Chapter 4 Overfitting, Regularization, and Validation

Overfitting is an important issue to understand and address in machine learning: it is a phenomenon where good fitting of training data does not necessarily imply satisfactory generalization. In this chapter, we address the question of when overfitting occur and present two approaches to dealing with the overfitting problem in supervised learning. The materials presented here are largely based on [8].

4.1 What Is Overfitting?

Roughly speaking, overfitting refers to *fitting the data more than is warranted* [8]. The main case of overfitting is when the learning model yields a hypothesis with a low E_{in} but a high E_{out}. As will become transparent shortly, there are three things involved in overfitting data: target complexity, noise, and data size. Below we illustrate the interplay between them in relation to the overfitting phenomenon through two examples.

Example 4.1 *Overfitting with polynomials when data are noisy*

Consider the problem of fitting a polynomial $p(x)$ into15 points $\mathcal{D} = \{(x_n, y_n), n = 1, \ldots, 15\}$ in least-squares (LS) sense, where the data points are produced by a target function which is a polynomial of order 10 plus Gaussian noise with variance $\sigma^2 = 0.09$ (i.e. $\sigma = 0.3$). The MATLAB code below generates a target function and data set $\mathcal{D}$ as shown in Fig. 4.1.

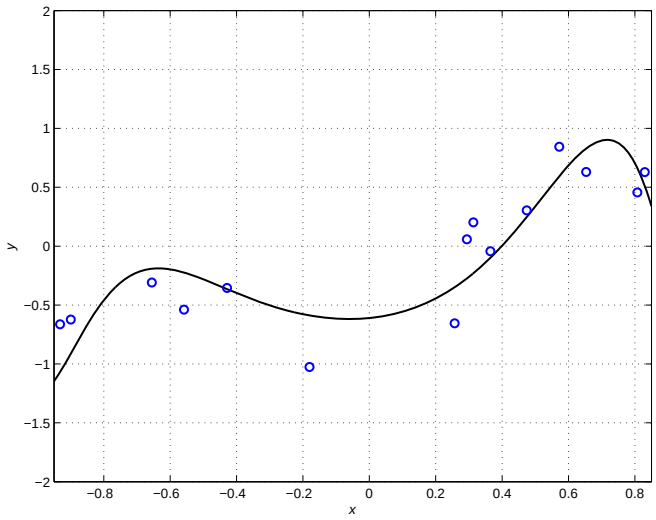


Fig. 4.1 10th-order target function and 15 noisy data points with noise variance $\sigma^2 = 0.09$.

```
rand('state',148),
x = sort(2*(rand(1,15)-0.5));
randn('state',119)
p = 5*randn(1,11);
y = polyval(p,x);
randn('state',18)
w = 0.3*randn(1,15);
yn = y + w;
```

```
xt = -0.95:1.8/1000:0.85;
yt = polyval(p,xt);
figure(1)
plot(xt,yt,'k-','linewidth',1.5)
hold on
plot(x,yn,'bo','linewidth',1.5)
grid
axis([-0.95 0.85 -2 2])
xlabel('\itx')
ylabel('\ity')
hold off
```

Next, we apply 10^{th} order and 2^{nd} order LS fits to 15 noisy data points :

```
pn = polyfit(x,yn,10);
zn = polyval(pn,x);
zt = polyval(pn,xt);
```

where order_m was set to 10 and 2 to carrying the 10^{th} order and 2^{nd} order LS fits, respectively. Figures 4.2 and 4.3 depict these two cases.

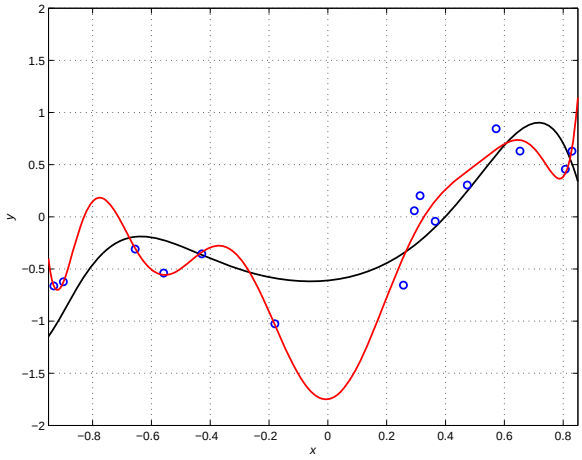


Fig. 4.2 10^{th} order LS fit (red curve).

