CHAPTER

32 CONFIDENCE OF PREDICTION

CHAPTER CONTENTS

By the supervised learning methods introduced in Part 4, output values for arbitrary input points can be predicted. In addition to predicting output values, however, it is useful to assess the *confidence* of prediction in some applications. The probabilistic classification methods introduced in Chapter 28 allow us to learn the confidence of classification, but its range of applications was limited to classification. In this chapter, more general methods for assessing the confidence of prediction are discussed.

32.1 PREDICTIVE VARIANCE FOR ℓ_2-REGULARIZED LS

Let us consider regression by *ℓ_2-regularized LS* for *linear-in-parameter models* (see Chapter 23). More specifically, from input-output paired samples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, a predictor of output for input $\boldsymbol{x}$ is obtained as

$$\widehat{f}(\boldsymbol{x}) = \sum_{j=1}^{b} \widehat{\theta}_j \phi_j(\boldsymbol{x}) = \widehat{\boldsymbol{\theta}}^\top \boldsymbol{\phi}(\boldsymbol{x}),$$

where $\widehat{\boldsymbol{\theta}} = (\widehat{\theta}_1, \ldots, \widehat{\theta}_b)^\top$ is the vector of learned parameters given by

$$\widehat{\boldsymbol{\theta}} = (\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1} \boldsymbol{\Phi}^\top \boldsymbol{y},$$

and $\boldsymbol{\phi}(\boldsymbol{x}) = (\phi_1(\boldsymbol{x}), \ldots, \phi_b(\boldsymbol{x}))^\top$ is the vector of basis functions. $\boldsymbol{\Phi}$ is the design matrix given by

$$\boldsymbol{\Phi} = \begin{pmatrix} \phi_1(\boldsymbol{x}_1) & \cdots & \phi_b(\boldsymbol{x}_1) \\ \vdots & \ddots & \vdots \\ \phi_1(\boldsymbol{x}_n) & \cdots & \phi_b(\boldsymbol{x}_n) \end{pmatrix},$$

An Introduction to Statistical Machine Learning. DOI: 10.1016/B978-0-12-802121-7.00043-1

$\lambda \geq 0$ is the regularization parameter, $\boldsymbol{I}$ is the identity matrix, and $\boldsymbol{y} = (y_1, \ldots, y_n)^\top$ is the training output vector.

Suppose that training output vector $\boldsymbol{y}$ is given by

$$\boldsymbol{y} = \boldsymbol{\Phi}\boldsymbol{\theta}^* + \boldsymbol{\epsilon}.$$

Here, $\boldsymbol{\theta}^*$ is the *true* parameter and $\boldsymbol{\epsilon}$ is a noise vector such that

$$\mathbb{E}[\boldsymbol{\epsilon}] = \boldsymbol{0} \quad \text{and} \quad \mathbb{E}[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top] = \sigma^2 \boldsymbol{I},$$

where $\mathbb{E}$ denotes the expectation over $\boldsymbol{\epsilon}$ and σ^2 denotes the noise variance. Then the variance of prediction at a test input point $\boldsymbol{x}$,

$$V(\boldsymbol{x}) = \mathbb{E}\Big[\Big(\widehat{f}(\boldsymbol{x}) - \mathbb{E}[\widehat{f}(\boldsymbol{x})]\Big)^2\Big],$$

can be computed analytically as

$$\begin{aligned}
V(\boldsymbol{x}) &= \mathbb{E}\Big[\Big(\boldsymbol{\phi}(\boldsymbol{x})^\top \widehat{\boldsymbol{\theta}} - \boldsymbol{\phi}(\boldsymbol{x})^\top \mathbb{E}[\widehat{\boldsymbol{\theta}}]\Big)^2\Big] \\
&= \mathbb{E}\Big[\Big(\boldsymbol{\phi}(\boldsymbol{x})^\top \big((\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1}\boldsymbol{\Phi}^\top(\boldsymbol{\Phi}\boldsymbol{\theta}^* + \boldsymbol{\epsilon} - \boldsymbol{\Phi}\boldsymbol{\theta}^* - \mathbb{E}[\boldsymbol{\epsilon}])\big)\Big)^2\Big] \\
&= \mathbb{E}\Big[\Big(\boldsymbol{\phi}(\boldsymbol{x})^\top (\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1}\boldsymbol{\Phi}^\top \boldsymbol{\epsilon}\Big)^2\Big] \\
&= \boldsymbol{\phi}(\boldsymbol{x})^\top (\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1}\boldsymbol{\Phi}^\top \mathbb{E}[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top]\boldsymbol{\Phi}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1}\boldsymbol{\phi}(\boldsymbol{x}) \\
&= \sigma^2 \Big\|\boldsymbol{\Phi}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1}\boldsymbol{\phi}(\boldsymbol{x})\Big\|^2.
\end{aligned}$$

