CHAPTER

NONPARAMETRIC ESTIMATION 16

CHAPTER CONTENTS

So far, estimation with parametric methods was discussed, which is useful when an appropriate parametric model is available (Fig. 16.1(a)). However, if the true probability density is highly complicated, it may be difficult to prepare an appropriate parametric model and then parametric methods do not work well (Fig. 16.1(b)). In this chapter, *nonparametric estimation* methods are introduced that do not use parametric models.

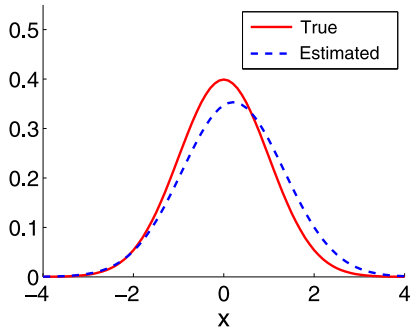
16.1 HISTOGRAM METHOD

The simplest nonparametric method would be the *histogram method*. In the histogram method, the pattern space $\mathcal{X}$ is partitioned into several *bins*. Then, in each bin, the probability density is approximated by a constant proportional to the number of training samples that fall into the bin. An example of the histogram method is illustrated in Fig. 16.2, and its MATLAB code is provided in Fig. 16.3. The graph shows that the profile of a complicated probability density function can be captured well by the histogram method.

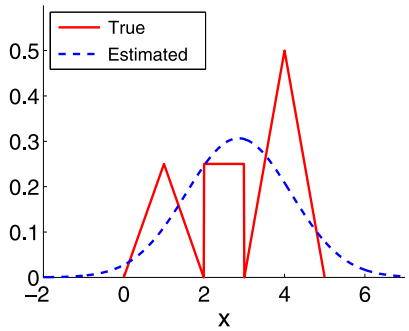
However, the histogram method has several drawbacks:

- Probability densities become discontinuous across different bins.
- Appropriately determining the shape and size of bins is not straightforward (see Fig. 16.4).

An Introduction to Statistical Machine Learning. DOI: 10.1016/B978-0-12-802121-7.00027-3



(a) When the true probability density is simple



(b) When the true probability density is complicated

FIGURE 16.1

Examples of Gaussian MLE.

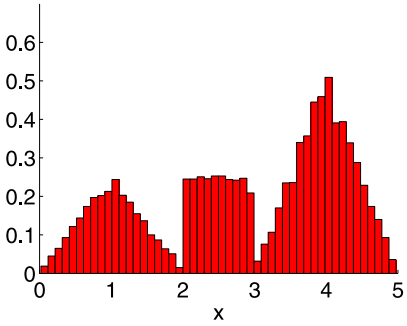


FIGURE 16.2

Example of histogram method.

- Naive splitting of the pattern space such as the equidistant grid exponentially grows the number of bins with respect to the input dimensionality d.

In this chapter, nonparametric methods that can address these issues are introduced.

16.2 PROBLEM FORMULATION

Let us consider the problem of estimating $p(\boldsymbol{x}')$, the value of probability density at point $\boldsymbol{x}'$. Let $\mathcal{R}$ be a region in the pattern space $\mathcal{X}$ that contains the point of interest

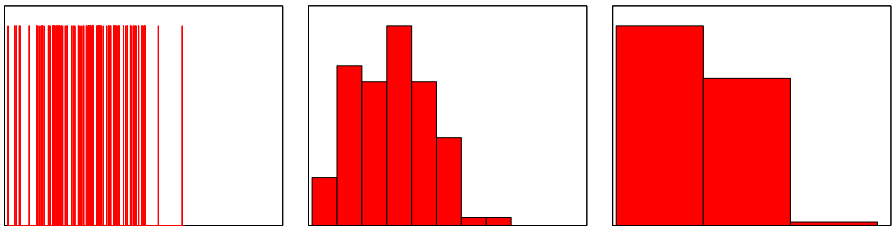
```
n=10000; x=myrand(n); s=0.1; b=[0:s:5];
figure(1); clf; hold on
a=histc(x,b); bar(b,a/s/n,'histc')
```

```
function x=myrand(n)

x=zeros(1,n); u=rand(1,n);
t=(0<=u & u<1/8); x(t)=sqrt(8*u(t));
t=(1/8<=u & u<1/4); x(t)=2-sqrt(2-8*u(t));
t=(1/4<=u & u<1/2); x(t)=1+4*u(t);
t=(1/2<=u & u<3/4); x(t)=3+sqrt(4*u(t)-2);
t=(3/4<=u & u<=1); x(t)=5-sqrt(4-4*u(t));
```

FIGURE 16.3

MATLAB code for inverse transform sampling (see Section 19.3.1) for probability density function shown in Fig. 16.1(b). The bottom function should be saved as "`myrand.m`."



