CHAPTER

CONSTRAINED LS REGRESSION

23

CHAPTER CONTENTS

The least squared method introduced in Chapter 22 forms the basis of various machine learning techniques. However, the naive LS method often yields *overfitting* to noisy training samples (Fig. 23.1(a)). This is caused by the fact that the model is too complicated compared with the number of available training samples. In this chapter, *constrained LS* methods are introduced for controlling the model complexity.

23.1 SUBSPACE-CONSTRAINED LS

In the LS method for linear-in-parameter model,

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

parameters $\{\theta_j\}_{j=1}^{b}$ can be determined without any constraint, implying that the entire parameter space is used for learning (see Fig. 23.2(a)). In this section, the method of *subspace-constrained LSsubspace-constrained least squares* is introduced, which imposes a subspace constraint in the parameter space:

$$\min_{\boldsymbol{\theta}} J_{\mathrm{LS}}(\boldsymbol{\theta}) \quad \text{subject to } \boldsymbol{P}\boldsymbol{\theta} = \boldsymbol{\theta},$$

where $\boldsymbol{P}$ is a $b \times b$ projection matrix such that

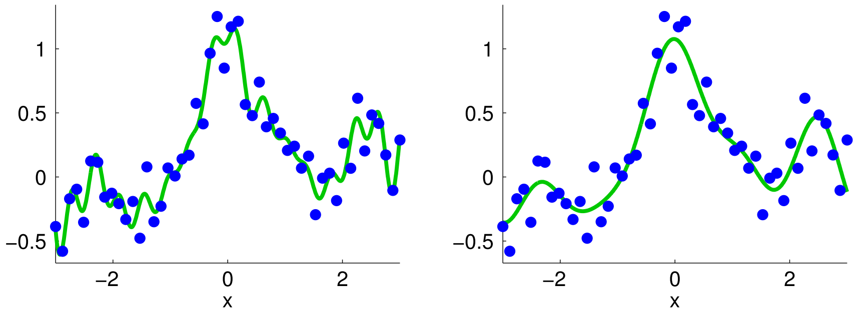
$$\boldsymbol{P}^2 = \boldsymbol{P} \quad \text{and} \quad \boldsymbol{P}^\top = \boldsymbol{P}.$$

As illustrated in Fig. 23.2(b), with the constraint $\boldsymbol{P}\boldsymbol{\theta} = \boldsymbol{\theta}$, parameter $\boldsymbol{\theta}$ is confined in the range of $\boldsymbol{P}$.

The solution $\widehat{\boldsymbol{\theta}}$ of subspace-constrained LS can be obtained simply by replacing the design matrix $\boldsymbol{\Phi}$ with $\boldsymbol{\Phi}\boldsymbol{P}$:

$$\widehat{\boldsymbol{\theta}} = (\boldsymbol{\Phi}\boldsymbol{P})^\dagger \boldsymbol{y}.$$

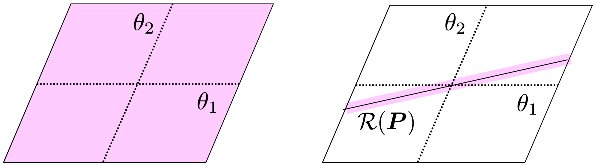
An Introduction to Statistical Machine Learning. DOI: 10.1016/B978-0-12-802121-7.00034-0



(a) Ordinary LS (b) Subspace-constrained least squared

FIGURE 23.1

Examples of LS regression for linear-in-parameter model when the noise level in training output is high. Sinusoidal basis functions $\{1, \sin\frac{x}{2}, \cos\frac{x}{2}, \sin\frac{2x}{2}, \cos\frac{2x}{2}, \ldots, \sin\frac{15x}{2}, \cos\frac{15x}{2}\}$ are used in ordinary LS, while its subset $\{1, \sin\frac{x}{2}, \cos\frac{x}{2}, \sin\frac{2x}{2}, \cos\frac{2x}{2}, \ldots, \sin\frac{5x}{2}, \cos\frac{5x}{2}\}$ is used in the subspace-constrained LS method.