The noise variance σ^2 may be estimated as

$$\widehat{\sigma}^2 = \frac{\|\boldsymbol{U}\boldsymbol{y}\|^2}{\mathrm{tr}\,(\boldsymbol{U})},$$

where

$$\boldsymbol{U} = \boldsymbol{I} - \boldsymbol{\Phi}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1}\boldsymbol{\Phi}^\top.$$

$\mathrm{tr}\,(\boldsymbol{U})$ is often called the *effective number of parameters*. When $\lambda = 0$, the above $\widehat{\sigma}^2$ is reduced to the standard unbiased estimator of the noise variance, given by the sum of squared residuals divided by the number of parameters.

A MATLAB code for analytic computation of predictive variance is provided in Fig. 32.1, and its behavior is illustrated in Fig. 32.2. This shows that, depending on the density of training samples, confidence intervals are adaptively estimated.

Note that the above analytic computation technique can be extended to any estimator $\widehat{\boldsymbol{\theta}}$ that is linear with respect to $\boldsymbol{y}$. It can also be applied to correlated noise, as long as the variance-covariance matrix of the noise can be estimated.

```
n=50; x=linspace(-3,3,n)'; %x=randn(n,1);
N=1000; X=linspace(-3,3,N)';
pix=pi*x; y=sin(pix)./(pix)+0.1*x+0.2*randn(n,1);
x2=x.^2; xx=repmat(x2,1,n)+repmat(x2',n,1)-2*x*x';
hhs=2*[0.03 0.3 3].^2; ls=[0.0001 0.1 100];
m=5; u=mod(randperm(n),m)+1;
for hk=1:length(hhs)
  hh=hhs(hk); k=exp(-xx/hh);
  for i=1:m
    ki=k(u~=i,:); kc=k(u==i,:); yi=y(u~=i); yc=y(u==i);
    for lk=1:length(ls)
    t=(ki'*ki+ls(lk)*eye(n))\(ki'*yi);
    g(hk,lk,i)=mean((yc-kc*t).^2);
end, end, end
[gl,ggl]=min(mean(g,3),[],2); [ghl,gghl]=min(gl);
L=ls(ggl(gghl)); HH=hhs(gghl);
K=exp(-(repmat(X.^2,1,n)+repmat(x2',N,1)-2*X*x')/HH);
k=exp(-xx/HH); Q=inv(k^2+L*eye(n)); Qk=Q*k'; t=Qk*y; F=K*t;
V=sum((K*Q*K').^2,2)*sum((y-k'*t).^2)/(N-sum(sum(k.*Qk)));

figure(1); clf; hold on; axis([-2.8 2.8 -0.7 1.7]);
errorbar(X,F,sqrt(V),'y-'); plot(X,mean(F,2),'g-');
plot(x,y,'bo');
```

FIGURE 32.1

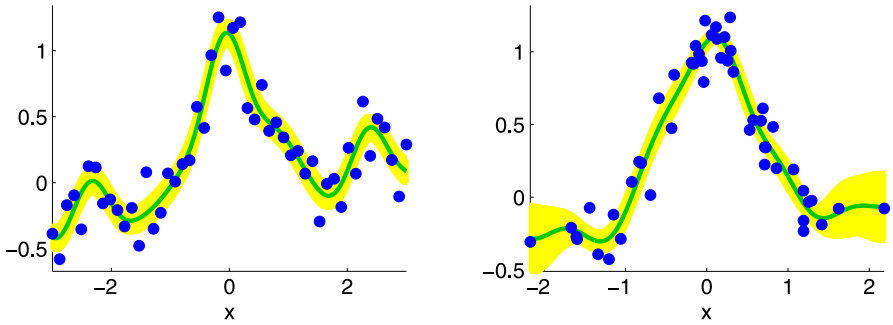
MATLAB code for analytic computation of predictive variance.

32.2 BOOTSTRAP CONFIDENCE ESTIMATION

The above analytic computation approach is useful to assess the confidence of prediction. However, its range of applications is rather limited due to the linearity assumption. Here, a more general approach based on the *bootstrap* technique (see Section 9.3.2) is introduced, which can be applied to *any* learning methods for assessing any statistics beyond the variance.

