=-1|f)=\exp(-Cg(-f))$ . By summing over both values of  $y$ , we

require that  $\exp(-Cg(f)) + \exp(-Cg(-f))$  be a constant independent of  $f$ . But it turns out this is not possible for any  $C > 0$  (Sollich 2002).

However, if we are willing to relax the sum-to-one condition, and work with a pseudolikelihood, we can derive a probabilistic interpretation of the hinge loss (Polson and Scott 2011). In particular, one can show that

$$\exp(-2(1 - y_i x^T w)^+) = \int_0^\infty \frac{1}{\sqrt{2\pi\lambda}} \exp(-\frac{1}{2}(1 + \lambda_i - y_i x^T w)^2) d\lambda_i \quad (14.68)$$

Thus the exponential of the negative hinge loss can be represented as a Gaussian scale mixture. This allows one to fit an SVM using EM or Gibbs sampling, where  $\lambda_i$  are the latent variables. This in turn opens the door to Bayesian methods for setting the hyper-parameters for the prior on  $w$ . See (Polson and Scott 2011) for details. (See also (Franc et al. 2011) for a different probabilistic interpretation of SVMs.)

已思考 34s

下面先给出准确中文翻译（含公式与图注，尽量保持原符号），接着是通俗详尽的解释，最后给出要点总结。

## 逐句翻译

### 图 14.10

- (a) 对三种损失的示意： $L_2$ 、Huber 以及  $\varepsilon$ -不敏感 ( $\varepsilon$ -insensitive) 损失，其中  $\varepsilon = 1.5$ 。图由 `huberLossDemo` 生成。
- (b) 在 SVM 回归中所用  $\varepsilon$ -管 ( $\varepsilon$ -tube) 的示意。位于管道上方的点满足  $\zeta_i > 0$ ,  $\zeta_i^{*} = 0$ ；位于管道下方的点满足  $\zeta_i = 0$ ,  $\zeta_i^{*} > 0$ ；位于管道内部的点满足  $\zeta_i = \zeta_i^{*} = 0$ 。（改绘自 Bishop 2006a 的图 7.7。）

**图 14.11 大间隔原则示意。**左：大间隔的分离超平面。右：小间隔的分离超平面。

### 图 14.12

- (a) 二维线性判别边界的几何示意。若  $f(x) > 0$ , 点  $x$  被判为属于决策区域  $R_1$ , 否则属于  $R_2$ ；其中  $f(x)$  称为**判别函数**。满足  $f(x) = 0$  的点的集合就是**决策边界**。向量  $w$  与决策边界**垂直**。项  $w_0$  控制决策边界到原点的距离。点  $x$  到其在边界上的正交投影  $x_\perp$  的**有符号距离**为  $f(x)/\|w\|$ 。（改绘自 Bishop 2006a 的图 4.1。）

(b) 软间隔原则示意。被圈出的点是支持向量，同时标出了相应的松弛变量  $\zeta$  的取值。 (改绘自 Bishop 2006a 的图 7.3。)

**图 14.13** 三种方法的对数几率 (log-odds) 随  $x$  的变化。 (改绘自 Tipping 2001 的图 10。)

#### 图 14.14

- (a) \*\*一对其余 (one-vs-rest) \*\*方法：绿色区域会同时被预测为类别 1 和类别 2。
- (b) \*\*一对一 (one-vs-one) \*\*方法：绿色区域的标签同样存在歧义。 (改绘自 Bishop 2006a 的图 4.2。)

#### 图 14.15

- (a) 在一个由两高斯混合生成的合成数据集上，采用 RBF 核的 SVM 分类器，对不同精度  $\gamma = 1/(2\sigma^2)$  与不同正则  $\lambda = 1/C$  的0-1 错误的交叉验证估计。
  - (b) 在  $\gamma = 5$  处的纵切面。红色虚线是用生成模型按贝叶斯法则计算的贝叶斯最优错误率。 (改绘自 Hastie 等 2009 的图 12.6。图由 `svmCgammaDemo` 生成。)
- 

## 14.5 支持向量机 (SVMs)

在 14.3.2 节里，我们看到一种得到稀疏核机器的方法：在 GLM 中使用核基函数，再配合  $\ell_1$  或 ARD 等促稀疏先验。另一种路线是把目标函数从负对数似然改成别的损失（见 6.5.5 节）。具体地，考虑  $\ell_2$  正则的经验风险：

$$J(w, \lambda) = \sum_{i=1}^N L(y_i, \hat{y}_i) + \lambda \|w\|^2 \quad (14.45)$$

其中  $\hat{y}_i = w^\top x_i + w_0$ 。（此处仍在原始特征空间；稍后引入核。）若  $L$  是平方损失，就得到岭回归；若  $L$  是 6.73 式的对数损失，则是逻辑回归。岭回归情形下解为  $\hat{w} = (X^\top X + \lambda I)^{-1} X^\top y$ ，预测为  $\hat{w}_0 + \hat{w}^\top x$ 。如 14.4.3 节所示，可把这些式子改写成只含  $\langle x, x' \rangle$  的形式，再用核  $\kappa(x, x')$  替换，得到核化（但不稀疏）。然而，如果把平方/对数损失改成下面将介绍的其他损失，我们可保证解是稀疏的：预测只依赖一部分训练点（支持向量）。这种“核技巧 + 改损失”的组合就是SVM。它最初用于二分类，也能扩展到回归和多分类（见后文）。

从概率观点看，SVM 有些“不自然”：

- 1) 把稀疏性体现在损失里，而不是先验里；
- 2) 核通过算法技巧引入，而不是模型结构的一部分；
- 3) 它不直接输出概率，在多分类时尤其麻烦（见 14.5.2.4）。

实际上，用 14.3.2 节的 L1VM 或 RVM 等也能得到稀疏、概率化、多分类的核方法，效果

不逊于 SVM。尽管如此，本书仍讨论 SVM，主要因为：a) 它非常流行；b) 在结构化输出情形有计算优势（见 19.7 节）。

### 14.5.1 用于回归的 SVM

核化的岭回归问题在于：解向量  $w$  依赖所有训练输入。我们想要稀疏解。Vapnik (Vapnik et al., 1997) 提出了 Huber 损失 (7.4 节) 的变体—— $\varepsilon$ -不敏感损失：

$$L_\varepsilon(y, \hat{y}) = \begin{cases} 0, & |y - \hat{y}| < \varepsilon \\ |y - \hat{y}| - \varepsilon, & \text{否则} \end{cases} \quad (14.46)$$

即：凡是落在预测曲线两侧宽度为  $\varepsilon$  的管道内的点，不计惩罚（见图 14.10）。相应目标常写成

$$J = C \sum_{i=1}^N L_\varepsilon(y_i, \hat{y}_i) + \frac{1}{2} \|w\|^2, \quad \hat{y}_i = f(x_i) = w^\top x_i + w_0, \quad C = 1/\lambda \quad (14.47)$$

该目标凸但不可导（因为绝对值）。如 13.4 节套索问题，可把它改成带约束的最优化：引入松弛变量刻画点超出管道的程度

$\begin{aligned} &y_i - f(x_i) + \varepsilon + \xi_i \geq 0 \\ &y_i - f(x_i) - \varepsilon - \xi_i = 0 \end{aligned}$

目标改写为

$$J = C \sum_{i=1}^N (\xi_i^+ + \xi_i^-) + \frac{1}{2} \|w\|^2 \quad (14.50)$$

在约束 (14.48)–(14.49) 以及  $\xi_i^\pm \geq 0$  下最小化。

这是一个含  $2N + D + 1$  个变量的二次规划。可证明（见 Schölkopf & Smola, 2002）最优解形如

$$\hat{w} = \sum_i \alpha_i x_i, \quad \alpha_i \geq 0 \quad (14.51)$$

而且向量  $\alpha$  稀疏（因为  $\varepsilon$  内的误差不关心）。满足  $\alpha_i > 0$  的  $x_i$  称为支持向量（误差在管道上或之外）。训练好后，

$$\hat{y}(x) = \hat{w}_0 + \hat{w}^\top x \quad (14.52)$$

代入  $\hat{w}$  得

$$\hat{y}(x) = \hat{w}_0 + \sum_i \alpha_i x_i^\top x \quad (14.53)$$

再把内积换成核，

$$\hat{y}(x) = \hat{w}_0 + \sum_i \alpha_i \kappa(x_i, x). \quad (14.54)$$

### 14.5.2 用于分类的 SVM

先讲二分类，多分类见 14.5.2.4。

#### 14.5.2.1 合页损失 (hinge loss)

6.5.5 节表明，逻辑回归的负对数似然

$$L_{\text{nll}}(y, \eta) = \log(1 + e^{-y\eta}), \quad \eta = f(x) = w^\top x + w_0 \quad (14.55)$$

是 0-1 风险的一个**凸上界**（此处标签取  $y \in \{1, -1\}$ ）。这里我们用**合页损失**

$$L_{\text{hinge}}(y, \eta) = \max(0, 1 - y\eta) = (1 - y\eta)_+ \quad (14.56)$$

$\eta = f(x)$  可理解为我们把  $y = 1$  作为正类时的“置信度”（非概率）。总体目标：

$$\min_{w, w_0} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N (1 - y_i f(x_i))_+ \quad (14.57)$$

同样不可导。引入松弛变量  $\xi_i$  可得等价约束式

$$\min_{w, w_0, \xi} \frac{1}{2} \|w\|^2 + C \sum_i \xi_i \quad \text{s.t. } \xi_i \geq 0, y_i(x_i^\top w + w_0) \geq 1 - \xi_i \quad (14.58)$$

这是一个含  $N + D + 1$  个变量、约  $O(N)$  个约束的**二次规划**。可消去原始变量，仅解  $N$  个对偶变量（拉格朗日乘子），标准 QP 求解约  $O(N^3)$ 。专用算法如 SMO (Platt, 1998) 可达  $O(N^2)$ ；当  $N$  很大时常用**线性 SVM**（训练  $O(N)$ ）。

解形如

$$\hat{w} = \sum_i \alpha_i x_i, \quad \alpha_i = \lambda_i y_i, \quad \alpha \text{ 稀疏} \quad (14.59)$$

满足  $\alpha_i > 0$  的样本为**支持向量**（误分或位于/以内于间隔），见图 14.12(b)。测试时

$$\hat{y}(x) = \text{sgn}(f(x)) = \text{sgn}(\hat{w}_0 + \hat{w}^\top x) \quad (14.60)$$