(a) Ordinary LS (b) Subspace-constrained LS

FIGURE 23.2

Constraint in parameter space.

A MATLAB code for subspace-constrained LS is provided in Fig. 23.3, where a linear-in-parameter model with sinusoidal basis functions,

$$\left\{1, \sin\frac{x}{2}, \cos\frac{x}{2}, \sin\frac{2x}{2}, \cos\frac{2x}{2}, \ldots, \sin\frac{15x}{2}, \cos\frac{15x}{2}\right\},$$

is constrained to be confined in the subspace spanned by

$$\left\{1, \sin\frac{x}{2}, \cos\frac{x}{2}, \sin\frac{2x}{2}, \cos\frac{2x}{2}, \ldots, \sin\frac{5x}{2}, \cos\frac{5x}{2}\right\}.$$

The behavior of subspace-constrained LS is illustrated in Fig. 23.1(b), showing that the constraint effectively contributes to mitigating overfitting. Although the projection matrix $\boldsymbol{P}$ is manually determined in the above example, it may be determined in a data-dependent way, e.g., by PCA introduced in Section 35.2.1 or *multitask learning with the trace norm* introduced in Section 34.3.

```
n=50; N=1000; x=linspace(-3,3,n)'; X=linspace(-3,3,N)';
pix=pi*x; y=sin(pix)./(pix)+0.1*x+0.2*randn(n,1);

p(:,1)=ones(n,1); P(:,1)=ones(N,1);
for j=1:15
  p(:,2*j)=sin(j/2*x); p(:,2*j+1)=cos(j/2*x);
  P(:,2*j)=sin(j/2*X); P(:,2*j+1)=cos(j/2*X);
end
t1=p\y; F1=P*t1;
t2=(p*diag([ones(1,11) zeros(1,20)]))\y; F2=P*t2;

figure(1); clf; hold on; axis([-2.8 2.8 -0.8 1.2]);
plot(X,F1,'g-'); plot(X,F2,'r--'); plot(x,y,'bo');
legend('LS','Subspace-Constrained LS');
```

FIGURE 23.3

MATLAB code for subspace-constrained LS regression.

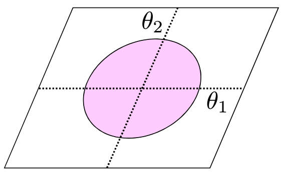


FIGURE 23.4

Parameter space in ℓ_2-constrained LS.

23.2 ℓ_2-CONSTRAINED LS

In the subspace-constrained LS method, the constraint is given by a projection matrix $\boldsymbol{P}$. However, since $\boldsymbol{P}$ has many degrees of freedom, it is not easy to handle in practice. In this section, an alternative approach called *ℓ_2-constrained LSℓ_2-constrained least squares* is introduced:

$$\min_{\boldsymbol{\theta}} J_{\mathrm{LS}}(\boldsymbol{\theta}) \quad \text{subject to } \|\boldsymbol{\theta}\|^2 \leq R^2, \tag{23.1}$$

where $R \geq 0$. As illustrated in Fig. 23.4, parameters are searched within an origin-centered hypersphere in the ℓ_2-constrained LS method, where R denotes the radius of the hypersphere.

Let us consider the constrained optimization problem,

$$\min_{\boldsymbol{t}} \; f(\boldsymbol{t}) \;\; \text{subject to } \boldsymbol{g}(\boldsymbol{t}) \leq \boldsymbol{0},$$

where $f : \mathbb{R}^d \to \mathbb{R}$ and $\boldsymbol{g} : \mathbb{R}^d \to \mathbb{R}^p$ are the differentiable convex functions. Its *Lagrange dual problem* is given as

$$\max_{\boldsymbol{\lambda}} \inf_{\boldsymbol{t}} L(\boldsymbol{t}, \boldsymbol{\lambda}) \;\; \text{subject to } \boldsymbol{\lambda} \geq \boldsymbol{0},$$

where

$$\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_p)^\top$$

is called the *Lagrange multipliers* and

$$L(\boldsymbol{t}, \boldsymbol{\lambda}) = f(\boldsymbol{t}) + \boldsymbol{\lambda}^\top \boldsymbol{g}(\boldsymbol{t})$$

is called the *Lagrange function* (or simply the Lagrangian). The solution of the Lagrange dual problem for $\boldsymbol{t}$ agrees with the original constrained optimization problem.