More specifically, from the original set of samples $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, n pseudosamples $\mathcal{D}' = \{(\boldsymbol{x}_i', y_i')\}_{i=1}^n$ are generated by *sampling with replacement* (see Fig. 3.3). Then, from the bootstrap samples $\mathcal{D}' = \{(\boldsymbol{x}_i', y_i')\}_{i=1}^n$, a predictor $\widehat{f}(\boldsymbol{x})$ is computed. This resampling and prediction procedure is repeated many times and its variance (or any statistics such as *quantiles*) is computed.

A MATLAB code for bootstrap-based confidence estimation is provided in Fig. 32.3, and its behavior is illustrated in Fig. 32.4. This shows that similar results to Fig. 32.2 can be numerically obtained by bootstrapping.



(a) Uniform inputs (b) Nonuniform inputs

FIGURE 32.2

Examples of analytic computation of predictive variance. The shaded area indicates the confidence interval.

32.3 APPLICATIONS

In this section, useful applications of predictive confidence are described.

32.3.1 TIME-SERIES PREDICTION

Let us consider the problem of *time-series prediction*, i.e., given $\{y_i\}_{i=1}^{t}$, the objective is to predict y_{t+1} (Fig. 32.5).

Given that input variables $\{\boldsymbol{x}_i\}_{i=1}^{t}$ are not available in the current time-series prediction setup, a simple modeling approach is to use m previous output values $\{y_{t-i+1}\}_{i=1}^{m}$ to predict the next value y_{t+1} (Fig. 32.6):

$$\widehat{y}_{t+1} = \sum_{j=1}^{b} \theta_j \phi_j(y_{t-m+1}, \ldots, y_t).$$

This approach also allows multiple-step ahead prediction as

$$\widehat{y}_{t+2} = \sum_{j=1}^{b} \theta_j \phi_j(y_{t-m+2}, \ldots, y_t, \widehat{y}_{t+1}),$$

$$\widehat{y}_{t+3} = \sum_{j=1}^{b} \theta_j \phi_j(y_{t-m+3}, \ldots, y_t, \widehat{y}_{t+1}, \widehat{y}_{t+2}).$$

It is expected that prediction of distant future is more difficult, which may be assessed by the prediction variance.

A MATLAB code of time-series prediction by ℓ_2-regularized LS for the *linear-in-input model* (which is also referred to as the *autoregressive model* in the context

```
n=50; x0=linspace(-3,3,n)'; %x0=randn(n,1);
N=1000; X=linspace(-3,3,N)';
pix=pi*x0; y0=sin(pix)./(pix)+0.1*x0+0.2*randn(n,1);
for s=1:100
  r=ceil(n*rand(n,1));x=x0(r,:); y=y0(r);
  x2=x.^2; xx=repmat(x2,1,n)+repmat(x2',n,1)-2*x*x';
  hhs=2*[0.03 0.3 3].^2; ls=[0.0001 0.1 100];
  m=5; u=mod(randperm(n),m)+1;
  for hk=1:length(hhs)
    hh=hhs(hk); k=exp(-xx/hh);
    for i=1:m
      ki=k(u~=i,:); kc=k(u==i,:); yi=y(u~=i); yc=y(u==i);
      for lk=1:length(ls)
        t=(ki'*ki+ls(lk)*eye(n))\(ki'*yi);
        g(hk,lk,i)=mean((yc-kc*t).^2);
  end, end, end
  [gl,ggl]=min(mean(g,3),[],2); [ghl,gghl]=min(gl);
  L=ls(ggl(gghl)); HH=hhs(gghl);
  K=exp(-(repmat(X.^2,1,n)+repmat(x2',N,1)-2*X*x')/HH);
  k=exp(-xx/HH); t=(k^2+L*eye(n))\(k*y); F(:,s)=K*t;
end
figure(1); clf; hold on; axis([-2.8 2.8 -0.7 1.7]);
errorbar(X,mean(F,2),std(F,0,2),'y-');
plot(X,mean(F,2),'g-'); plot(x0,y0,'bo');
```

FIGURE 32.3

MATLAB code for bootstrap-based confidence estimation.