配合核技巧

$$\hat{y}(x) = \text{sgn}(\hat{w}_0 + \sum_{i=1}^N \alpha_i \kappa(x_i, x)) \quad (14.61)$$

计算量与**支持向量数**  $s$  成正比（线性核约  $O(sD)$ ；一般核为  $s$  次核调用）。

#### 14.5.2.2 大间隔原则

换个角度推导 (14.58)。在由核诱导的特征空间中 (判别函数线性)，如图 14.12(a)，

$$x = x_{\perp} + r \frac{w}{\|w\|} \quad (14.62)$$

其中  $r$  是点  $x$  到决策边界的距离， $x_{\perp}$  是其在边界上的正交投影。于是

$$f(x) = w^T x + w_0 = (w^T x_{\perp} + w_0) + r \frac{w^T w}{\|w\|}$$

由  $f(x_{\perp}) = 0$  得  $w^T x_{\perp} + w_0 = 0$ ，所以

$$f(x) = r \|w\| \Rightarrow r = \frac{f(x)}{\|w\|}. \quad (14.63)$$

我们想让 **最小的  $r$**  尽量大 (见图 14.11)，并保证每点在正确一侧，即  $y_i f(x_i) > 0$ 。目标可写为

$$\max_{w, w_0} \min_{i=1..N} \frac{y_i(w^T x_i + w_0)}{\|w\|}. \quad (14.64)$$

缩放  $w, w_0$  不改变距离，因此规定**最近点**满足  $y_i f_i = 1$ 。于是得到

$$\min_{w, w_0} \frac{1}{2} \|w\|^2 \text{ s.t. } y_i(w^T x_i + w_0) \geq 1. \quad (14.65)$$

若数据 (即便核化后) **不可分**，就引入松弛  $\xi_i \geq 0$ ：在正确边的间隔内  $0 < \xi_i \leq 1$ ，误分则  $\xi_i > 1$  (见图 14.12(b))。把硬约束  $y_i f_i \geq 0$  换成**软间隔**  $y_i f_i \geq 1 - \xi_i$ ：

$$\min_{w, w_0, \xi} \frac{1}{2} \|w\|^2 + C \sum_i \xi_i \text{ s.t. } \xi_i \geq 0, y_i(w^T x_i + w_0) \geq 1 - \xi_i \quad (14.66)$$

这与 (14.58) 相同。由于  $\xi_i > 1$  表示第  $i$  个样本被误分， $\sum_i \xi_i$  可视为误分数的上界。常把  $C$  写成  $C = 1/(vN)$  ( $0 < v \leq 1$ )， $v$  控制训练期允许的误分比例，称为  $v$ -SVM (通常用交叉验证设定，见 14.5.3)。

### 14.5.2.3 概率输出

SVM 给出**硬标签**  $\hat{y}(x) = \text{sign}(f(x))$ 。我们常希望得到**置信度**。一种经验做法是把  $f(x)$  当作**对数几率**并做

$$p(y = 1|x, \theta) = \sigma(af(x) + b) \quad (14.67)$$

其中  $a, b$  在**独立验证集**上用极大似然估计 (若用训练集会严重过拟合)。此法最早由 Platt (2000) 提出。但得到的“概率”通常**校准差**，因为 SVM 训练并不保证  $f(x)$  可被解释为对数几率。Tipping (2001) 的例子：若  $p(x|y=0) = \text{Unif}(0, 1), p(x|y=1) = \text{Unif}(0.5, 1.5)$ ，则在  $[0.5, 1.0]$  处真对数几率应为 0，外侧为  $\infty$ 。RVM 与 SVM (高斯核宽度 0.1) 都能

达到 25% 的贝叶斯最优错误，但 RVM 的概率输出近似真实对数几率，而 SVM 的并不（见图 14.13）。

#### 14.5.2.4 多分类的 SVM

8.3.7 节中，二元逻辑回归升级到多类很直接 ( $\text{sigmoid} \rightarrow \text{softmax}$ ,  $\text{Bernoulli} \rightarrow \text{multinomial}$ )。SVM 不易升级：其输出不在同一标度上，难以比较。

- **一对其余 (OvR)**：训练  $C$  个二分类器  $f_c(x)$  (第  $c$  类为正，其他为负)。可能出现同时被多个分类器判为正的区域 (图 14.14a)。常见替代是取  $\arg \max_c f_c(x)$ ，但不同  $f_c$  的量级不可比。且每个二分类都面临类别不平衡 (如 10 类等量时，训练  $f_1$  的正例仅 10%)。也可设计同时训练全部  $C$  个分类器的方法 (Weston & Watkins, 1999)，但训练代价变为  $O(C^2N^2)$  (常见做法为  $O(CN^2)$ )。
- **一对一 (OvO)**：对每一对类别训练  $C(C - 1)/2$  个分类器  $f_{c,c'}$ ，投票决定。但也会有投票冲突 (图 14.14b)。训练  $O(C^2N^2)$ ，测试每点  $O(C^2N_{sv})$ 。还可用纠错输出码 (ECOC; Allwein et al., 2000)。

根源：SVM 不建模不确定性，各类别的分数不可比，因此产生大量启发式权衡。

#### 14.5.3 如何选择 $C$

SVM (分类/回归) 都要指定核函数和参数  $C$ 。通常用交叉验证选  $C$ 。注意  $C$  与核超参强烈耦合。以 RBF 核为例， $\gamma = 1/(2\sigma^2)$ ：

- 若  $\gamma = 5$  (核很窄)，需要强正则  $\rightarrow$  小  $C$  (即  $\lambda = 1/C$  大)；
- 若  $\gamma = 1$ ，则应取较大的  $C$ 。

图 14.15 展示了 CV 的 0–1 风险随  $C$ 、 $\gamma$  的变化。libsvm 作者建议在网格上交叉验证 (Hsu et al., 2009)：

$$C \in \{2^{-5}, 2^{-3}, \dots, 2^{15}\}, \gamma \in \{2^{-15}, 2^{-13}, \dots, 2^3\}.$$

另外需先对数据做标准化，否则“球形高斯核”不合理。

更高效地选  $C$  可用类似 LARS 的路径跟踪：从很大的  $\lambda$  开始 (间隔宽，所有点在间隔内， $\alpha_i = 1$ )，缓慢减小  $\lambda$ ，只有少数点从间隔内移出、其  $\alpha_i$  从 1 变 0 (不再是支持向量)。当  $\lambda$  最大时，函数被完全平滑、无支持向量 (详见 Hastie et al., 2004)。

#### 14.5.4 关键点小结

SVM 分类器包含三要素：核技巧、稀疏性、大间隔原则。

- **核技巧防止欠拟合** (特征足够丰富，使线性分类器能分开数据；任意 Mercer 核等价于隐式高维特征)。若原始特征已高维且信息强 (如文本/基因表达)，用线性核  $\kappa(x, x') = x^\top x'$  常已足够。
- **稀疏 + 大间隔 防止过拟合** (并非使用所有基函数)。两者都来自合页损失。当然，也有别的促稀疏方式 (如  $\ell_1$ )，和别的“最大间隔”方法 (如 boosting)。

#### 14.5.5 SVM 的概率解释

14.3 节我们用核构造了**概率分类器**（如 L1VM、RVM）；15.3 节还会讲**高斯过程分类器**。它们都用 logistic/probit 似然，而 SVM 用**合页损失**。能否直接把 SVM 解释成概率模型？若要这样做，需把  $Cg(m)$  解释为负对数似然，其中  $g(m) = (1 - m)_+$ 、 $m = yf(x)$  为间隔。于是应有

$$p(y=1|f) = \exp(-Cg(f)), \quad p(y=-1|f) = \exp(-Cg(-f)).$$

要成立， $\exp(-Cg(f)) + \exp(-Cg(-f))$  必须与  $f$  **无关**。事实证明当  $C > 0$  时做不到 (Sollich, 2002)。

不过，如果放松“和为 1”的要求，用**伪似然**，可得到合页损失的概率解释 (Polson & Scott, 2011)。具体地，

$$\exp(-2(1 - y_i x_i^\top w)_+) = \int_0^\infty \frac{1}{\sqrt{2\pi\lambda_i}} \exp\left[-\frac{(1 + \lambda_i - y_i x_i^\top w)^2}{2\lambda_i}\right] d\lambda_i \quad (14.68)$$

即“负合页损失的指数”可表示为**高斯尺度混合**。由此可用 EM 或 Gibbs 拟合 SVM ( $\lambda_i$  为潜变量)，并可引入贝叶斯方法来设定  $w$  的先验超参 (见 Polson & Scott, 2011；另见 Franc et al., 2011 的不同解释)。

## 通俗解释（不依赖太多数学）

- **SVM 的核心直觉：**
  - 1) 把数据（可能核映射到高维）用一条**直线/超平面**分开；
  - 2) 这条线要离两边最近的点尽可能远（**大间隔**），这样新点更稳；
  - 3) 只在“边界上/违规”的点才影响这条线（**支持向量**），所以最后模型只记住少数训练点（**稀疏**）。
- **SVM-回归**：允许预测曲线周围有一条宽度  $\varepsilon$  的“**安全带**”，带内误差不罚，带外才罚，于是只少量“关键点”决定曲线。
- **为什么要核？** 核=相似度。把内积换成核，等效于在高维里做线性分割→原空间就是**非线性**边界；还能处理字符串、图结构等。
- **SVM 与概率方法的区别**：SVM 的分数不是概率，类间不可直接比；要概率常用 Platt 标定，但校准不一定好。像 RVM/GP 这类**概率模型**通常在概率上更可信。
- **多分类的麻烦**：因为“不是概率”，需要 OvR/OvO 等**拆分法**，会有歧义与额外复杂度。
- **超参数  $C$  要配合核**：核越“锋利”（RBF 的  $\gamma$  越大），越容易过拟合，就要小  $C$ （强正则）；反之亦然——实际靠**交叉验证**找好组合。

## 要点总结

- SVM = 核技巧 + 合页损失 + 大间隔 → 既能非线性又稀疏稳健。
- 回归版 SVM 用  $\varepsilon$ -不敏感损失：管内零惩罚 → 少量支持向量决定曲线。
- 分类版 SVM 的训练是一个二次规划；解只依赖支持向量；预测为  $\text{sign}(b + \sum \alpha_i \kappa(x_i, x))$ 。
- 大间隔 = 让最靠近边界的点离边界尽可能远；引入软间隔处理不可分与噪声。
- 概率性弱：输出非概率，多分类需 OvR/OvO 等技巧；若要好概率，可用 RVM/GP/校准。
- 调参： $C$  与核参数（如 RBF 的  $\gamma$ ）强耦合，需网格 CV；数据需标准化。
- 计算：标准核 SVM 训练代价随  $N$  呈二/三次方；可用线性 SVM/稀疏性/近似方法扩展规模。