FIGURE 23.5

Lagrange dual problem.

Lagrange duality (Fig. 23.5) yields that the solution of optimization problem (23.1) can be obtained by solving the following Lagrange dual problem:

$$\max_{\lambda} \min_{\boldsymbol{\theta}} \left[J_{\mathrm{LS}}(\boldsymbol{\theta}) + \frac{\lambda}{2}\left(\|\boldsymbol{\theta}\|^2 - R^2 \right) \right] \quad \text{subject to } \lambda \geq 0.$$

Although the Lagrange multiplier λ is determined based on the radius R in principle, λ may be directly specified in practice. Then the solution of ℓ_2-constrained LS $\widehat{\boldsymbol{\theta}}$ is given by

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left[J_{\mathrm{LS}}(\boldsymbol{\theta}) + \frac{\lambda}{2} \|\boldsymbol{\theta}\|^2 \right]. \tag{23.2}$$

The first term in Eq. (23.2) measures the goodness of fit to training samples, while the second term $\frac{\lambda}{2}\|\boldsymbol{\theta}\|^2$ measures the amount of overfitting.

Differentiating the objective function in Eq. (23.2) with respect to parameter $\boldsymbol{\theta}$ and setting it to zero give the solution of ℓ_2-constrained LS analytically as

$$\widehat{\boldsymbol{\theta}} = (\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1} \boldsymbol{\Phi}^\top \boldsymbol{y}, \tag{23.3}$$

where $\boldsymbol{I}$ denotes the identity matrix. From Eq. (23.3), ℓ_2-constrained LS enhances the regularity of matrix $\boldsymbol{\Phi}^\top\boldsymbol{\Phi}$ by adding $\lambda\boldsymbol{I}$, which contributes to stabilizing the computation of its inverse. For this reason, ℓ_2-constrained LS is also called *ℓ_2-regularization learning*, the second term $\|\boldsymbol{\theta}\|^2$ in Eq. (23.2) is called a *regularizer*, and λ is called the *regularization parameter*. In statistics, ℓ_2-constrained LS is referred to as *ridge regression* [56].

Let us consider *singular value decomposition* (see Fig. 22.2) of design matrix $\boldsymbol{\Phi}$:

$$\boldsymbol{\Phi} = \sum_{k=1}^{\min(n,b)} \kappa_k \boldsymbol{\psi}_k \boldsymbol{\varphi}_k^\top,$$

where $\boldsymbol{\psi}_k$, $\boldsymbol{\phi}_k$, and κ_k are a *left singular vector*, a *right singular vector*, and a *singular value* of $\boldsymbol{\Phi}$, respectively. Then the solution of ℓ_2-constrained LS can be expressed as

$$\widehat{\boldsymbol{\theta}} = \sum_{k=1}^{\min(n,b)} \frac{\kappa_k}{\kappa_k^2+\lambda} \boldsymbol{\psi}_k^\top \boldsymbol{y} \boldsymbol{\varphi}_k.$$

When $\lambda = 0$, ℓ_2-constrained LS is reduced to ordinary LS. When the design matrix $\boldsymbol{\Phi}$ is *ill-conditioned* in the sense that a very small singular value exists, the inverse of that singular value, κ_k/κ_k^2 $(= 1/\kappa_k)$, can be very large. Then noise included in training output vector $\boldsymbol{y}$ is magnified by the factor $1/\kappa_k$ in ordinary LS. On the other hand, in ℓ_2-constrained LS, κ_k^2 in the denominator is increased by adding a positive scalar λ, which prevents $\kappa_k/(\kappa_k^2+\lambda)$ from being too large and thus overfitting can be mitigated.