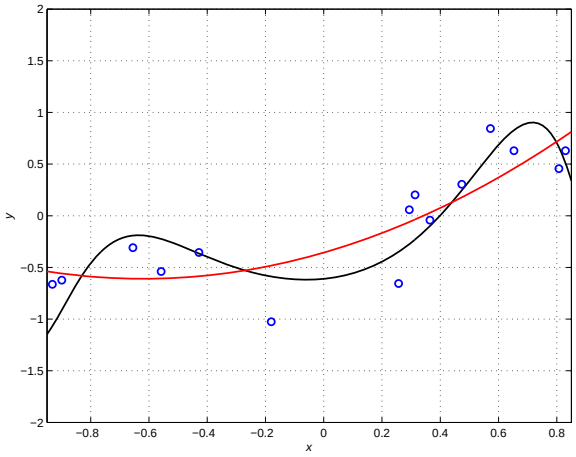


Fig. 4.3 2^{nd} order LS fit (red curve).

The performance of the two LS fits can also be evaluated in terms of in-sample and, more importantly, out-of-sample errors. The in-sample error is defined by the average squared 2-norm of the difference between the noisy sample point and the corresponding point generated by the fitting polynomial $p(x)$, namely,

$$E_{\text{in}} = \frac{1}{N}\sum_{n=1}^{N} | \, p(x_n) - y_n \, |^2 \tag{4.1}$$

The out-of-sample error is also of average squared 2-norm type, but over the region outside the data set $\mathcal{D}$. In this case study the $\{x_n\}$ in data D are chosen at random over interval [–1, 1], while the $\{x_t, t = 1, \ldots, T\}$ are dense grids that are placed uniformly over [–1, 1]. Thus

$$E_{\text{out}} = \frac{1}{T}\sum_{t=1}^{T} | \, p(x_t) - y_t \, |^2 \qquad (4.2)$$

where y_t was generated by the target function at x_t plus the noise. In our simulation $N = 15$ and T was set to 1001. The MATLAB code associated this part of simulation is given below:

```
ein = (norm(yn-zn)^2)/15;
randn('state',18)
wn = sig*randn(1,1001);
ytn = yt + wn;
eout = (norm(ytn-zt)^2)/1001;
```

Based on the definitions from (4.1) and (4.2), the outcome of numerical evaluations are as follows. For the 10th order fit, $E_{\text{in}} = 0.0201$ and $E_{\text{out}} = 0.3250$, while for the 2nd order fit, $E_{\text{in}} = 0.0834$ and $E_{\text{out}} = 0.1544$. The numerical results indicate a clear case of ovefitting. Here we make two observations: (1) The target function is also a 10th order polynomial. Therefore, choosing a 10th order polynomial to fit the data seems reasonable and justified, so why overfitting has occurred? One explanation is that it has to do with the presence of noise in the data– overfitting occurs because the 10th order polynomial tries to fit the noise! Note that the 2nd order fit is considerably less capable of fitting the noisy data which, on one hand increases E_{in} relative to that achieved by the 10th order fit, but on the other hand tends to be less sensitive to the noise, hence a reduced E_{out}. (2) Put it in another perspective, overfitting may occur when the *quality* of the data does not match the target complexity. ■

A question that naturally arises is whether the observations made above are merely a coincidence. Addressing the question requires simulation studies where a large number of experiments at a range of noise levels are conducted to see how overfitting in average is related to the noise level in the data given the target complexity and a certain order of polynomial fit. To prepare the simulations, we introduce a class of polynomials known as *Legendre polynomials* whose orders provide a better indication of their complexity than the monomials $\{x^k, k = 0, 1, 2 \ldots\}$ do.