这一小节主要在讲什么？我不太懂

已思考 9s

这小节在讲**支持向量机 (SVM)**：一种把“核技巧 + 大间隔 + 稀疏解”结合起来的学习方法，用于**分类和回归**。

## 核心在说什么（白话）

- 目标：在（可能由核映射后的）空间里找一条“直线/超平面”，既把两类分开，又离最近的点尽可能远（这叫**大间隔**）。
- 损失函数：分类用**合页损失 hinge**；回归用 **$\varepsilon$ -不敏感损失**（预测曲线两侧留一条宽度为  $\varepsilon$  的“安全带”，带内误差不罚）。
- 稀疏性：只有少数“关键样本”会决定那条线/曲线，它们叫**支持向量**；预测只用到这些点 → **模型小、预测快**。
- 核技巧：把内积换成核  $\kappa(x, x')$ ，等效于在高维做线性分割，从而在原空间得到**非线性**边界/曲线。

## 两个主要场景

### 1. SVM 回归

- 用  $\varepsilon$ -不敏感损失：管内 0 惩罚，超出管上/管下用松弛变量  $\zeta^+, \zeta^-$  计罚。
- 解可以写成： $\hat{y}(x) = b + \sum_i \alpha_i \kappa(x_i, x)$ ，只有**超出  $\varepsilon$** 管的点会有  $\alpha_i > 0$ 。

### 2. SVM 二分类

- 目标（软间隔形式）：最小化  $\frac{1}{2} \|w\|^2 + C \sum_i \zeta_i$ ，约束  $y_i f(x_i) \geq 1 - \zeta_i$ 。

- 解同样是稀疏的： $\hat{y}(x) = \text{sign}(b + \sum_i \alpha_i \kappa(x_i, x))$ 。

## 多分类怎么做

- 常用一一对其余 (OvR) 或一对一 (OvO) 拆分成多个二分类再组合，但会出现歧义/不平衡等工程问题（因为 SVM 的分数不是概率，类间难直接比较）。

## 超参数怎么选

- 需要选核及其参数（如 RBF 的  $\gamma = 1/2\sigma^2$ ）和正则强度  $C$ 。
- $\gamma$  与  $C$  强耦合：核越“窄”（ $\gamma$  大），越容易过拟合  $\rightarrow C$  要小；反之  $C$  可大。
- 实务上用网格交叉验证搜索（并注意先把特征标准化）。

## 和概率方法的区别

- SVM 不直接给概率，常用 Platt 标定把分数映成概率，但校准不一定好；若想“天生概率”的核方法，可看 RVM/GP 等。

## 这些图在传达的点

- 14.10：三种回归损失对比、以及  $\epsilon$ -管示意。
- 14.11–14.12：大/小间隔与软间隔、支持向量的几何意义。
- 14.13：SVM 的“概率”校准可能不佳（对比 RVM）。
- 14.14：OvR 与 OvO 在多分类里的歧义。
- 14.15： $C$  与  $\gamma$  的联合调参要靠交叉验证。

## 一句话总结

SVM = 用核把模型变非线性，用“大间隔”保泛化，用合页损失带来稀疏的“支持向量”解；  
回归用  $\epsilon$ -管，分类用软间隔；多分类与概率输出需额外技巧，调参靠交叉验证。

Method	Opt. w	Opt. kernel	Sparse	Prob.	Multiclass	Non-Mercer	Section
L2VM	Convex	EB	No	Yes	Yes	Yes	14.3.2
L1VM	Convex	CV	Yes	Yes	Yes	Yes	14.3.2
RVM	Not convex	EB	Yes	Yes	Yes	Yes	14.3.2
SVM	Convex	CV	Yes	No	Indirectly	No	14.5
GP	N/A	EB	No	Yes	Yes	No	15

翻译并解释以下内容，翻译不能有遗漏，解释请通俗易懂且详尽，因为我的数学基础不太好，最后总结要点。

Table 14.1 Comparison of various kernel based classifiers.  
EB = empirical Bayes, CV = cross validation. See text for details.

## 14.6 Comparison of discriminative kernel methods

We have mentioned several different methods for classification and regression based on kernels, which we summarize in Table 14.1. (GP stands for “Gaussian process”, which we discuss in Chapter 15.) The columns have the following meaning:

- Optimize  $w$ : a key question is whether the objective  $J(w) = -\log p(D|w) - \log p(w)$  is convex or not. L2VM, L1VM and SVMs have convex objectives. RVMs do not. GPs are Bayesian methods that do not perform parameter estimation.
- Optimize kernel: all the methods require that one “tune” the kernel parameters, such as the bandwidth of the RBF kernel, as well as the level of regularization. For methods based on Gaussians, including L2VM, RVMs and GPs, we can use efficient gradient based optimizers to maximize the marginal likelihood. For SVMs, and L1VM, we must use cross validation, which is slower (see Section 14.5.3).
- Sparse: L1VM, RVMs and SVMs are sparse kernel methods, in that they only use a subset of the training examples. GPs and L2VM are not sparse: they use all the training examples. The principle advantage of sparsity is that prediction at test time is usually faster. In addition, one can sometimes get improved accuracy.
- Probabilistic: All the methods except for SVMs produce probabilistic output of the form  $p(y|x)$ . SVMs produce a “confidence” value that can be converted to a probability, but such probabilities are usually very poorly calibrated (see Section 14.5.2.3).
- Multiclass: All the methods except for SVMs naturally work in the multiclass setting, by using a multinoulli output instead of Bernoulli. The SVM can be made into a multiclass classifier, but there are various difficulties with this approach, as discussed in Section 14.5.2.4.
- Mercer kernel: SVMs and GPs require that the kernel is positive definite; the other techniques do not.

Apart from these differences, there is the natural question: which method works best? In a small experiment<sup>3</sup>, we found that all of these methods had similar accuracy when averaged over a range of problems, provided they have the same kernel, and provided the regularization constants are chosen appropriately.

Given that the statistical performance is roughly the same,

what about the computational performance? GPs and L2VM are generally the slowest, taking  $O(N^3)$  time, since they don't exploit sparsity (although various speedups are possible, see Section 15.6). SVMs also take  $O(N^3)$  time to train (unless we use a linear kernel, in which case we only need  $O(N)$  time (Joachims 2006)). However, the need to use cross validation can make SVMs slower than RVMs. L1VM should be faster than an RVM, since an RVM requires multiple rounds of  $L_1$  minimization (see Section 13.7.4.3). However, in practice it is common to use a greedy method to train RVMs, which is faster than  $L_1$  minimization. This is reflected in our empirical results. The conclusion of all this is as follows: if speed matters, use an RVM, but if well-calibrated probabilistic output matters (e.g., for active learning or control problems), use a GP. The only circumstances under which using an SVM seems sensible is the structured output case, where likelihood-based methods can be slow. (We attribute the enormous popularity of SVMs not to their superiority, but to ignorance of the alternatives, and also to the lack of high quality software implementing the alternatives.) Section 16.7.1 gives a more extensive experimental comparison of supervised learning methods, including SVMs and various non kernel methods.

已思考 14s

下面先把表格与小节原文**完整翻译**, 再用**通俗解释**讲清每一列/每个方法的含义与取舍, 最后给出**要点总结与选型建议**。

## 逐字翻译

**表 14.1 各种基于核的分类器对比。EB = 经验贝叶斯 (empirical Bayes) , CV = 交叉验证 (cross validation) 。细节见正文。**

方法	优化 $w$	优化核	稀疏	概率化	多分类	可用 Merc
L2VM	Convex (凸)	EB	No	Yes	Yes	Yes
L1VM	Convex (凸)	CV	Yes	Yes	Yes	Yes

方法	优化 $w$	优化核	稀疏	概率化	多分类	可用: Mercer
RVM	Not convex (非凸)	EB	Yes	Yes	Yes	Yes
SVM	Convex (凸)	CV	Yes	No	Indirectly (间接)	No
GP	N/A	EB	No	Yes	Yes	No

## 14.6 判别式核方法的比较 (翻译)

我们已经提到了若干基于核的分类/回归方法，表 14.1 给出了汇总。（GP 指“高斯过程”，第 15 章讨论。）各列含义如下：

- **Optimize  $w$  (优化  $w$ )**：一个关键问题是目标  $J(w) = -\log p(D|w) - \log p(w)$  是否凸。L2VM、L1VM 与 SVM 的目标是凸的；RVM 不是；GP 属于贝叶斯方法，不进行参数估计。
- **Optimize kernel (优化核)**：所有方法都要“调”核的参数（如 RBF 的带宽）以及正则强度。对基于高斯的办法（包括 L2VM、RVM 与 GP），可以用高效的**梯度优化最大化边际似然**。对 SVM 与 L1VM，必须用**交叉验证**，更慢（见 14.5.3）。
- **Sparse (稀疏)**：L1VM、RVM 与 SVM 是**稀疏**的核方法——只用到训练集的一个子集。GP 与 L2VM **不稀疏**，会使用全部训练样本。稀疏的主要优点是**测试时预测更快**，有时还能提升准确率。
- **Probabilistic (概率化)**：除 SVM 外，其他方法都会输出形如  $p(y|x)$  的**概率**。SVM 给出一个“置信度”，可转成概率，但校准通常很差（见 14.5.2.3）。
- **Multiclass (多分类)**：除 SVM 外，其他方法天然支持多分类（把伯努利换成多项分布即可）。SVM 也能做多分类，但存在各种困难（14.5.2.4）。
- **Mercer kernel (正定核)**：SVM 与 GP 要求核是**正定**的；其他技术**不要求**。

除这些差异外，自然会问：**哪种最好？** 一个小实验显示：只要使用**相同的核**并恰当选择正则常数，上述方法的**平均准确率相近**。

既然统计性能差不多，那**计算性能**呢？GP 与 L2VM 一般**最慢**，训练要  $O(N^3)$ （不利用稀疏性，尽管有一些加速，见 15.6）。SVM 训练也要  $O(N^3)$ （若用线性核，只需  $O(N)$ ；Joachims 2006）。但 SVM 需要交叉验证，常比 RVM 更慢。按道理 L1VM 应比 RVM 快，因为 RVM 需要多轮  $\ell_1$  最小化（13.7.4.3）；不过实践里 RVM 常用**贪心算法**训练，反而比  $\ell_1$  更快，这也体现在我们的经验结果中。