(a) Too small bin width (b) Appropriate bin width (c) Too large bin width

FIGURE 16.4

Choice of bin width in histogram method.

$\boldsymbol{x}'$, and let V be its *volume*:

$$V = \int_{\mathcal{R}} \mathrm{d}\boldsymbol{x}.$$

The probability P that a pattern $\boldsymbol{x}$ falls into region $\mathcal{R}$ is given by

$$P = \int_{\mathcal{R}} p(\boldsymbol{x})\mathrm{d}\boldsymbol{x}.$$

These notations are summarized in Fig. 16.5.

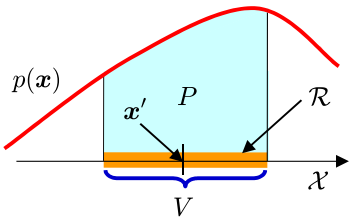


FIGURE 16.5

Notation of nonparametric methods.

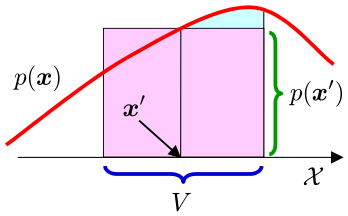


FIGURE 16.6

Probability P approximated by the size of rectangle.

The probability P may be approximated by two ways. One is to use the point of interest $\boldsymbol{x}'$ as follows (Fig. 16.6):

$$P \approx V p(\boldsymbol{x}'). \tag{16.1}$$

The other is to use k, the number of training samples that fall into the region $\mathcal{R}$, as

$$P \approx \frac{k}{n}. \tag{16.2}$$

Then combining Eq. (16.1) and Eq. (16.2) allows us to eliminate P and obtain

$$p(\boldsymbol{x}') \approx \frac{k}{nV}. \tag{16.3}$$

This is the fundamental form of nonparametric density estimation that approximates $p(\boldsymbol{x}')$, the value of probability density at point $\boldsymbol{x}'$, without using any parametric model.

The accuracy of approximation (16.3) depends on the accuracy of Eq. (16.1) and Eq. (16.2), which can be controlled by the choice of region $\mathcal{R}$. Let us evaluate the accuracy of Eq. (16.1) and Eq. (16.2) in terms of the choice of $\mathcal{R}$. Eq. (16.1) is exact if the probability density is constant in the region $\mathcal{R}$. Thus, if the region $\mathcal{R}$ is smaller, Eq. (16.1) may be more accurate.

Next, let us evaluate the accuracy of Eq. (16.2). The probability that k points out of n training samples fall into the region $\mathcal{R}$ follows the *binomial distribution* (see Section 3.2). Its probability mass function is given by

$$\binom{n}{k} P^k (1-P)^{n-k}, \tag{16.4}$$

where $\binom{n}{k}$ is the binomial coefficient:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}.$$

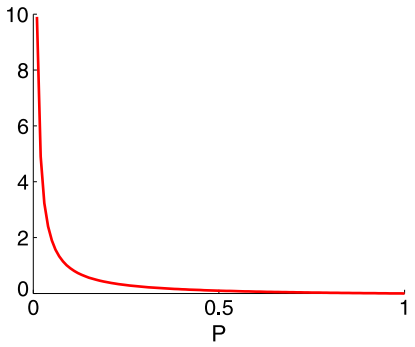


FIGURE 16.7

Normalized variance of binomial distribution.