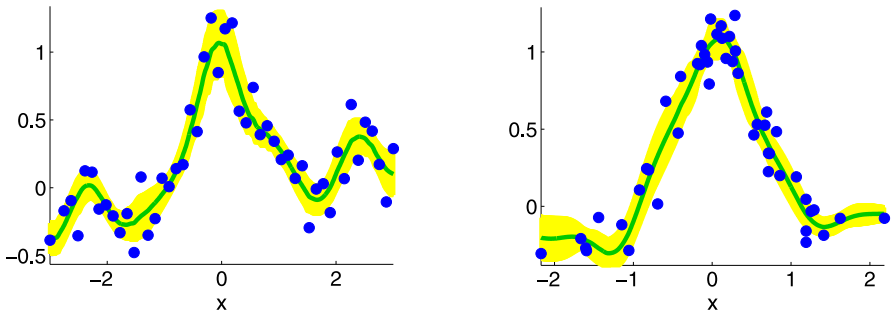
of time-series prediction),

$$\widehat{y}_{t+1} = \sum_{j=1}^{b} \theta_j y_{t-j+1},$$

is provided in Fig. 32.7. Its behavior illustrated in Fig. 32.8 shows that the confidence interval tends to grow as distant future is predicted.

32.3.2 TUNING PARAMETER OPTIMIZATION

The supervised learning methods introduced in Part 4 contain various tuning parameters such as the regularization parameter and the Gaussian bandwidth. Those tuning parameters may be optimized by *cross validation* with respect to the prediction error,



(a) Uniform inputs (b) Nonuniform inputs

FIGURE 32.4

Examples of bootstrap-based confidence estimation. The shaded area indicates the confidence interval.

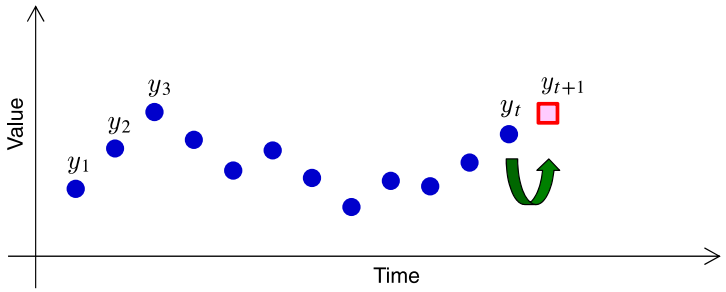


FIGURE 32.5

Problem of time-series prediction.

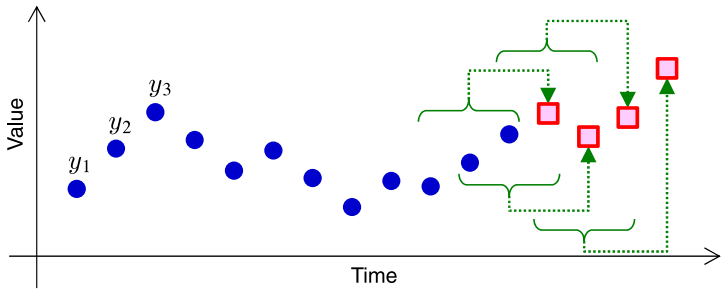


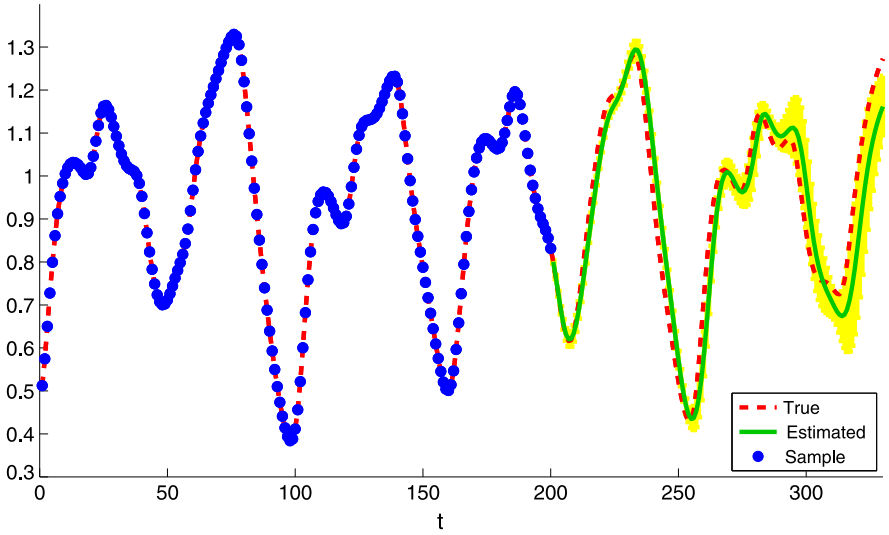
FIGURE 32.6

Time-series prediction from previous samples.