**结论：**如果速度最重要，用 RVM；如果概率要校准得好（例如主动学习/控制问题），用 GP。只有在**结构化输出**场景、基于似然的方法训练很慢时，使用 SVM 才显得合理。（我们认为 SVM 的流行并非因为更优，而是因为不了解替代方案、且缺少高质量实现。）第 16.7.1 节对监督学习（含 SVM 与多种非核方法）做了更广泛的实验比较。

---

## 通俗解释与对照

### 这些缩写是什么？

- L2VM/L1VM：把核基函数丢进线性/逻辑回归，用  $\ell_2/\ell_1$  正则； $\ell_1$  会把很多系数推成  $0 \Rightarrow$  稀疏。
- RVM：相关向量机，贝叶斯稀疏化 (ARD/SBL)，通常比  $\ell_1$  还稀疏。
- SVM：用合页损失 + 大间隔 + 核技巧得到稀疏解；但不输出概率。
- GP：高斯过程，完全贝叶斯；天然输出带**不确定度**的预测，但不稀疏、训练贵。

### 每列怎么理解？

- “Optimize w: Convex/Not convex”：凸就像“碗形”目标，易全局最优；非凸可能有多峰，但 RVM 有好用的贪心近似。
- “Opt. kernel: EB vs CV”：
  - EB (经验贝叶斯) = 用数据直接最大化边际似然自动调核/正则，**一趟梯度搞定**；
  - CV (交叉验证) = 多组参数反复训练对比，**更耗时**。
- Sparse：只保留少量“关键训练点”（支持/相关向量）；预测只对这些点算核  $\Rightarrow$  **快**。
- Prob.：要的是**可比的概率**。SVM 的分数不是概率、不同类别不好比；L1VM/RVM/GP 则直接给  $p(y|x)$ 。
- Multiclass：概率模型换 softmax 就行；SVM 需 OvR/OvO 等拼装，存在歧义与复杂度。
- Non-Mercer：若核不是正定，SVM/GP 不适用；而 L2VM/L1VM/RVM 因为显式构造了有限维特征向量，可用一些**非正定的相似度**（更灵活）。

### 计算量直观（训练复杂度）：

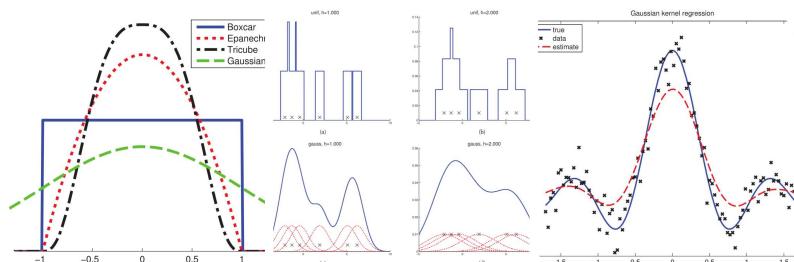
- GP / L2VM:  $O(N^3)$  (全体样本都用)。
  - SVM: 约  $O(N^3)$ ；线性核可到  $O(N)$ ，但调参要 CV。
  - RVM: 实践中用贪心  $\Rightarrow$  **通常最快**；同时**最稀疏**。
  - L1VM: 理论上应快，但需要求解  $\ell_1$  问题；实现和调参会影响速度。
- 

## 选型建议（快速决策表）

- 要可靠概率/不确定度（主动学习、决策控制、代价敏感）：GP。
- 要预测快、模型小：RVM（常最稀疏）。
- 特征已很高维（文本/基因表达）：先试线性核（L1VM / 线性 SVM / GP with linear kernel）。
- 工程上想用现成库、数据规模大：线性 SVM 或 L1VM（注意标准化与交叉验证）。
- 核不是正定或想用“领域相似度”：首选 L1VM/L2VM/RVM（可用非 Mercer）。
- \*\*结构化输出（序列、树、匹配）\*\*且要解大间隔问题：SVM（计算上有优势）。

## 要点总结

- 这些核方法在准确率层面差距不大（核与正则合适时）。
- 速度与输出形态才是主要差别：RVM 快且稀疏；GP 概率好；SVM 流行但概率差、调参靠 CV。
- 是否要求正定核：SVM/GP 要求，L2VM/L1VM/RVM 不严格要求。
- 选型时关注：是否要概率、是否追求稀疏/推理速度、核是否正定、数据维度与规模、是否多分类/结构化输出。



翻译并解释以下内容，翻译不能有遗漏，解释请通俗易懂且详尽，因为我的数学基础不太好，最后总结要点。

Figure 14.16 A comparison of some popular smoothing kernels. The boxcar kernel has compact support but is not smooth. The Epanechnikov kernel has compact support but is not differentiable at its boundary. The tri-cube has compact support and two continuous derivatives at the boundary of its support. The Gaussian is differentiable, but does not have compact support. Based on Figure 6.2 of (Hastie et al. 2009).

Figure 14.17 A nonparametric (Parzen) density estimator in 1D estimated from 6 data points, denoted by x. Top

row: uniform kernel. Bottom row: Gaussian kernel. Rows represent increasingly large bandwidth parameters. Based on [http://en.wikipedia.org/wiki/Kernel\\_density\\_estimation](http://en.wikipedia.org/wiki/Kernel_density_estimation). Figure 14.18 An example of kernel regression in 1d using a Gaussian kernel.

## 14.7 Kernels for building generative models

There is a different kind of kernel known as a smoothing kernel which can be used to create non-parametric density estimates. This can be used for unsupervised density estimation,  $p(x)$ , as well as for creating generative models for classification and regression by making models of the form  $p(y, x)$ .

### 14.7.1 Smoothing kernels

A smoothing kernel is a function of one argument which satisfies the following properties:

$$\kappa(x)dx = 1, x\kappa(x)dx = 0, x^2\kappa(x)dx > 0 \quad (14.69)$$

A simple example is the Gaussian kernel,

$$\kappa(x) = \frac{1}{(2\pi)^{1/2}} e^{-x^2/2} \quad (14.70)$$

We can control the width of the kernel by introducing a bandwidth parameter  $h$ :

$$\kappa_h(x) = \frac{1}{h} \kappa(\frac{x}{h}) \quad (14.71)$$

We can generalize to vector valued inputs by defining an RBF kernel:

$$\kappa_h(x) = \kappa_h(\|x\|) \quad (14.72)$$

In the case of the Gaussian kernel, this becomes

$$\kappa_h(x) = \frac{1}{h^D} \frac{1}{(2\pi)^{D/2}} \sum_{j=1}^D \exp(-\frac{1}{2h^2} x_j^2) \quad (14.73)$$

Although Gaussian kernels are popular, they have unbounded support. An alternative kernel, with compact support, is the Epanechnikov kernel, defined by

$$\kappa(x) = \frac{3}{4}(1 - x^2)I(|x| \leq 1) \quad (14.74)$$

This is plotted in Figure 14.16. Compact support can be useful for efficiency reasons, since one can use fast nearest neighbor methods to evaluate the density.

Unfortunately, the Epanechnikov kernel is not differentiable at the boundary of its support. An alternative is the tri-cube kernel, defined as follows:

$$\kappa(x) = \frac{70}{81}(1 - |x|^3)I(|x| \leq 1) \quad (14.75)$$

This has compact support and has two continuous derivatives at the boundary of its support. See Figure 14.16. The boxcar kernel is simply the uniform distribution:

$$\kappa(x) = I(|x| \leq 1) \quad (14.76)$$

We will use this kernel below.

#### 14.7.2 Kernel density estimation (KDE)

Recall the Gaussian mixture model from Section 11.2.1.

This is a parametric density estimator for data in RD.

However, it requires specifying the number K and locations  $\mu_k$  of the clusters. An alternative to estimating the  $\mu_k$  is to allocate one cluster center per data point, so  $\mu_i = x_i$ . In this case, the model becomes

$$p(x|D) = \frac{1}{N} \sum_{i=1}^N N(x|x_i, \sigma^2 I) \quad (14.77)$$

We can generalize the approach by writing

$$\hat{p}(x) = \frac{1}{N} \sum_{i=1}^N k_h(x - x_i) \quad (14.78)$$

This is called a Parzen window density estimator, or kernel density estimator (KDE), and is a simple non-parametric density model. The advantage over a parametric model is that no model fitting is required (except for tuning the bandwidth, usually done by cross-validation). and there is no need to pick K. The disadvantage is that the model takes a lot of memory to store, and a lot of time to evaluate. It is also of no use for clustering tasks. Figure 14.17 illustrates KDE in 1d for two kinds of kernel. On the top, we use a boxcar kernel,  $k(x) = I(-1 \leq z \leq 1)$ . The result is equivalent to a histogram estimate of the density, since we just count how many data points land within an interval of size h around  $x_i$ . On the bottom, we use a Gaussian kernel, which results in a smoother fit. The usual way to pick h is to minimize an estimate (such as cross validation) of the frequentist risk (see e.g., (Bowman and Azzalini 1997)). In Section 25.2, we discuss a Bayesian approach to non-parametric density estimation, based on Dirichlet process mixture models, which allows us to infer h. DP mixtures can also be more efficient than KDE, since they do not need to store all the data. See also Section 15.2.4 where we discuss an empirical Bayes approach to estimating kernel parameters in a Gaussian process model for classification/ regression.

#### 14.7.3 From KDE to KNN

We can use KDE to define the class conditional densities in a generative classifier. This turns out to provide an alternative derivation of the nearest neighbors classifier, which we introduced in Section 1.4.2. To show this, we follow the presentation of (Bishop 2006a, p125). In kde with a boxcar kernel, we fixed the bandwidth and count how many data points fall within the hyper-cube centered

on a datapoint. Suppose that, instead of fixing the bandwidth  $h$ , we instead allow the bandwidth or volume to be different for each data point. Specifically, we will "grow" a volume around  $x$  until we encounter  $K$  data points, regardless of their class label. Let the resulting volume have size  $V(x)$  (this was previously  $hD$ ), and let there be  $N_c(x)$  examples from class  $c$  in this volume. Then we can estimate the class conditional density as follows:

$$p(x|y = c, D) = N_c(x) / V(x) \quad (14.79)$$

where  $N_c$  is the total number of examples in class  $c$  in the whole data set. The class prior can be estimated by

$$p(y = c|D) = N_c / N \quad (14.80)$$

Hence the class posterior is given by

$$p(y = c|x, D) = N_c(x) / V(x) \quad (14.81)$$

where we used the fact that  $N_c(x) = K$ , since we choose a total of  $K$  points (regardless of class) around every point. This is equivalent to Equation 1.2, since  $N_c(x) = \sum_{i=1}^N I(y_i = c)$ .