The expectation and variance of binomial random variable k are given by

$$E[k] = nP \quad \text{and} \quad V[k] = nP(1-P).$$

Then it can be easily confirmed that the expectation of k/n agrees with true P:

$$E\left[\frac{k}{n}\right] = P.$$

However, the fact that the expectation of k/n agrees with true P does not necessarily mean k/n is a good estimator of P. Indeed, if its variance is large, k/n may be a poor estimator of P. Here, let us normalize k by nP as

$$z = \frac{k}{nP},$$

so that the expectation of z is always 1 for any P:

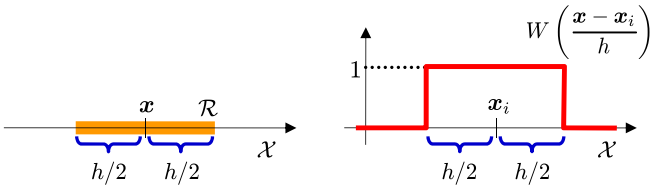
$$E[z] = \frac{E[k]}{nP} = 1.$$

The variance of z is given by

$$V[z] = \frac{V[k]}{(nP)^2} = \frac{1-P}{nP}.$$

If $V[z]$ is small, k/n will be a good approximator to P. Fig. 16.7 plots $V[z]$ as a function of P, showing that larger P gives smaller $V[z]$. P can be increased if region $\mathcal{R}$ is widen, and thus larger $\mathcal{R}$ makes Eq. (16.2) more accurate.

As explained above, Eq. (16.1) is more accurate if $\mathcal{R}$ is taken to be small, while Eq. (16.2) is more accurate if $\mathcal{R}$ is taken to be large. Thus, region $\mathcal{R}$ should be chosen



(a) Region $\mathcal{R}$ (b) Parzen window function

FIGURE 16.8

Parzen window method.

appropriately to improve the accuracy of approximation (16.3). In the following sections, two methods to determine region $\mathcal{R}$ based on training samples $\{\boldsymbol{x}_i\}_{i=1}^n$ are introduced. In Section 16.3, the volume V of region $\mathcal{R}$ is fixed, and the number of training samples k that fall into $\mathcal{R}$ is determined from data. On the other hand, in Section 16.4, k is fixed, and the volume V of region $\mathcal{R}$ is determined from data.

16.3 KDE

In this section, the volume V of region $\mathcal{R}$ is fixed, and the number of training samples k that fall into $\mathcal{R}$ is determined from data.

16.3.1 PARZEN WINDOW METHOD

As region $\mathcal{R}$, let us consider the *hypercube* with edge length h centered at $\boldsymbol{x}$ in region $\mathcal{R}$ (Fig. 16.8(a)). Its volume V is given by

$$V = h^d, \tag{16.5}$$

where d is the dimensionality of the pattern space. The number of training samples falling into region $\mathcal{R}$ is expressed as

$$k = \sum_{i=1}^{n} W\left(\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h}\right), \tag{16.6}$$

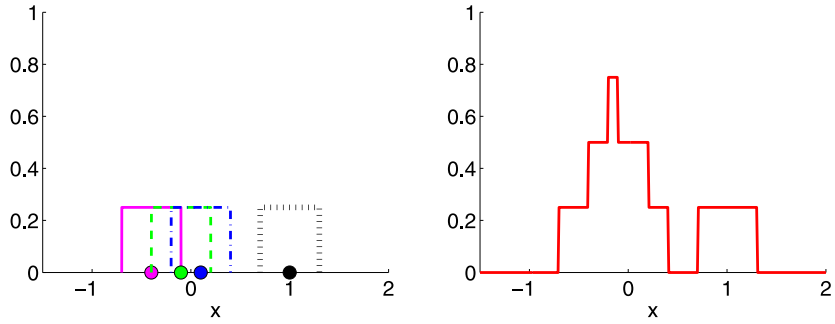
where $W(\boldsymbol{x})$ is called the *Parzen window function* defined for

$$\boldsymbol{x} = (x^{(1)}, \ldots, x^{(d)})^\top$$

as follows (Fig. 16.8(b)):

$$W(\boldsymbol{x}) = \begin{cases} 1 & \max\limits_{i=1,\ldots,d} |x^{(i)}| \leq \dfrac{1}{2}, \\ 0 & \text{otherwise.} \end{cases}$$

h is called the *bandwidth* of the Parzen window function.



(a) Each Parzen window function (b) Parzen window estimator

FIGURE 16.9

Example of Parzen window method.