Legendre Polynomials

The Legendre polynomials $\{L_k(x), k = 0, 1, \ldots\}$ are defined over interval $[-1, 1]$ by

$$L_k(x) = \frac{1}{2^k k!}\frac{d^k}{dx^k}\left[\left(x^2 - 1\right)^k\right] \quad \text{for } k = 0, 1, \ldots \qquad (4.3)$$

It follows that $L_0(x) = 1$, $L_1(x) = x$. Several properties of the Legendre polynomials are listed below.

(i) $L_k(-x) = (-1)^k L_k(x)$

(ii) $L_k(-1) = (-1)^k$, $L_k(1) = 1$

(iii) Orthogonality of Legendre polynomials:

$$\int_{-1}^{1} L_k(x) L_l(x) dx = \frac{2}{2k+1}\,\delta(k-l)$$

where $\delta(i)$ equals to one when $i = 0$ and zero elsewhere.

(iv) With $L_0(x) = 1$ and $L_1(x) = x$, $L_k(x)$ for $k > 1$ can be found by recursion as

$$L_{k+1}(x) = \frac{(2k+1)}{k+1} x \cdot L_k(x) - \frac{k}{k+1} L_{k-1}(x) \quad \text{for } k = 1, 2, \ldots \tag{4.6}$$

The MATLAB function below computes the coefficients of $L_k(x)$ for $k = 0, 1, \ldots n$.

```
% Generate the coefficients of Legendre polynomials pk(x) for k = 0, 1, ..., n. It is well known
% that p0(x) = 1 and p1(x) = x. The input of the MATLAB code is for n >= 2.
% The output is a matrix of size (n+1) by (n+1) whose (k+1)th row lists the coefficients of pk(x)
% in descending order.
% Written by W.-S. Lu, University of Victoria. Last modified: Feb. 11, 2015.
function P = coeff_legendre(n)
P = zeros(n+1,n+1);
P(1,n+1) = 1;
P(2,n) = 1;
for i = 1:(n-1),
   p1 = P(i+1,:);
   p1new = [p1(2:n+1) 0];
   p2 = P(i,:);
   r1 = (2*i+1)/(i+1);
   r2 = i/(i+1);
   P(i+2,:) = r1*p1new - r2*p2;
end
```

Fig. 4.4 shows the first six Legendre polynomials from which we can sense how the "complexity" of the polynomials grows with the order (degree).

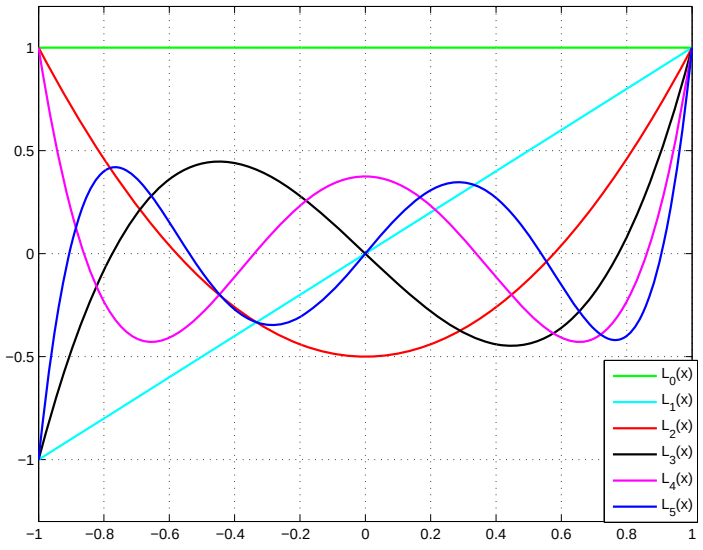


Fig. 4.4 Legendre polynomials $L_0(x)$, …, $L_5(x)$.

We now consider the case with a 10^{th} order target polynomial of the form

$$f(x) = \sum_{k=0}^{10} \alpha_k L_k(x)$$

where $\{\alpha_k\}$ are chosen at random and normalized so that $E_x[f(x)^2] = 1$. We let standard deviation σ of the noise in the data vary from $\sigma = 0$ to $\sigma = 0.5$, each time with an increment of 0.025. At each noise level, we carry out 1000 runs for 10^{th} order and 2^{nd} order fits, compute the average E_{out} for each fit, and use the difference $E_{\text{out}}[g_{10}(x)] - E_{\text{out}}[g_2(x)]$ as an overfit measure. The results are depicted in Fig. 4.5. The figure shows how overfitting gets worse as data gets noiser.