#### 14.7.4 Kernel regression

In Section 14.7.2, we discussed the use of kernel density estimation or KDE for unsupervised learning. We can also use KDE for regression. The goal is to compute the conditional expectation

$$f(x) = E[y|x] = \int y p(y|x) dy = \int y p(x,y) dy / \int p(x,y) dy \quad (14.82)$$

We can use KDE to approximate the joint density  $p(x,y)$  as follows:

$$p(x,y) \approx \frac{1}{N} \sum_{i=1}^N k_h(x - x_i) k_h(y - y_i) \quad (14.83)$$

Hence

$$f(x) = \frac{1}{N} \sum_{i=1}^N k_h(x - x_i) \int y k_h(y - y_i) dy \quad (14.84)$$

$$= \frac{1}{N} \sum_{i=1}^N k_h(x - x_i) y_i \quad (14.85)$$

To derive this result, we used two properties of smoothing kernels. First, that they integrate to one, i.e.,  $\int k_h(y - y_i) dy = 1$ . And second, the fact that  $\int y k_h(y - y_i) dy = y_i$ . This follows by defining  $x = y - y_i$  and using the zero mean property of smoothing kernels:

$$(x + y_i) k_h(x) dx = x k_h(x) dx + y_i k_h(x) dx = 0 + y_i = y_i \quad (14.86)$$

We can rewrite the above result as follows:

$$f(x) = \sum_{i=1}^N w_i(x) y_i \quad (14.87)$$

$$w_i(x) = k_h(x - x_i) \quad (14.88)$$

We see that the prediction is just a weighted sum of the outputs at the training points, where the weights depend

on how similar  $x$  is to the stored training points. This method is called kernel regression, kernel smoothing, or the Nadaraya-Watson model. See Figure 14.18 for an example, where we use a Gaussian kernel. Note that this method only has one free parameter, namely  $h$ . One can show (Bowman and Azzalini 1997) that for 1d data, if the true density is Gaussian and we are using Gaussian kernels, the optimal bandwidth  $h$  is given by

$$h = 4 \sqrt{N} / (5 \sigma^2) \quad (14.89)$$

We can compute a robust approximation to the standard deviation by first computing the mean absolute deviation

$$MAD = \text{median}(|x - \text{median}(x)|) \quad (14.90)$$

and then using

$$\sigma^2 = 1.4826 \text{ MAD} = 1 / 0.6745 \text{ MAD} \quad (14.91)$$

The code used to produce Figure 14.18 estimated  $hx$  and  $hy$  separately, and then set  $h = hxhy$ .

Although these heuristics seem to work well, their derivation rests on some rather dubious assumptions (such as Gaussianity of the true density). Furthermore, these heuristics are limited to tuning just a single parameter. In Section 15.2.4 we discuss an empirical Bayes approach to estimating multiple kernel parameters in a Gaussian process model for classification/ regression, which can handle many tuning parameters, and which is based on much more transparent principles (maximizing the marginal likelihood).

#### 14.7.5 Locally weighted regression

If we define  $\kappa_h(x-x_i) = \kappa(x, x_i)$ , we can rewrite the prediction made by kernel regression as follows

$$\hat{f}(x^*) = \frac{1}{N} \sum_{i=1}^N y_i \kappa(x^*, x_i) \quad (14.92)$$

Note that  $\kappa(x, x_i)$  need not be a smoothing kernel. If it is not, we no longer need the normalization term, so we can just write

$$\hat{f}(x^*) = \sum_{i=1}^N y_i \kappa(x^*, x_i) \quad (14.93)$$

This model is essentially fitting a constant function locally.

We can improve on this by fitting a linear regression model for each point  $x^*$  by solving

$$\min \beta(x^*) \sum_{i=1}^N \kappa(x^*, x_i) [y_i - \beta(x^*) \varphi(x_i)]^2 \quad (14.94)$$

where  $\varphi(x) = [1, x]$ . This is called locally weighted regression. An example of such a method is LOESS, aka LOWESS, which stands for “locally-weighted scatterplot smoothing” (Cleveland and Devlin 1988). See also (Edakunni et al. 2010) for a Bayesian version of this model.

We can compute the parameters  $\beta(x^*)$  for each test case by solving the following weighted least squares problem:

$$\beta(x^*) = (\Phi^T D(x^*) \Phi)^{-1} \Phi^T D(x^*) y \quad (14.95)$$

where  $\Phi$  is an  $N \times (D + 1)$  design matrix and  $D = \text{diag}(\kappa(x^*, x_i))$ . The corresponding prediction has the form

$$\hat{f}(x^*) = \varphi(x^*)^T \beta(x^*) = (\Phi^T D(x^*) \Phi)^{-1} \Phi^T D(x^*) y = \sum_{i=1}^N w_i(x^*) y_i \quad (14.96)$$

The term  $w_i(x^*)$ , which combines the local smoothing kernel with the effect of linear regression, is called the equivalent kernel. See also Section 15.4.2.

下面先给出**完整直译**（含图注与公式），然后是**通俗讲解**，最后用**要点总结**快速回顾。

---

## 逐字翻译

**图 14.16** 一些常见平滑核 (smoothing kernels) 的比较。Boxcar (长方窗) 核具有紧支撑 (compact support) 但不光滑；Epanechnikov 核具有紧支撑，但在其支撑的边界处不可导；Tri-cube 核具有紧支撑，并且在支撑边界处具有二阶连续导数；Gaussian (高斯) 核可导，但没有紧支撑。改绘自 (Hastie et al., 2009) 的图 6.2。

**图 14.17** 用 6 个数据点 (以  $\times$  标出) 在 1 维上得到的非参数 (Parzen) 密度估计。上排：均匀核；下排：高斯核。每一排从左到右表示带宽参数逐渐增大。基于维基百科“Kernel density estimation”页面。

**图 14.18** 使用高斯核的一维核回归示例。

---

## 14.7 用于构造生成模型的核

存在另一类称为平滑核 (smoothing kernel) 的核函数，可用于创建非参数密度估计。它既可用于无监督地估计  $p(x)$ ，也可以通过构造形如  $p(y, x)$  的模型，为分类与回归建立生成式模型。

### 14.7.1 平滑核

平滑核是一个单变量函数，满足以下性质：

$$\int \kappa(x) dx = 1, \quad \int x \kappa(x) dx = 0, \quad \int x^2 \kappa(x) dx > 0 \quad (14.69)$$

一个简单例子是高斯核：

$$\kappa(x) = \frac{1}{(2\pi)^{1/2}} e^{-x^2/2} \quad (14.70)$$

可以通过**带宽参数**  $h$  控制核的宽度：

$$\kappa_h(x) = \frac{1}{h} \kappa\left(\frac{x}{h}\right) \quad (14.71)$$

可推广到向量输入，定义为**RBF 核**：

$$\kappa_h(x) = \kappa_h(\|x\|) \quad (14.72)$$

在高斯核的情形下，有

$$\kappa_h(x) = \frac{1}{h^D (2\pi)^{D/2}} \prod_{j=1}^D \exp\left(-\frac{x_j^2}{2h^2}\right) \quad (14.73)$$

虽然高斯核流行，但它的**支撑是无界的**。一种具有**紧支撑**的替代核是 **Epanechnikov 核**：

$$\kappa(x) = \frac{3}{4}(1 - x^2) I(|x| \leq 1) \quad (14.74)$$

如图 14.16 所示。紧支撑在效率上有用，因为可以用**快速近邻方法**计算密度。不幸的是，Epanechnikov 核在支撑边界处**不可导**。另一种是 **tri-cube 核**：

$$\kappa(x) = \frac{70}{81}(1 - |x|^3)^3 I(|x| \leq 1) \quad (14.75)$$

它既有紧支撑，又在支撑边界处具有**两次连续可导**（见图 14.16）。Boxcar 核就是**均匀分布**：

$$\kappa(x) = I(|x| \leq 1) \quad (14.76)$$

下面我们会用到该核。

### 14.7.2 核密度估计 (KDE)

回忆 11.2.1 节的**高斯混合模型**：它是  $\mathbb{R}^D$  数据的**参数化**密度估计，但需要指定簇数  $K$  与中心  $\mu_k$ 。一种替代是：为每个数据点都分配一个“簇中心”，即  $\mu_i = x_i$ 。此时模型变为

$$p(x|D) = \frac{1}{N} \sum_{i=1}^N N(x|x_i, \sigma^2 I) \quad (14.77)$$

进一步可推广为

$$\hat{p}(x) = \frac{1}{N} \sum_{i=1}^N \kappa_h(x - x_i) \quad (14.78)$$

这称为Parzen 窗密度估计或核密度估计 (KDE) , 是一种简单的**非参数**密度模型。相对参数模型, 它的优点是**无需拟合模型结构** (只需调带宽, 通常用交叉验证), 也**无需选择 K**。缺点是**占内存且计算慢**; 此外**不适合做聚类**。图 14.17 展示了 1 维 KDE 的例子: 上面用 **boxcar 核**  $\kappa(x) = I(-1 \leq z \leq 1)$ , 效果等价于**直方图** (统计以  $x_i$  为中心、宽度  $h$  的区间内点的个数); 下面用**高斯核**, 得到**更平滑**的拟合。选  $h$  的常见方法是最小化对**频率学风险** (如交叉验证) 的估计 (见 Bowman & Azzalini, 1997)。第 25.2 节会讨论基于 Dirichlet 过程混合的贝叶斯**非参数密度估计**, 它可以推断  $h$ , 并且由于不需要存所有数据, 常比 KDE **更高效**。另见 15.2.4 节, 我们将讨论在**高斯过程分类/回归**中, 用**经验贝叶斯**估计核参数。