Substituting Eq. (16.5) and Eq. (16.6) into Eq. (16.3) gives the following density estimator:

$$\widehat{p}_{\text{Parzen}}(\boldsymbol{x}) = \frac{1}{nh^d} \sum_{i=1}^{n} W\left(\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h}\right).$$

This estimator called the *Parzen window method* and its numerical behavior are illustrated in Fig. 16.9. The result resembles that of the histogram method, but the bin widths are determined adaptively based on the training samples. However, discontinuity of estimated densities across different bins still remains in the Parzen window method.

16.3.2 SMOOTHING WITH KERNELS

The problem of discontinuity can be effectively overcome by KDE, which uses a smooth *kernel function* $K(\boldsymbol{x})$ instead of the Parzen window function:

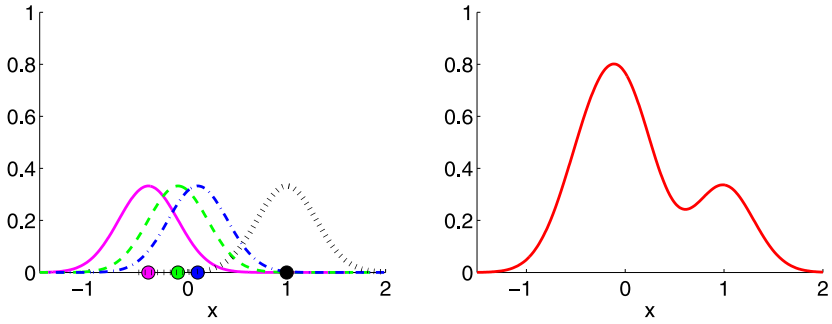
$$\widehat{p}_{\text{KDE}}(\boldsymbol{x}) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h}\right).$$

Note that the kernel function should satisfy

$$\forall \boldsymbol{x} \in \mathcal{X}, \quad K(\boldsymbol{x}) \geq 0, \quad \text{and} \quad \int_{\mathcal{X}} K(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} = 1.$$

The *Gaussian kernel* is a popular choice as a kernel function:

$$K(\boldsymbol{x}) = \frac{1}{(2\pi)^{\frac{d}{2}}} \exp\left(-\frac{\boldsymbol{x}^\top \boldsymbol{x}}{2}\right),$$



(a) Each Gaussian kernel function (b) Kernel density estimator

FIGURE 16.10

Example of Gaussian KDE. Training samples are the same as those in Fig. 16.9.

where the bandwidth h corresponds to the standard deviation of the Gaussian density function. An example of Gaussian KDE is illustrated in Fig. 16.10, showing that a nice smooth density estimator is obtained.

A generalized KDE,

$$\widehat{p}_{\mathrm{KDE}}(\boldsymbol{x}) = \frac{1}{n \det(\boldsymbol{H})} \sum_{i=1}^{n} K\left(\boldsymbol{H}^{-1}(\boldsymbol{x} - \boldsymbol{x}_i)\right), \tag{16.7}$$

may also be considered, where $\boldsymbol{H}$ is the $d \times d$ positive definite matrix called the *bandwidth matrix*. If $K(\boldsymbol{x})$ is the Gaussian function, $\boldsymbol{H}\boldsymbol{H}^{\top}$ corresponds to the variance-covariance matrix of the Gaussian density function.

16.3.3 BANDWIDTH SELECTION

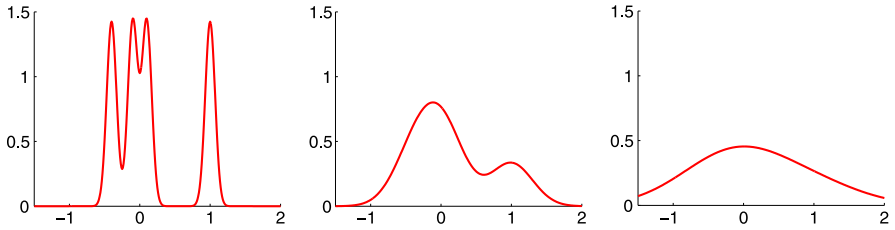
The estimator $\widehat{p}_{\mathrm{KDE}}(\boldsymbol{x})$ obtained by KDE depends on the bandwidth h (Fig. 16.11). Here, data-driven methods to choose h are introduced.