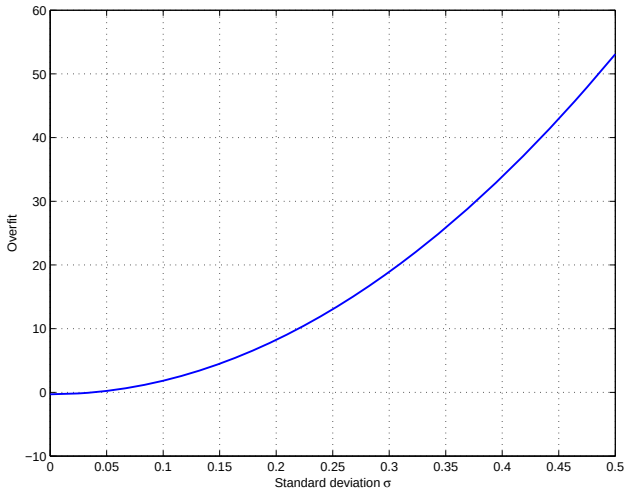


Fig. 4.5 Overfit as a function of standard deviation σ of the noise in data set. The order of the target function was set to 10, with its coefficients chosen at random and normalized. The overfit shown here was the averaged $E_{\text{out}}[g_{10}(x)] - E_{\text{out}}[g_2(x)]$ over 1000 runs. See the MATLAB code below for implementation details.

The MATLAB code that implements the simulation and generates Fig. 4.5 are given below.

```
% To conduct the first set of simulations described in the case studies in Sec. 4.1.
% Input:
% N: number of sample points.
% st1: initial state for sample points.
% st2: initial state for weights that combine Q + 1 Legendre polynomials in
%       order to generate the target poly of order Q (which is set to 10 here).
% st3: initial state for noise.
% K: number of experiments for each case with fixed noise level.
% Output:
% E2: Average out-of-sample error for 2nd-order fit as function of noise level.
% E10: Average out-of-sample error for 10th-order fit as function of noise level.
% U: Overfit measure by E10 - E2.
% Written by W.-S. Lu, University of Victoria.  Last modified: Feb. 14, 2015.
% Example: [U,E2,E10] = ex4_1b(15,148,9,18,2000);
function [U,E2,E10] = ex4_1b(N,st1,st2,st3,K)
Q = 10;                     % Order of target polynomial.
P = coeff_legendre(Q);
```

```
xt = -1:2/1000:1;
sig = 0:0.025:0.5;
Ls = length(sig);
E2 = zeros(Ls,1);
E10 = E2;
rand('state',st1),
x = sort(2*(rand(1,N)-0.5));
for i = 1:Ls,
   sigi = sig(i);
   eg2 = 0;
   eg10 = 0;
   for k = 1:K,
      randn('state',st2+k-1)
      ck = randn(1,Q+1);
      pk = ck*P;
      ytk = polyval(pk,xt);
      mk = mean(ytk.^2);
      pk = pk/sqrt(mk);
      yk = polyval(pk,x);
      ytk = polyval(pk,xt);
      randn('state',st3+k-1)
      ykn = yk + sigi*randn(1,N);
      randn('state',st3+k-1)
      ytkn = ytk + sigi*randn(1,1001);
      p2 = polyfit(x,ykn,2);
      ztk2 = polyval(p2,xt);
      p10 = polyfit(x,ykn,10);
      ztk10 = polyval(p10,xt);
      eg2 = eg2 + (norm(ytkn-ztk2)^2)/1001;
      eg10 = eg10 + (norm(ytkn-ztk10)^2)/1001;
    end
    E2(i) = eg2/K;
    E10(i) = eg10/K;
end
U = E10 - E2;
figure(1)
plot(sig,U,'b-','linewidth',1.5)
xlabel('Standard deviation \sigma')
ylabel('Overfit')
grid
```

Next, we fixed a noise level to $\sigma = 0.4$ and order of target polynomial to $Q = 10$, and evaluated averaged overfit as the data size N varies in the range from 15 to 120, each time by an increment of 5. The results are depicted in Fig. 4.6. The figure shows how overfitting vanishes as data size gets larger.