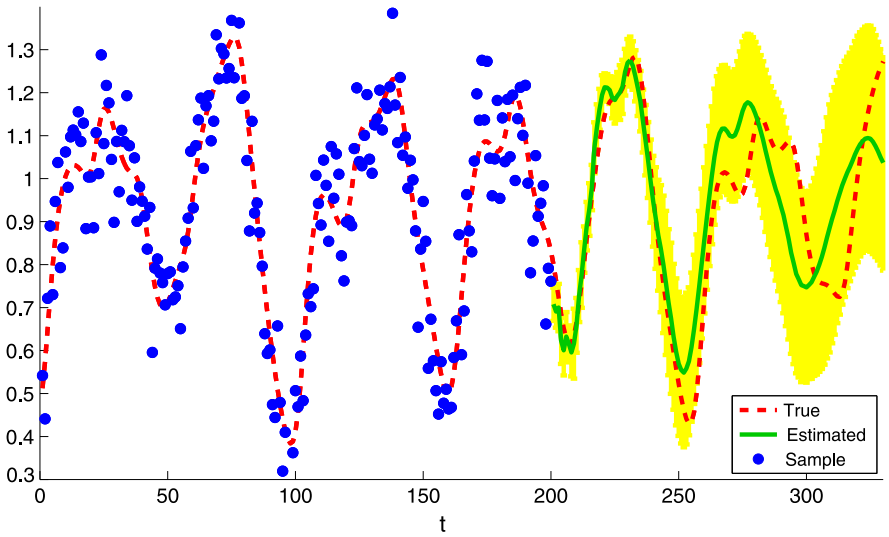
```
u=17; z(1:u+1,1)=1.2; n=200; d=20; N=n+130;
for t=u+1:u+N+d
  z(t+1)=0.9*z(t)+0.2*z(t-u)/(1+z(t-u)^10);
end
z=z(u+1:u+N+d); Y=z(d+1:end); x0=zeros(n+d-1,d);
%z=z+0.1*randn(N+d,1);
for i=1:d
  x0(i:n+d-1,i)=z(1:n+d-i);
end
x0=x0(d:end,:); y0=z(d+1:n+d);

B=100; v=zeros(N-n+d,B);
for s=1:B
  r=ceil(n*rand(n,1)); x=x0(r,:); y=y0(r);
  x2=sum(x.^2,2); xx=repmat(x2,1,n)+repmat(x2',n,1)-2*x*x';
  hhs=median(xx(:))*2*[0.5,0.2:1.5].^2; ls=[0.01 0.1 1 10];
  m=5; u=mod(randperm(n),m)+1;
  for hk=1:length(hhs)
    hh=hhs(hk); k=exp(-xx/hh);
    for i=1:m
      ki=k(u~=i,:); kc=k(u==i,:); yi=y(u~=i); yc=y(u==i);
      for lk=1:length(ls)
        t=(ki'*ki+ls(lk)*eye(n))\(ki'*yi);
        g(hk,lk,i)=mean((yc-kc*t).^2);
  end, end, end
  [gl,ggl]=min(mean(g,3),[],2); [ghl,gghl]=min(gl);
  L=ls(ggl(gghl)); HH=hhs(gghl); k=exp(-xx/HH);
  t=(k^2+L*eye(n))\(k*y); v(:,s)=[y0(n-d+1:n); zeros(N-n,1)];
  for i=1:N-n
    X=fliplr(v(i:d+i-1,s)');
    K=exp(-(repmat(sum(X.^2),1,n)+x2'-2*X*x')/HH);
    v(d+i,s)=K*t;
end, end

figure(1); clf; hold on; a=mean(v(d+1:end,:),2);
errorbar([n+1:N],a,std(v(d+1:end,:),0,2),'y-');
plot([1:N],Y,'r--'); plot([n+1:N],a,'g-');
plot([1:n],y0,'ko'); legend('','True','Estimated','Sample',4)
```

FIGURE 32.7

MATLAB code for time-series prediction by ℓ_2-regularized LS.