For generalized KDE (16.7), let us consider a diagonal bandwidth matrix $\boldsymbol{H}$:

$$\boldsymbol{h} = \mathrm{diag}\left(h^{(1)}, \ldots, h^{(d)}\right),$$

where d denotes the dimensionality of input $\boldsymbol{x}$. When the true probability distribution is Gaussian, the optimal bandwidth is given asymptotically as follows [90, 93]:

$$\widehat{h}^{(j)} = \left(\frac{4}{(d+2)n}\right)^{\frac{1}{d+4}} \sigma^{(j)},$$



(a) $h = 0.07$ (b) $h = 0.3$ (c) $h = 0.7$

FIGURE 16.11

Choice of kernel bandwidth h in KDE.

```
n=500; x=myrand(n); x2=x.^2; hs=[0.01 0.1 0.5]; t=5;
d2=repmat(x2,[n 1])+repmat(x2',[1 n])-2*x'*x;
v=mod(randperm(n),t)+1;
for i=1:length(hs)
  hh=2*hs(i)^2; P=exp(-d2/hh)/sqrt(pi*hh);
  for j=1:t
    s(j,i)=mean(log(mean(P(v~=j,v==j))));
  end
end
[dum,a]=max(mean(s)); h=hs(a); hh=2*h^2;
ph=mean(exp(-d2/hh)/(sqrt(pi*hh)));
figure(1); clf; plot(x,ph,'r*'); h
```

FIGURE 16.12

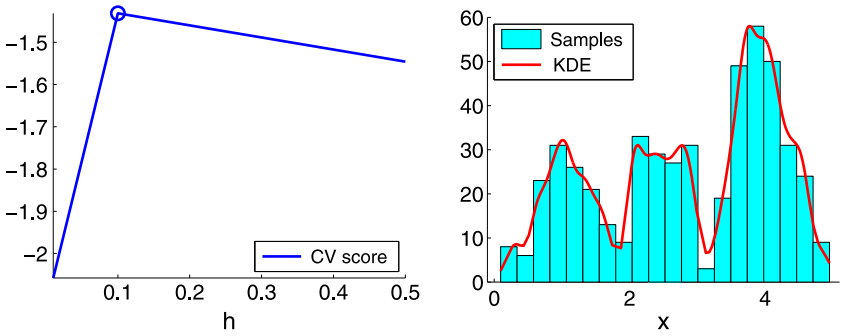
MATLAB code for Gaussian KDE with bandwidth selected by likelihood cross validation. A random number generator "`myrand.m`" shown in Fig. 16.3 is used.

where $\sigma^{(j)}$ denotes the standard deviation of the jth element of $\boldsymbol{x}$. Since $\sigma^{(j)}$ may be unknown in practice, it is estimated from samples as

$$\widehat{\sigma}^{(j)} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(x_i^{(j)} - \frac{1}{n}\sum_{i=1}^{n} x_i^{(j)}\right)^2}.$$

This is called *Silverman's bandwidth selector*.

Although Silverman's bandwidth selector is easy to implement, its validity is only guaranteed when the true probability distribution is Gaussian. For more flexible model selection, *cross validation* explained in Section 14.4 is highly useful. A



(a) CV score (b) Estimated density

FIGURE 16.13

Example of Gaussian KDE with bandwidth selected by likelihood cross validation.

MATLAB code of likelihood cross validation for Gaussian KDE is provided in Fig. 16.12, and its behavior is illustrated in Fig. 16.13.

16.4 NNDE

In KDE explained above, the volume V of region $\mathcal{R}$ is fixed, and the number of training samples k that fall into region $\mathcal{R}$ is determined from data. In this section, an alternative method is introduced, where k is fixed and the volume V of region $\mathcal{R}$ is determined from data.

16.4.1 NEAREST NEIGHBOR DISTANCE

As region $\mathcal{R}$, let us consider the *hypersphere* with radius r centered at $\boldsymbol{x}$. Then the volume V of region $\mathcal{R}$ is given by

$$V = \frac{\pi^{\frac{d}{2}} r^d}{\Gamma(\frac{d}{2}+1)},$$

where $\Gamma(\cdot)$ is the *gamma function* (see Section 4.3).