A MATLAB code for ℓ_2-constrained LS is provided in Fig. 23.6, where the Gaussian kernel model is used:

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

where

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

The Gaussian bandwidth is set at $h = 0.3$, and the regularization parameter is set at $\lambda = 0.1$. The behavior of ℓ_2-constrained LS is illustrated in Fig. 23.7, showing that overfitting can be successfully avoided.

ℓ_2-constrained LS can be slightly generalized by using a $b\times b$ positive semidefinite matrix $\boldsymbol{G}$ as

$$\min_{\boldsymbol{\theta}} J_{\mathrm{LS}}(\boldsymbol{\theta}) \quad \text{subject to } \boldsymbol{\theta}^\top \boldsymbol{G} \boldsymbol{\theta} \leq R^2,$$

which is called *generalized ℓ_2-constrained LS*. When matrix $\boldsymbol{G}$, called a *regularization matrix*, is positive and symmetric, $\boldsymbol{\theta}^\top \boldsymbol{G} \boldsymbol{\theta} \leq R^2$ represents an ellipsoidal

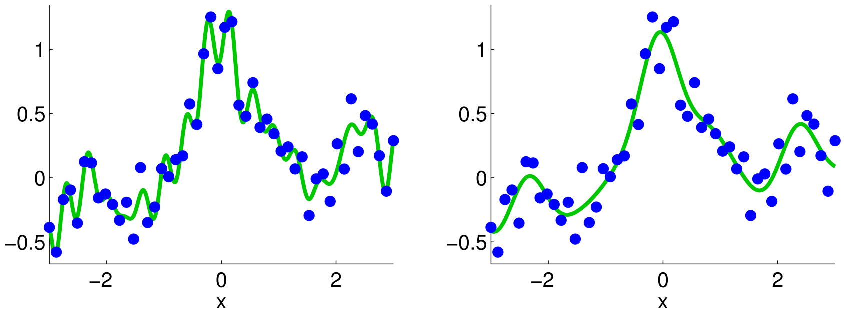
```
n=50; N=1000; x=linspace(-3,3,n)'; X=linspace(-3,3,N)';
pix=pi*x; y=sin(pix)./(pix)+0.1*x+0.2*randn(n,1);

x2=x.^2; X2=X.^2; hh=2*0.3^2; l=0.1;
k=exp(-(repmat(x2,1,n)+repmat(x2',n,1)-2*x*x')/hh);
K=exp(-(repmat(X2,1,n)+repmat(x2',N,1)-2*X*x')/hh);
t1=k\y; F1=K*t1; t2=(k^2+l*eye(n))\(k*y); F2=K*t2;

figure(1); clf; hold on; axis([-2.8 2.8 -1 1.5]);
plot(X,F1,'g-'); plot(X,F2,'r--'); plot(x,y,'bo');
legend('LS','L2-Constrained LS');
```

FIGURE 23.6

MATLAB code of ℓ_2-constrained LS regression for Gaussian kernel model.



(a) Ordinary LS (b) ℓ_2-constrained LS

FIGURE 23.7

Example of ℓ_2-constrained LS regression for Gaussian kernel model. The Gaussian bandwidth is set at $h = 0.3$, and the regularization parameter is set at $\lambda = 0.1$.

constraint (Fig. 23.8). The solution of generalized ℓ_2-constrained LS can be obtained in the same way as ordinary ℓ_2-constrained LS by

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

23.3 MODEL SELECTION

In the previous sections, constrained LS was demonstrated to contribute to mitigating overfitting. However, the behavior of constrained LS depends on the choice of con-