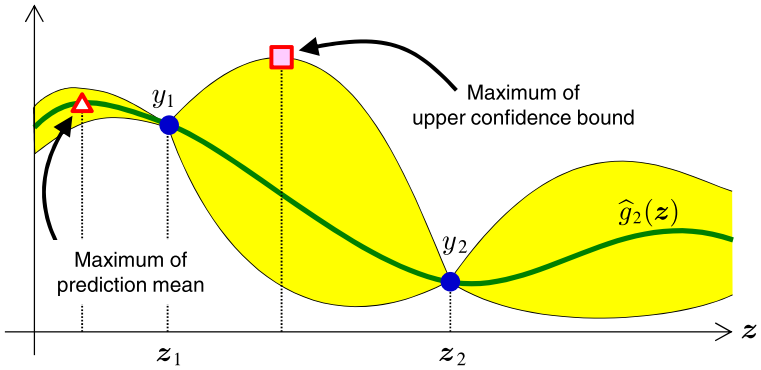
(a) Noiseless case



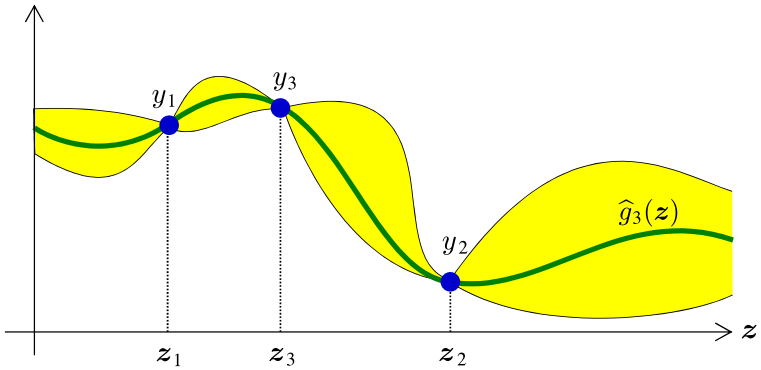
(b) Noisy case

FIGURE 32.8

Examples of time-series prediction by ℓ_2-regularized LS. The shaded areas indicate the confidence intervals.



(a) Choosing the third input point z_3



(b) After observing the third output value y_3

FIGURE 32.9

Bayesian optimization. The shaded areas indicate the confidence intervals.

as discussed in Section 23.3. However, naive grid search over tuning parameters is computationally expensive when multiple tuning parameters are optimized.

To cope with this problem, *Bayesian optimization* [12] has attracted a great deal of attention recently. Its basic idea is to find the maximizer of an unknown function $g(z)$:

$$z^* = \underset{z}{\operatorname{argmax}}\, g(z).$$

In the current setup, the function g corresponds the prediction performance (i.e., the negative of the prediction error), and z corresponds to the tuning parameters.

If cross validation is performed for some values z_i, corresponding output values $y_i = g(z_i)$ can be observed. Then the unknown function $g(z)$ can be learned from those obtained values $\{(z_i, y_i)\}_{i=1}^n$ by any regression method. The basic idea of

Bayesian optimization is to iteratively learn the function $g(z)$ from $\{(z_i, y_i)\}_{i=1}^n$ and choose the next input point z_{n+1} to observe its output value y_{n+1} (Fig. 32.9(a)).

For function $\widehat{g}_n(z)$ learned from $\{(z_i, y_i)\}_{i=1}^n$, the most naive choice of z_{n+1} would be the maximizer:

$$z_{n+1} = \operatorname*{argmax}_z \widehat{g}_n(z).$$

A more sensible approach is to take into account the uncertainty of the learned function $\widehat{g}_n(z)$, e.g., by the *upper confidence bound* [10]:

$$z_{n+1} = \operatorname*{argmax}_z \left[\widehat{g}_n(z) + k\sigma_n(z)\right],$$

where k is a constant and $\sigma_n(z)$ is the standard deviation of $\widehat{g}_n(z)$.

Other popular choices are the *probability of improvement* (PI) and the *expected improvement* (EI):

$$\text{PI:}\quad z_{n+1} = \operatorname*{argmax}_z \Pr\left(\widehat{g}_n(z) \geq (1+k) \max_{i=1,\ldots,n} y_i\right),$$

$$\text{EI:}\quad z_{n+1} = \operatorname*{argmax}_z \int_0^\infty \Pr\left(\widehat{g}_n(z) > \max_{i=1,\ldots,n} y_i + m\right) m\mathrm{d}m,$$

where $k > 0$ is a constant. When $\widehat{g}_n(z)$ is assumed to follow a Gaussian distribution, the PI and the EI (i.e., the right-hand sides of the above equations) can be computed analytically.