### 14.7.3 从 KDE 到 KNN

可用 KDE 定义生成分类器里的**类条件密度**, 这会给出**最近邻分类器**的另一种推导 (1.4.2 节)。按 (Bishop, 2006a, p125) 的表述: 在用 **boxcar 核** 的 KDE 中, 我们固定带宽  $h$ , 统计以样本为中心的**超立方体**里落入的点数。现在不固定  $h$ , 而是让该体积**针对每个查询点自适应扩大**, 直到包含  $K$  个**数据点** (不管它们的类别)。设此体积为  $V(x)$  (之前是  $h^D$ ), 其中来自类别  $c$  的点有  $N_c(x)$  个。则类条件密度估计为

$$p(x|y=c, D) = \frac{N_c(x)}{N_c V(x)} \quad (14.79)$$

其中  $N_c$  是全体数据中属于类别  $c$  的样本数。类先验估计为

$$p(y=c|D) = \frac{N_c}{N} \quad (14.80)$$

于是后验为

$$p(y=c|x, D) = \frac{\frac{N_c(x)}{N_c V(x)} \frac{N_c}{N}}{\sum_{c'} \frac{N_{c'}(x)}{N_{c'} V(x)} \frac{N_{c'}}{N}} = \frac{N_c(x)}{\sum_{c'} N_{c'}(x)} = \frac{N_c(x)}{K} \quad (14.81)$$

最后一步用了  $\sum_c N_c(x) = K$ , 因为我们总是围绕每个点选取  $K$  个邻居 (不分类别)。这与公式 (1.2) 等价, 因为  $N_c(x) = \sum_{i \in N_K(x, D)} I(y_i = c)$ 。

### 14.7.4 核回归

14.7.2 节讨论了 KDE 在**无监督学习**中的用法; 它也能用于**回归**。目标是计算条件期望

$$f(x) = E[y|x] = \frac{\int y p(x, y) dy}{\int p(x, y) dy} \quad (14.82)$$

用 KDE 近似联合密度:

$$p(x, y) \approx \frac{1}{N} \sum_{i=1}^N \kappa_h(x - x_i) \kappa_h(y - y_i) \quad (14.83)$$

于是

$$f(x) = \frac{\frac{1}{N} \sum_i \kappa_h(x - x_i) \int y \kappa_h(y - y_i) dy}{\frac{1}{N} \sum_i \kappa_h(x - x_i) \int \kappa_h(y - y_i) dy} = \frac{\sum_i \kappa_h(x - x_i) y_i}{\sum_i \kappa_h(x - x_i)} \quad (14.84-14.85)$$

推导用到平滑核的两条性质:  $\int \kappa_h(y - y_i) dy = 1$ , 以及

$$\int y \kappa_h(y - y_i) dy = y_i \quad (\text{令 } x = y - y_i, \text{ 用 } \int x \kappa_h(x) dx = 0). \quad (14.86)$$

可写成

$$f(x) = \sum_{i=1}^N w_i(x) y_i, \quad w_i(x) = \frac{\kappa_h(x - x_i)}{\sum_j \kappa_h(x - x_j)} \quad (14.87-14.88)$$

即预测是对训练输出  $y_i$  的**相似度加权平均**, 权重取决于  $x$  与  $x_i$  的相似度。这称为**核回归 / 核平滑 / Nadaraya–Watson 模型**。图 14.18 给出用高斯核的例子。该法只有一个自由参数: **带宽  $h$** 。可证明 (Bowman & Azzalini, 1997) : 在 1 维、真实密度为高斯且核也为高斯时, 最优带宽

$$h = \left( \frac{4}{3N} \right)^{1/5} \hat{\sigma} \quad (14.89)$$

其中先用**中位绝对偏差估计**

$$\text{MAD} = \text{median}(|x - \text{median}(x)|) \quad (14.90)$$

再用

$$\hat{\sigma} = 1.4826 \text{ MAD} = \frac{1}{0.6745} \text{ MAD}. \quad (14.91)$$

生成图 14.18 的代码分别估计了  $h_x$  与  $h_y$ , 然后设  $h = h_x h_y$ 。

尽管这些启发式方法效果不错, 但其推导依赖一些**可疑假设** (如真实分布高斯性), 且只能调单个参数。15.2.4 节将介绍在**高斯过程**模型中的**经验贝叶斯**做法, 可同时估计多个核参数, 并以**最大化边际似然**为原则, 更透明。

### 14.7.5 局部加权回归 (Locally weighted regression)

若令  $\kappa_h(x - x_i) = \kappa(x, x_i)$ , 核回归的预测可写为

$$\hat{f}(x^*) = \frac{\sum_i y_i \kappa(x^*, x_i)}{\sum_i \kappa(x^*, x_i)} \quad (14.92)$$

注意  $\kappa(x, x_i)$  不一定是平滑核。若它不是，就不必做分母归一化，可直接写成

$$\hat{f}(x^*) = \sum_i y_i \kappa(x^*, x_i). \quad (14.93)$$

此模型本质是在局部拟合一个常数。可改进为：对每个  $x^*$  用加权线性回归拟合

$$\min_{\beta(x^*)} \sum_i \kappa(x^*, x_i) [y_i - \beta(x^*)^\top \phi(x_i)]^2, \quad \phi(x) = [1, x] \quad (14.94)$$

这称为**局部加权回归** (LOESS/LOWESS, Cleveland & Devlin, 1988)。其贝叶斯版本见 Edakunni et al. (2010)。对每个测试点，参数由**加权最小二乘**给出：

$$\beta(x^*) = (\Phi^\top D(x^*) \Phi)^{-1} \Phi^\top D(x^*) y \quad (14.95)$$

其中  $\Phi$  为  $N \times (D + 1)$  的设计矩阵， $D(x^*) = \text{diag}(\kappa(x^*, x_i))$ 。预测为

$$\hat{f}(x^*) = \phi(x^*)^\top \beta(x^*) = (\Phi^\top D(x^*) \Phi)^{-1} \Phi^\top D(x^*) y = \sum_i w_i(x^*) y_i \quad (14.96)$$

这里  $w_i(x^*)$  同时综合了**局部核与线性回归**的作用，称为**等效核** (equivalent kernel)。另见 15.4.2 节。

## 通俗解释

### 1) 什么是“平滑核”？

把它当作一个**形状函数**：以你要估的位置为中心，给附近的数据**分配权重**。权重之和为 1 (式 14.69 第一条)，整体平均为 0 (第二条)，有一定方差 (第三条)。

- **带宽**  $h$  决定“影响范围”， $h$  小→看得近、曲线更**抖**； $h$  大→看得远、曲线更**平**。
- **紧支撑** (只在有限区间非零) → 计算可只看近邻，快；但在边界处要注意**光滑性** (是否可导)。图 14.16 就比较了不同核的“形状、平滑性与支撑”。

### 2) 核密度估计 (KDE)

把每个样本“涂抹”成一个小鼓包 (核)，再把所有鼓包**相加平均**，就得到整个分布的估计 (式 14.78)。

- **boxcar** 相当于直方图；
- **高斯核**会让曲线更顺滑 (图 14.17)。

关键是挑好**带宽**  $h$ ：太小→过拟合，太大→过平滑。一般用交叉验证或经验公式。

### 3) 从 KDE 推出 KNN

如果不用固定  $h$ , 而是对每个点把体积膨胀到刚好有  $K$  个邻居, 则后验  $p(y = c|x)$  恰好等于“这  $K$  个邻居中属于类  $c$  的比例”(式 14.81)。这就是K 近邻分类器。

### 4) 核回归 (Nadaraya–Watson)

预测  $f(x) =$  “附近样本  $y_i$  的加权平均”(式 14.87–14.88), 权重是对距离的核函数。唯一超参是带宽  $h$ 。图 14.18 里, 红虚线是核回归的拟合。

### 5) 局部加权回归 (LOESS/LOWESS)

核回归等价于“在局部拟合一个常数”。LOESS 进一步在局部拟合一条线(式 14.94), 能更好地捕捉斜率变化, 本质是“对每个测试点做一次加权最小二乘”。

## 要点总结

- **平滑核**: 权重函数; 带宽  $h$  控制平滑程度。紧支撑核(如 boxcar/Epanechnikov/tricube)只影响邻近、算得快; 高斯核处处非零但很平滑。
- **KDE**: 把每个样本变成一个核、求和平均得到  $\hat{p}(x)$ 。优点: 无需设簇数; 缺点: 要存全体数据、计算慢。
- **KDE  $\rightarrow$  KNN**: 自适应选择体积到含  $K$  个点  $\Rightarrow$  后验等于“邻居投票占比”。
- **核回归**: 预测 = 样本输出的相似度加权平均(Nadaraya–Watson); 只需调带宽  $h$ 。
- **LOESS**: 在局部做**加权线性回归**, 比简单加权平均更灵活; 可用矩阵公式一次性算出每个测试点的回归系数。
- 选型提示: 想要**平滑、泛化强** $\rightarrow h$ 大; 想要**细节、灵敏** $\rightarrow h$ 小; 大数据时优先考虑**紧支撑核 + 近邻加速或用贝叶斯/GP 的边际似然**来系统地选核参数。

这一小节主要在讲什么? 我不太懂

已思考 17s

这小节讲的是另一种“核”——**平滑核**(smoothing kernel), 用它来**搭建生成式模型**: 估计概率密度  $p(x)$ 、类条件密度  $p(x|y)$ , 以及做**核回归/局部加权回归**。它跟前面用在 SVM 等判别式方法里的“Mercer 核/核技巧”不是一回事。

## 核心想法 (通俗版)

把每个样本点当成一个会“发光/起雾”的小灯泡(核), 在它附近给更高的权重, 远处权重变小。把所有灯泡的亮度**叠加平均**, 就得到了:

- 整体的雾的浓度 → 对应 密度估计  $\hat{p}(x)$  (KDE/Parzen 窗)；
- 同一位置附近 y 的加权平均 → 对应 核回归 (Nadaraya–Watson)；
- 若把“雾团”扩张到刚好包含 K 个邻居，就得到 KNN 分类的概率解释。

## 各块在说什么

### 1. 平滑核是什么

- 是单变量权重函数  $\kappa$ ，满足积分为 1、均值 0、方差>0。
- 常见形状：Boxcar (方窗)、Epanechnikov、Tri-cube、Gaussian。
- 关键超参是带宽  $h$ :  $h$  小→曲线“抖”(细节多)， $h$  大→曲线“平”(细节少)。
- 是否紧支撑：方窗/Epanechnikov/Tri-cube 只在有限区间非零(便于近邻加速)；高斯处处非零但最平滑。

### 2. 核密度估计 KDE

- 公式:  $\hat{p}(x) = \frac{1}{N} \sum_{i=1}^N \kappa_h(x - x_i)$ 。
- 直观：给每个样本抹一个“鼓包”，再相加。方窗像直方图，高斯更顺滑。
- 优点：**不用定簇数/参数方程；**缺点：**要存全体数据、评估慢、在高维易失效(维度灾难)，不适合聚类。