Setting the radius r at the minimum number such that k training samples are included in the hypersphere, Eq. (16.3) immediately gives the following density estimator:

$$\widehat{p}_{\mathrm{KNN}}(\boldsymbol{x}) = \frac{k\Gamma(\frac{d}{2}+1)}{n\pi^{\frac{d}{2}} r^d}. \tag{16.8}$$

This is called NNDE.

k controls the smoothness of the density estimator, and it would be natural to increase k as the number of training samples n grows. Indeed, to guarantee

```
n=500; x=myrand(n); x2=x.^2;
ks=[10 50 100]; t=5; g=gamma(3/2);
d2=repmat(x2,[n 1])+repmat(x2',[1 n])-2*x'*x;
v=mod(randperm(n),t)+1;
for j=1:t
  S=sort(d2(v~=j,v==j));
  for i=1:length(ks)
    k=ks(i); r=sqrt(S(k+1,:));
    s(j,i)=mean(log(k*g./(sum(v~=j)*sqrt(pi)*r)));
  end
end
[dum,a]=max(mean(s)); k=ks(a);
m=1000; X=linspace(0,5,m);
D2=repmat(X.^2,[n 1])+repmat(x2',[1 m])-2*x'*X;
S=sort(D2); r=sqrt(S(k+1,:))'; Ph=k*g./(n*sqrt(pi)*r);
figure(1); clf; plot(X,Ph,'r*'); k
```

FIGURE 16.14

MATLAB code for NNDE with the number of nearest neighbors selected by likelihood cross validation. A random number generator "`myrand.m`" shown in Fig. 16.3 is used.

consistency of $\widehat{p}_{\mathrm{KNN}}(\boldsymbol{x})$ (i.e., $\widehat{p}_{\mathrm{KNN}}(\boldsymbol{x})$ converges to $p(\boldsymbol{x})$ as n tends to infinity), k should satisfy the following conditions [70]:

$$\lim_{n\to\infty} k = \infty \quad \text{and} \quad \lim_{n\to\infty} \frac{k}{n} = 0.$$

For example, $k = \sqrt{n}$ satisfy the above condition.

However, *likelihood cross validation* explained in Section 14.4 is more useful in practice. A MATLAB code of likelihood cross validation for NNDE is provided in Fig. 16.14, and its behavior is illustrated in Fig. 16.15.

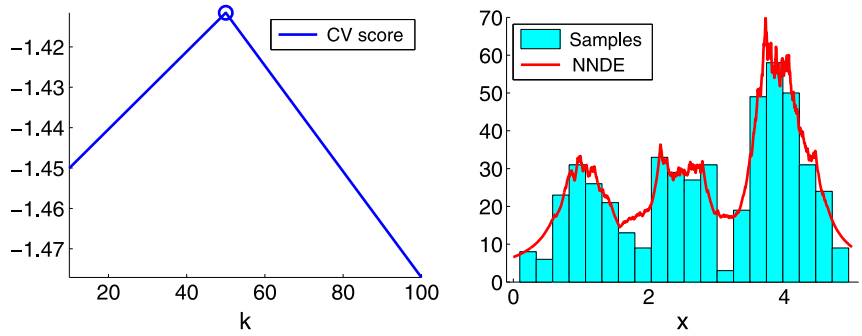
16.4.2 NEAREST NEIGHBOR CLASSIFIER

Finally, NNDE is applied to estimating the class-conditional probability density $p(\boldsymbol{x}|y)$, and pattern recognition is performed based on the *MAP rule* (Section 11.3.1).

Pattern Recognition with Nearest Neighbor Distance

From Bayes' theorem (see Section 5.4), the class-posterior probability $p(y|\boldsymbol{x})$ is expressed as

$$p(y|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|y)p(y)}{\sum_{y'=1}^{c} p(\boldsymbol{x}|y')p(y')} \propto p(\boldsymbol{x}|y)p(y).$$



(a) CV score (b) Estimated density

FIGURE 16.15

Example of NNDE with the number of nearest neighbors selected by likelihood cross validation.