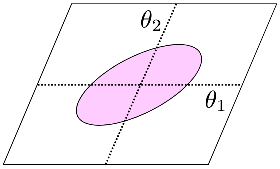
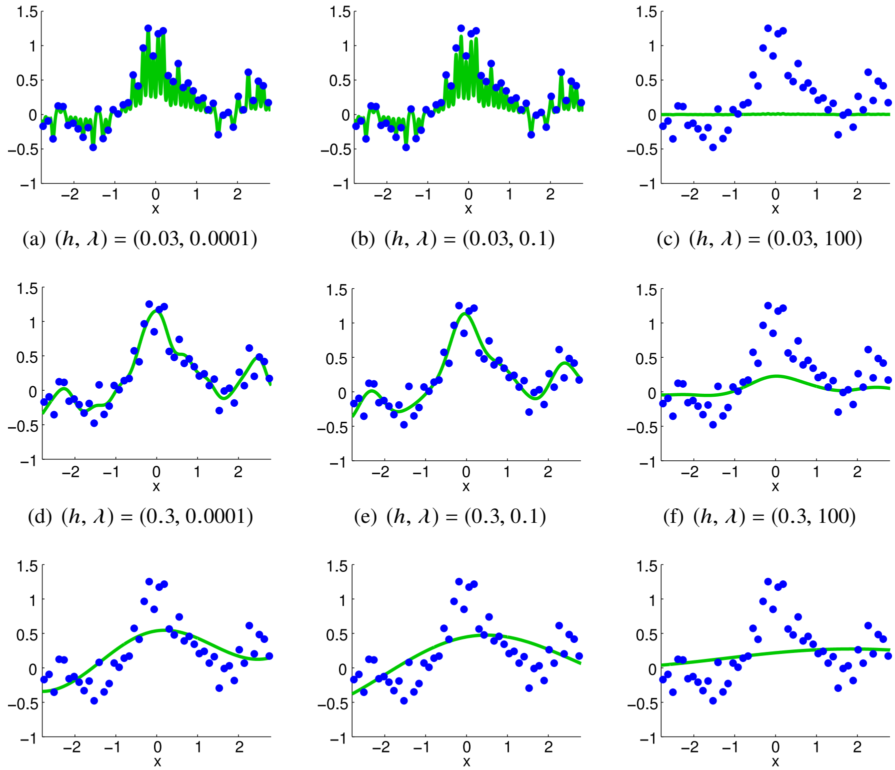


FIGURE 23.8

Parameter space in generalized ℓ_2-constrained LS.



(a) $(h, \lambda) = (0.03, 0.0001)$ (b) $(h, \lambda) = (0.03, 0.1)$ (c) $(h, \lambda) = (0.03, 100)$

(d) $(h, \lambda) = (0.3, 0.0001)$ (e) $(h, \lambda) = (0.3, 0.1)$ (f) $(h, \lambda) = (0.3, 100)$

(g) $(h, \lambda) = (3, 0.0001)$ (h) $(h, \lambda) = (3, 0.1)$ (i) $(h, \lambda) = (3, 100)$