### 3. 从 KDE 到 KNN

- 若不固定  $h$ ，改为把体积扩到刚好含  $K$  个点，可推出  

$$p(y = c|x) = \frac{N_c(x)}{K}$$
 (邻居中有多少属于类  $c$ )。
- 这给 KNN 一个**生成式**的推导。

### 4. 核回归 (Nadaraya–Watson)

- 目标  $f(x) = E[y|x]$ ，结果变成  

$$\hat{f}(x) = \frac{\sum_i \kappa_h(x - x_i) y_i}{\sum_i \kappa_h(x - x_i)}$$
  
= “离我越近权重越大的加权平均”。
- 只有一个主要超参：带宽  $h$  (可用交叉验证或一些经验公式选)。

### 5. 局部加权回归 (LOESS/LOWESS)

- 核回归等价于在局部拟合常数；LOESS 在每个测试点附近做**加权线性回归**，能更好跟上局部斜率变化。

## 跟“核技巧”的区别

- 这里的核是**平滑权重**，不要求正定(非 Mercer 也行)，用于**估计概率/做局部平均**；
- 之前的核技巧是把内积替换成  $\kappa(x, x')$ ，用于**构造高维线性分类/回归**，要求Mercer 正定。

## 实用建议

- **形状不如带宽重要**: 选核形状 (高斯/三次方窗等) 影响小, 调好  $h$  才关键。
- **大数据/高维**: 优先考虑**紧支撑核 + 近邻索引**, 或改用可学习核参数的**高斯过程**等方法。
- **要概率密度用 KDE; 要数值预测先试核回归/LOESS; 要“投票式”分类用 KNN** (它其实就是自适应带宽的 KDE)。

## 一句话总结

这一小节教你用**平滑核做非参数生成式建模**: 把数据“抹开”再相加 (KDE) 得到  $p(x)$ , 据此能推导 KNN; 把输出也按相似度加权 (核回归/LOESS) 得到  $f(x)$ 。核心旋钮是**带宽**  $h$ 。

紧支撑是什么

已思考 20s

“**紧支撑** (compact support) ”指的是:

一个函数  $f(x)$  在**某个有限区域外完全为 0**。更正式地说,

$$\text{supp}(f) = \overline{\{x : f(x) \neq 0\}}$$

叫做函数的**支撑**; 如果这个支撑集是**紧集**, 就说  $f$  具有**紧支撑**。在  $\mathbb{R}^d$  中, “紧”等价于“**有界且闭**” (Heine–Borel 定理), 也就是落在一个有限大小的盒子里, 边界也包含在内。

## 直观比喻

- 把核函数看成一盏灯:
  - **紧支撑核**像“手电筒有硬边”, 照到一定半径外就**彻底黑** (值为 0)。
  - **非紧支撑核** (如高斯) 像“没有硬边的灯”, 越远越暗, 但**永远不为 0**。

## 例子

- 有**紧支撑** (半径 1 之外为 0) :
  - **Boxcar/Uniform**:  $\kappa(x) = \mathbf{1}(|x| \leq 1)$
  - **Epanechnikov**:  $\kappa(x) = \frac{3}{4}(1 - x^2) \mathbf{1}(|x| \leq 1)$
  - **Tri-cube**:  $\kappa(x) = \frac{70}{81}(1 - |x|^3)^3 \mathbf{1}(|x| \leq 1)$
- 无**紧支撑**:
  - **Gaussian**:  $\kappa(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$  (处处非零)

- Laplace 等指数尾分布

带宽  $h$  缩放后，紧支撑核常写成  $\kappa_h(x) = \frac{1}{h} \kappa(x/h)$ ，于是当  $|x| > h$  时取 0。

## 为什么重要（在平滑核/KDE/局部回归里）

### 好处

- **计算快**：评估  $\hat{p}(x) = \frac{1}{N} \sum_i \kappa_h(x - x_i)$  时，只需考虑半径  $h$  内的点，可用 KD-tree/球树等近邻索引；复杂度从看全部  $N$  个点降到看附近的  $k$  个点。
- **局部性强**：远处异常点完全不影响当前位置。

### 注意点

- **边界平滑性**：有些紧支撑核在边界不够光滑（如 Epanechnikov 在边界不可导），可能让曲线在边缘有折角；Tri-cube/“bump”函数可在边界保持一二阶连续。
- **边界偏差**：在样本域边缘，紧支撑核更容易产生偏差（因为可用数据在一侧更少）；常用镜像修正或边界校正方法。

## 和前面“核技巧”的关系

- 紧支撑是“平滑核”的一个性质，讲的是权重的空间范围，用于 KDE、核回归、LOESS 等生成式/局部平均方法；不要求正定。
- SVM 等“核技巧”要求的是 Mercer 正定核（便于把内积替换为  $\kappa(x, x')$ ），与是否紧支撑是另一件事；高斯核就不是紧支撑但常用于 SVM。

### 一句话：

紧支撑 = 函数在某个有限区域外严格为 0。在核平滑/密度估计/局部回归中，它带来“只看近邻、算得快、抗远处噪声”的优势，但要关注边界的光滑性与偏差问题。

基于本对话里我贴过的整章原文，输出：  
①知识点总结  
(分点+关键词+用途) ②子章节关联 (依赖/对比/递进+关系句，附Mermaid可选) ③本章总结。

已思考 36s

## ① 知识点总结（分点 + 关键词 + 用途）

- **核 (Kernel) 的两层含义**
  - 关键词：Mercer/正定核、平滑核 (smoothing kernel)
  - 用途：

- Mercer 核 → 通过**核技巧**把线性算法“升维”成**非线性**；常用于判别式学习（SVM、核化回归/分类、kPCA）。
- 平滑核 → 做**非参数生成式建模** (KDE、核回归、LOESS)，按“距离近权重大”做局部平均。
- **常见 Mercer 核与相似度**
  - RBF/高斯核： $\exp(-\|x - x'\|^2/2\sigma^2)$ ；关键词：**带宽**  $\sigma$ 、**各向同性**；用途：通用非线性相似度 (SVM、GP、RVM 等)。
  - ARD/RBF：维度自适应尺度；忽略不重要维度。
  - **线性核**： $x^\top x'$ ；高维稀疏特征 (文本、基因表达) 常有效。
  - **多项式核**： $(\gamma x^\top x' + r)^M$ ；关键词：**显式等价特征** (含到  $M$  次项)；用途：可控阶的非线性。
  - **Matérn 核**：平滑度由  $\nu$  控制； $\nu \rightarrow \infty$  近似 SE；用途：GP 回归里用来建模不同光滑性。
  - **文档核 (TF-IDF + 余弦)**：关键词：tf-idf、**停用词**、**爆发性用词**；用途：检索/分类。
  - **字符串/树核**：计数共享子串/子结构 (k-spectrum 等)；用途：生物序列、NLP 解析树。
  - **金字塔匹配核**：多分辨率直方图交；用途：图像的“包-词袋”匹配。
  - **由生成模型导出的核**
    - **概率乘积核 (PPK)**： $\int p(x|\hat{\theta}_i)^\rho p(x|\hat{\theta}_j)^\rho dx$ ；高斯情形  $\approx$  RBF。
    - **Fisher 核**：**得分函数**  $g = \nabla_{\theta} \log p(x|\hat{\theta})$ 、**Fisher 信息**；用途：把生成模型特征化为判别特征。
- **Mercer 定理与 Gram 矩阵**
  - 关键词：**正定 Gram 矩阵**、**特征映射**  $\phi(x)$ 、**特征无限维**。
  - 用途：说明只要核是正定，就等价于在某个 (可能无限维)  $\phi$  空间里做线性内积。
- **核化 GLM (Kernel machines)**
  - 关键词：**核化特征向量**  $\phi(x) = [\kappa(x, \mu_k)]$ 、**RBF 网络**、**原型/质心选择**。
  - 用途：让逻辑回归/线性回归获得**非线性边界/曲线**。
- **稀疏核方法**
  - L1VM： $\ell_1$  正则 → 稀疏、可多类 (组 lasso)；调参用 CV。
  - RVM：ARD/SBL → **最稀疏**、训练常用**贪心**；**概率输出**；核参数可 EB。
  - L2VM： $\ell_2$  正则，不稀疏；**概率输出**；核参数可 EB。

- SVM：合页损失 + 大间隔 + 核技巧；稀疏但非概率；多分类需 OvR/OvO；调参数靠  $C, \gamma$  网格 CV。
- 核技巧 (Kernel trick) 与核化算法
  - 关键词：用  $\kappa(x, x')$  替换内积、只依赖距离/相似度。
  - 用途：核化 1NN、K-medoids、岭回归（对偶）、kPCA（含中心化  $K \rightarrow HKH$ ）。
- SVM 细节
  - 回归： $\varepsilon$ -不敏感损失， $\varepsilon$ -管；支持向量=落在管外或边上的点。
  - 分类：合页损失、软间隔、大间隔几何；对偶解  $\hat{f}(x) = b + \sum \alpha_i \kappa(x_i, x)$ 。
  - 概率化：Platt 标定可得分→概率，但校准常差；RVM/GP 概率自然、校准好。
  - 多分类：OvR/OvO 存在歧义/不平衡/复杂度问题；也有联合优化但代价  $O(C^2 N^2)$ 。
  - 选参： $C$  与 RBF 的  $\gamma (= 1/2\sigma^2)$  强耦合；标准化特征再做网格 CV。
- 判别式核方法对比 (L2VM/L1VM/RVM/SVM/GP)
  - 关键词：凸性、核参数优化 (EB vs CV)、稀疏性、概率输出、多类、是否需 Mercer、训练复杂度。
  - 用途：选型：快→RVM，要好概率→GP，结构化输出→SVM 常见；高维稀疏特征优先线性核。
- 平滑核与生成式学习
  - 平滑核：积分 1、均值 0、方差>0；形状：boxcar / Epanechnikov / tri-cube (紧支撑)，Gaussian (无紧支撑)；带宽  $h$  决定平滑度。
  - KDE (核密度估计)： $\hat{p}(x) = \frac{1}{N} \sum \kappa_h(x - x_i)$ ；直方图≈boxcar，Gaussian 更平滑；优缺点：无需设 K、但存储/评估慢、高维困难。
  - KDE → KNN：把体积自适应到含 K 邻居  $\Rightarrow p(y = c|x) = N_c(x)/K$ 。
  - 核回归 (Nadaraya-Watson)： $\hat{f}(x) = \frac{\sum \kappa_h(x - x_i) y_i}{\sum \kappa_h(x - x_i)}$ ；只调  $h$ 。
  - LOESS/LOWESS：在每个查询点做局部加权线性回归，等效核更灵活。