If NNDE with $k = 1$ is used, the class-conditional probability density $p(\boldsymbol{x}|y)$ is estimated as

$$p(\boldsymbol{x}|y) \approx \frac{\Gamma(\frac{d}{2}+1)}{n_y \pi^{\frac{d}{2}} r_y^d},$$

where r_y denotes the distance between input pattern $\boldsymbol{x}$ and the nearest training sample among training samples in class y, and n_y denotes the number of training samples in class y. If the class-prior probability $p(y)$ is approximated as

$$p(y) \approx \frac{n_y}{n},$$

the class-posterior probability $p(y|\boldsymbol{x})$ is approximated as

$$p(y|\boldsymbol{x}) \approx \frac{\Gamma(\frac{d}{2}+1)}{n_y \pi^{\frac{d}{2}} r_y^d} \frac{n_y}{n} \propto \frac{1}{r_y^d}.$$

This means that, to perform pattern recognition based on NNDE, only r_y^d is necessary. Since the class y that maximizes $1/r_y^d$ is the class that minimizes r_y, the input pattern $\boldsymbol{x}$ is simply classified into the same class as the nearest training sample in NNDE-based pattern recognition. This pattern recognition method is called the *nearest neighbor classifier*.

Fig. 16.16 illustrates an example of the nearest neighbor classifier, showing that the decision boundaries can be obtained from the *Voronoi diagram*.

k-Nearest Neighbor Classifier

The nearest neighbor classifier is intuitive and easy to implement, but it is not robust against *outliers*. For example, one of the training samples has an incorrect class label; the decision regions contain an isolated region (see Fig. 16.16).

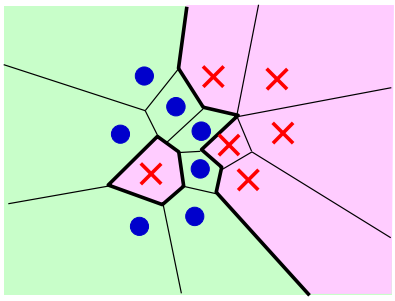


FIGURE 16.16

Example of nearest neighbor classifier.

To mitigate this problem, NNDE for $k > 1$ is useful:

$$\widehat{p}(\boldsymbol{x}|y) = \frac{k}{n_y V_y},$$

where n_y denotes the number of training samples in class y and V_y denotes the volume of the minimum hypersphere centered at $\boldsymbol{x}$ that contains k training samples.

However, the above method requires computation of hyperspheres for each class. A simpler implementation would be to find a hypersphere for training samples in all classes and to estimate the marginal probability $p(\boldsymbol{x})$ as

$$\widehat{p}(\boldsymbol{x}) = \frac{k}{nV}.$$

Since $p(\boldsymbol{x})$ can be decomposed as

$$p(\boldsymbol{x}) = \sum_{y=1}^{c} p(\boldsymbol{x}|y)p(y),$$

estimating the class-prior probability $p(y)$ by

$$\widehat{p}(y) = \frac{n_y}{n}$$

yields

$$\frac{k}{nV} \approx \sum_{y=1}^{c} p(\boldsymbol{x}|y)\frac{n_y}{n}.$$

Finally, the class-posterior probability $(\boldsymbol{x}|y)$ is estimated as

$$\widehat{p}(\boldsymbol{x}|y) = \frac{k_y}{n_y V}, \tag{16.9}$$

1. Split labeled training samples $\mathcal{Z} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ into t disjoint subsets of (approximately) the same size: $\{\mathcal{Z}_\ell\}_{\ell=1}^t$.
2. For each model candidate $\mathcal{M}_j$
   (a) For each split $\ell = 1, \ldots, t$
      i. Obtain a classifier $\widehat{y}_j^{(\ell)}(\boldsymbol{x})$ using model $\mathcal{M}_j$ from all training samples without $\mathcal{Z}_\ell$.
      ii. Compute the misclassification rate $J_j^{(\ell)}$ for $\widehat{y}_j^{(\ell)}(\boldsymbol{x})$ for holdout samples $\mathcal{Z}_\ell$:

$$J_j^{(\ell)} = \frac{1}{|\mathcal{Z}_\ell|} \sum_{(\boldsymbol{x}', y') \in \mathcal{Z}_\ell} I(\widehat{y}_j^{(\ell)}(\boldsymbol{x}') \neq y'),$$

      where $|\mathcal{Z}_\ell|$ denotes the number of elements in the set $\mathcal{Z}_\ell$ and $I(e) = 1$ if condition e is true and $I(e) = 0$ otherwise.
   (b) Compute the average misclassification rate J_j over all t splits:

$$J_j = \frac{1}{t} \sum_{\ell=1}^{t} J_j^{(\ell)}.$$

3. Choose the model $\mathcal{M}_{\widehat{j}}$ that minimizes the average misclassification rate:

$$\widehat{j} = \operatorname*{argmin}_j J_j.$$

4. Obtain a classifier using chosen model $\mathcal{M}_{\widehat{j}}$, from all training samples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$.