FIGURE 23.9

Examples of ℓ_2-constrained LS with the Gaussian kernel model for different Gaussian bandwidth h and different regularization parameter λ.

```
n=50; 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);

N=1000; X=linspace(-3,3,N)';
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=K*t;

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

FIGURE 23.10

MATLAB code of cross validation for ℓ_2-constrained LS regression.

straining parameters such as the projection matrix $\boldsymbol{P}$ and the regularization parameter λ. Furthermore, choice of basis/kernel functions also affects the performance.

Fig. 23.9 illustrates the solutions of ℓ_2-constrained LS with the Gaussian kernel model for different Gaussian bandwidth h and different regularization parameter λ. The obtained functions are highly fluctuated if the Gaussian bandwidth h is too small, while they are overly smoothed if the Gaussian bandwidth h is too large. Similarly, overfitting is prominent if the regularization parameter λ is too small, and the obtained functions become too flat if the regularization parameter λ is too large. In this particular example, $h = 0.3$ and $\lambda = 0.1$ would be a reasonable choice. However, the best values of h and λ depend on various unknown factors such as the true learning target function and the noise level.

The problem of data-dependent choice of these tuning parameters, called *model selection*, can be addressed by *cross validation* (see Section 14.4 and Section 16.4.2). Note that naive model selection based on the training squared error simply yields

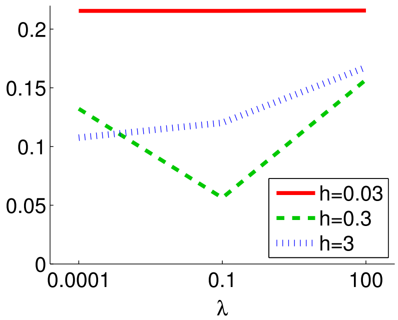


FIGURE 23.11

Example of cross validation for ℓ_2-constrained LS regression. The cross validation error for all Gaussian bandwidth h and regularization parameter λ is plotted, which is minimized at $(h, \lambda) = (0.3, 0.1)$. See Fig. 23.9 for learned functions.

For invertible and symmetric matrix $\boldsymbol{A}$, it holds that

$$(\boldsymbol{A} + \boldsymbol{b}\boldsymbol{b}^\top)^{-1} = \boldsymbol{A}^{-1} - \frac{\boldsymbol{A}^{-1}\boldsymbol{b}\boldsymbol{b}^\top\boldsymbol{A}^{-1}}{1 + \boldsymbol{b}^\top\boldsymbol{A}^{-1}\boldsymbol{b}}.$$

If $c - \boldsymbol{b}^\top\boldsymbol{A}^{-1}\boldsymbol{b} \neq 0$, it holds that

$$\begin{pmatrix} \boldsymbol{A} & \boldsymbol{b} \\ \boldsymbol{b}^\top & c \end{pmatrix}^{-1} = \begin{pmatrix} \boldsymbol{A}^{-1} + \alpha\boldsymbol{A}^{-1}\boldsymbol{b}\boldsymbol{b}^\top\boldsymbol{A}^{-1} & -\alpha\boldsymbol{A}^{-1}\boldsymbol{b} \\ -\alpha\boldsymbol{b}^\top\boldsymbol{A}^{-1} & \alpha \end{pmatrix},$$

where

$$\alpha = \frac{1}{c - \boldsymbol{b}^\top\boldsymbol{A}^{-1}\boldsymbol{b}}.$$

FIGURE 23.12

Matrix inversion lemma.

overfitting. For example, in Fig. 23.9, choosing the smallest h and λ minimizes the training squared error, which results in the heaviest overfitting. The algorithm of cross validation is exactly the same as the one described in Fig. 16.17, but the *squared loss function* is used to compute the validation error:

$$J_j^{(\ell)} = \frac{1}{|\mathcal{Z}_\ell|} \sum_{(\boldsymbol{x}', y') \in \mathcal{Z}_\ell} \left(y' - \widehat{f}_j^{(\ell)}(\boldsymbol{x}')\right)^2,$$

where $|\mathcal{Z}_\ell|$ denotes the number of elements in the set $\mathcal{Z}_\ell$ and $\widehat{f}_j^{(\ell)}$ denotes the function learned using model $\mathcal{M}_j$ from all training samples without $\mathcal{Z}_\ell$.

A MATLAB code of cross validation for the data in Fig. 23.9 is provided in Fig. 23.10, and its behavior is illustrated in Fig. 23.11. In this example, the Gaussian bandwidth h is chosen from $\{0.03, 0.3, 3\}$, the regularization parameter λ is chosen from $\{0.0001, 0.1, 100\}$, and the number of folds in cross validation is set at 5. The cross validation error is minimized at $(h, \lambda) = (0.3, 0.1)$, which would be a reasonable choice as illustrated in Fig. 23.9.

Cross validation when the number of folds is set at the number of training samples n, i.e., $n - 1$ samples are used for training and only the remaining single sample is used for validation, is called *leave-one-out cross validation*. Naive implementation of leave-one-out cross validation requires n repetitions of training and validation, which is computationally demanding when n is large. However, for ℓ_2-constrained LS, the score of leave-one-out cross validation can be computed analytically without repetition as follows [77]:

$$\frac{1}{n} \|\widetilde{\boldsymbol{H}}^{-1} \boldsymbol{H} \boldsymbol{y}\|^2, \tag{23.4}$$

where $\boldsymbol{H}$ is the $n \times n$ matrix defined as

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

and $\widetilde{\boldsymbol{H}}$ is the diagonal matrix with diagonal elements the same as $\boldsymbol{H}$. In the derivation of Eq. (23.4), the *matrix inversion lemma* was utilized (see Fig. 23.12).