## ② 子章节关联 (依赖 / 对比 / 递进)

- 14.1 引言 → 14.2 核函数：提出“对象难以定长表征 / 需要相似度”→给出各类核与正定性。
- 14.2 (核类型) → 14.3 (核化 GLM/稀疏机)：把核当基函数进入 GLM；讨论原型选择与稀疏化 (L1VM/RVM/SVM)。

- 14.3 → 14.4 (核技巧) : 从“显式核化特征”提升到“**算法级替换内积**”，推广到 1NN、K-medoids、岭回归、kPCA。
- 14.4 → 14.5 (SVM) : 在核技巧基础上，**换损失为合页并施加大间隔**，得到**稀疏对偶解**；扩展到回归/多类/调参。
- 14.5 → 14.6 (方法比较) : 从理论与实践维度（凸性、稀疏、概率、多类、速度）横评 L2VM/L1VM/RVM/SVM/GP，**给选型建议**。
- 14.2 对比 14.7: 前者讲 Mercer 核 (判别式)；后者讲 平滑核 (生成式)；二者“同名不同意”，应用面向不同任务，但都以“相似性/邻域”思想为核心。
- 14.7 (KDE/核回归) ↔ 14.4 (kPCA/核化线性) : 一边从概率角度“堆核”，一边从线性代数角度“替内积”；两条线汇合于“**只需相似度**”。

### Mermaid (可选)

```

flowchart TB
    A[14.1 引言: 用相似度刻画对象] --> B[14.2 核函数与正定性]
    B --> C[14.3 核化GLM & 稀疏机(L1VM/RVM/SVM前身)]
    C --> D[14.4 核技巧: NN/K-medoids/岭回归/kPCA]
    D --> E[14.5 SVM: 合页损失+大间隔+对偶稀疏]
    E --> F[14.6 对比: L2VM/L1VM/RVM/SVM/GP 选型]
    B --> G[14.7 平滑核: KDE/核回归/LOESS]
    G -. 生成式 .-> F
    G -. 判别式 .-> F
  
```

## ③ 本章总结

- **统一主题**: 核方法用“**相似度**”把复杂对象连接到可计算的学习器。
  - **判别式线**: 选一个**正定核**，用**核技巧**把线性算法核化；通过**稀疏化与大间隔**获得强泛化与高效预测 (SVM/L1VM/RVM/kPCA...)。
  - **生成式线**: 用**平滑核**把样本“抹开”求和，得到  $p(x)$  与  $f(x)$  (KDE/核回归/LOESS)，并能推导出 KNN。
- **关键抉择**
  - **核类型与参数** (RBF 带宽/Matérn 平滑度/TF-IDF 权重等) 极其重要；RBF 的  $\gamma$  与 SVM 的  $C$  强耦合，需标准化后做**交叉验证**或用**EB/边际似然** (GP/L2VM/RVM)。
  - **是否需要概率与不确定度**: 要就选 GP/RVM/L1VM/L2VM；SVM 需后标定且常不稳。
  - **效率**: 预测速度靠**稀疏性** (SVM/RVM/L1VM)；训练速度: 线性 SVM 可  $O(N)$ ，RVM 贪心常快于  $\ell_1$ ；GP/L2VM 通常  $O(N^3)$ 。
  - **核是否必须正定**: SVM/GP 必须；L1VM/L2VM/RVM 可更灵活地用**非 Mercer 相似度**。

## • 实践指南

- 特征本就高维稀疏（文本/基因）：先试**线性核**。
- 追求**小模型+快预测**：首选 RVM / L1VM。
- 需要**可靠概率/主动学习/控制**：GP。
- **结构化输出**（序列/匹配）：SVM 生态成熟、计算优势明显。
- 无监督/密度/平滑：**KDE/核回归/LOESS**；大数据可用**紧支撑核 + 近邻索引**。

一句话：**核方法=相似度驱动的学习框架**。判别式用**核技巧与稀疏/大间隔**，生成式用**平滑核作局部平均**。合理选核与带宽/正则，是把核方法用好的关键。

请用通俗易懂的话总结本章，使我能够理解

已思考 28s

下面把整章“核方法”用尽量接地气的话串起来，让你“有画面、有抓手”。

## 一句话版

**核 = 相似度。**

本章教你：用“相似度”代替“原始特征”，把简单的线性方法变成强大的非线性方法（判别式线）；以及用“邻近加权”的方式估计密度与做回归（生成式线）。难点不是公式，而是**选哪种核、带多大平滑、正则多强**。

## 1. 核到底是什么？

- 就把核看成“**两个对象有多像**”的评分函数。
- 常见的“像不像”：
  - **RBF/高斯核**：距离越近越像；只有一个关键旋钮：**带宽  $\sigma$** （相当于“多远算近”）。
  - **线性核**：普通内积；对文本这类**原本就高维**的特征常够用。
  - **多项式核**：允许“特征 $\times$ 特征”，相当于加了高阶项。
  - **TF-IDF + 余弦**：专为文档相似度。
  - **字符串/树/金字塔匹配核**：对序列、解析树、图像的“词袋”很有用。

- 只要核满足“**正定**”（Mercer 条件），理论上等价于把数据**隐式抬到一个超大维空间**做线性学习，算起来却不用真的去那儿。
- 

## 2. 核技巧：把“线性算法”一键变“非线性”

诀窍：凡是算法里出现内积  $x^\top x'$  的地方，都**替换成核**  $\kappa(x, x')$ 。于是：

- 1NN 的**距离**可以用核来算 → 能比较“结构化对象”（字符串、图像特征集）。
  - K-medoids（基于样本做中心）能核化。
  - 岭回归**切到对偶形式后能核化（训练花  $O(N^3)$ ，预测用到所有样本，若能稀疏就快）。
  - PCA → kPCA：把协方差的特征向量换成Gram 矩阵的特征向量，就能做**非线性降维**（记得先把 Gram 矩阵中心化）。
- 

## 3. 核化的广义线性模型 (GLM)

把核当“**基函数**”，在若干“原型点”周围起作用，就能让**逻辑回归/线性回归变成弯曲的边界/曲线**。

原型点怎么选？直接把**所有训练样本都当原型**，再用**稀疏化**挑关键的：

- L1VM： $\ell_1$  正则，天然稀疏；多类用 Group Lasso；调参靠 CV。
  - RVM：贝叶斯 ARD/SBL，通常**最稀疏、预测最快**；还能给概率；核参数用**经验贝叶斯学**。
  - L2VM： $\ell_2$  正则，不稀疏，但稳；也能给概率。
- 

## 4. SVM：核 + 合页损失 + 大间隔

- 想法**：找一条能分开两类且**离最近样本尽量远**的直线/超平面（**大间隔**）。
- 损失**：用**合页损失**（错了/离边界太近就罚），得到**稀疏对偶解**，只依赖少数**支持向量**。
- 回归版**：在预测曲线两侧留一条 $\epsilon$  管，在管里误差不罚；只有管外的点成为“**支持向量**”。
- 多分类**：常用 OvR/OvO 拼起来，会有**歧义/不平衡/开销**问题。
- 概率**：原生没有，只能做 Platt 标定，但校准常一般。
- 调参**：RBF 的  $\gamma$  ( $=1/2\sigma^2$ ) 和 C 强耦合： $\gamma$  大（核窄） $\rightarrow$  C 要小；先**标准化特征**再做网格 CV。

## 5. 生成式的“平滑核”：KDE / 核回归 / LOESS

这类“核”是权重函数（满足积分=1、均值=0），关注“**局部平均**”：

- **KDE (核密度估计)**：把每个样本涂成一个小鼓包，**相加平均**得到  $\hat{p}(x)$ 。
  - 方窗/Epanechnikov/Tri-cube：**紧支撑**（超过一定半径直接 0，计算快）；
  - **高斯**：处处非零，最平滑。
  - **关键旋钮**：**带宽 h**（太小抖动、太大过平滑）。
- **KDE  $\Rightarrow$  KNN**：若把体积自适应扩到刚好含 **K 个邻居**，可推出  $p(y = c|x) = K$  个邻居里 c 的占比——就是 **KNN** 的概率解释。
- **核回归 (Nadaraya–Watson)**：预测 = **附近样本的加权平均**（权重=核相似度）。
- **LOESS/LOWESS**：在**局部做加权线性回归**，比简单加权平均更灵活。

注意：平滑核与前面的 Mercer 核不是一回事；前者做生成式/局部平均，后者做核技巧/判别式。

## 6. 方法横评与选型顺口溜

- **RVM**：通常**最稀疏、预测最快，有概率**；训练常用贪心，实践很快。
- **GP (下章)**：**概率最靠谱**（含不确定度），但**训练最慢**；核参数可用**边际似然自动学习**。
- **SVM**：效果强、生态好，**结构化输出**任务有优势；但**非概率**、多类要拼装、调参靠 CV。
- **L1VM**：稀疏、简单；CV 调参。
- **L2VM**：稳健但不稀疏。
- **线性核**：文本/基因表达这类**原本就高维稀疏**的特征，往往足够好且快。

怎么选？

- **要可靠概率/不确定度**（主动学习、控制）：GP 或 RVM。
- **要小模型/推理快**：RVM / L1VM。
- **结构化输出**（序列、匹配）：常用 SVM。
- 特征已超高维：先试**线性核**。
- 做密度/平滑/局部建模：**KDE / 核回归 / LOESS**（大数据优先**紧支撑核+近邻索引**）。

## 7. 你需要记住的 5 件事

1. **核=相似度。** 正定核让线性方法拥有非线性表达力；平滑核让你做局部平均与密度估计。
2. **核技巧：** 把内积换成核；很多算法都能“核化”（NN、K-medoids、岭回归、PCA...）。
3. **稀疏 = 预测快。** SVM/RVM/L1VM 只靠少数训练点作预测。
4. **调参比公式重要：** RBF 的带宽/ $\gamma$  与正则 C 要联动搜索；平滑核的带宽 h 决定一切。
5. **选对工具：** 要概率→GP/RVM；要速度→RVM/线性核；要结构化→SVM；要无监督密度/回归→KDE/LOESS。

这样看，本章就是：**用“像不像”驱动学习。** 该抬维的抬维（核技巧），该局部的局部（平滑核），把模型的“形与度”交给相似度和带宽/正则去控制。