FIGURE 16.17

Algorithm of cross validation for misclassification rate.

where k_y denotes the number of training samples in the hypersphere that belong to class y. This allows us to construct a classifier in a simple way, which is called the *k-nearest neighbor classifier*.

Cross Validation for Misclassification Rate

The tuning parameter k in the k-nearest neighbor classifier can be chosen by *likelihood cross validation* for NNDE. However, in the context of pattern recognition, it would be more direct to choose k by cross validation in terms of the *misclassification rate*. The algorithm of cross validation for the misclassification rate is summarized in Fig. 16.17.

```
load digit.mat X T; [d,m,c]=size(X); X=reshape(X,[d m*c]);
Y=reshape(repmat([1:c],[m 1]),[1 m*c]);
ks=[1:10]; t=5; v=mod(randperm(m*c),t)+1;
for i=1:t
  Yh=knn(X(:,v~=i),Y(v~=i),X(:,v==i),ks);
  s(i,:)=mean(Yh~=repmat(Y(v==i),[length(ks) 1]),2);
end
[dum,a]=min(mean(s)); k=ks(a); [d,r,c]=size(T);
T=reshape(T,[d r*c]); U=reshape(knn(X,Y,T,k),[r c]);
for i=1:c, C(:,i)=sum(U==i); end, C, sum(diag(C))/sum(sum(C))
```

```
function U=knn(X,Y,T,ks)
m=size(T,2); D2=repmat(sum(T.^2,1),[size(X,2) 1]);
D2=D2+repmat(sum(X.^2,1)',[1 m])-2*X'*T; [dum,z]=sort(D2,1);
for i=1:length(ks)
  k=ks(i);
  for j=1:m
    Z=sort(Y(z(1:k,j))); g=find([1 Z(1:end-1)~=Z(2:end)]);
    [dum,a]= max([g(2:end) k+1]-g); U(i,j)=Z(g(a));
end, end
```

FIGURE 16.18

MATLAB code for k-nearest neighbor classifier with k chosen by cross validation. The bottom function should be saved as "knn.m."

Predicted class

True class \	1	2	3	4	5	6	7	8	9	0
1	200	0	0	0	0	0	0	0	0	0
2	0	193	1	0	0	0	1	4	1	0
3	0	0	195	0	3	0	0	1	1	0
4	0	0	0	191	1	2	0	0	6	0
5	0	3	4	0	187	0	1	1	2	2
6	0	2	0	0	2	195	0	0	0	1
7	0	0	1	2	0	0	192	2	3	0
8	0	1	4	1	3	0	0	186	2	3
9	0	0	0	3	0	0	1	1	195	0
0	0	1	1	0	0	0	0	0	0	198

FIGURE 16.19

Confusion matrix for 10-class classification by k-nearest neighbor classifier. $k = 1$ was chosen by cross validation for misclassification rate. The correct classification rate is 1932/2000 = 96.6%.

A MATLAB code for 10-class hand-written digit recognition (see Section 12.5) by the k-nearest neighbor classifier with k chosen by cross validation in terms of the misclassification rate is provided in Fig. 16.18. The obtained *confusion matrix* is shown in Fig. 16.19, where $k = 1$ was chosen by cross validation. The confusion matrix shows that 1932 test samples out of 2000 samples are correctly classified, meaning that the correct classification rate is

$$1932/2000 = 96.6\%.$$

On the other hand, as shown in Section 12.5, the correct classification rate for the same data set by FDA is

$$1798/2000 = 89.9\%.$$

Therefore, for this hand-written digit recognition experiment, the k-nearest neighbor classifier works much better than FDA.