Example 4.2 *Overfitting with polynomials for a highly complex target*

In this example, we look at how overfitting is related to target complexity. To focus on this aspect of overfitting, the data sets in our simulations were free of noise contamination. In the first

set of simulations, 10th and 2nd order LS fits were applied to 15 data points that were selected at random over [–1, 1] for a target polynomial of order 50 that was generated by random combination of 51 Legendre polynomials of orders varying from 0 to 50. The results of the two fits are depicted in Figs. 4.7 and 4.8.

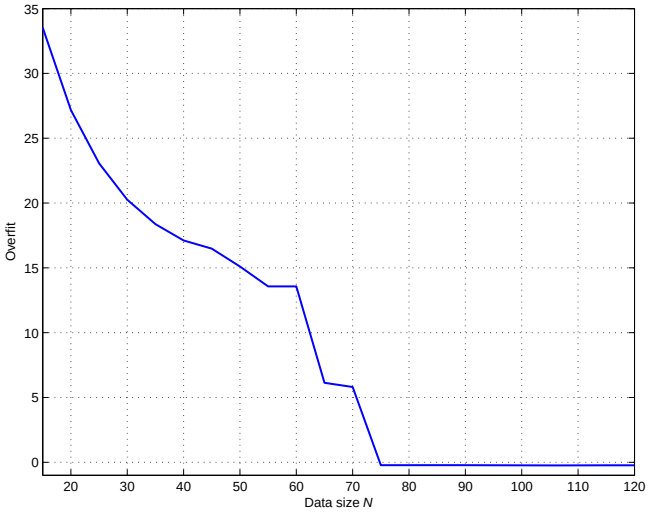


Fig. 4.6 Overfit as a function of data size N. The standard deviation σ of the noise in data set was set to 0.4; the order of the target function was set to $Q = 10$ with its coefficients chosen at random and normalized. The overfit shown here was the averaged $E_{\text{out}}[g_{10}(x)] - E_{\text{out}}[g_2(x)]$ over 2000 runs.

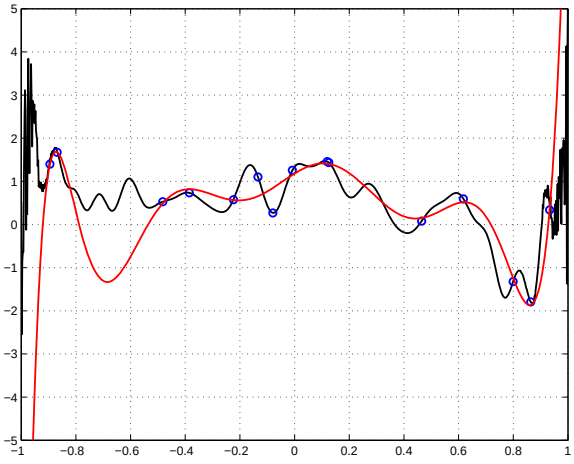


Fig. 4.7 10th order LS fit of a 50th order target.

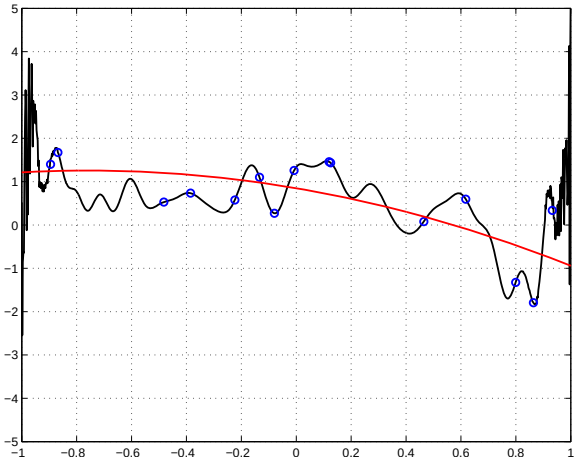


Fig. 4.8 2nd order LS fit of a 50th order target.

For the 10th order fit, $E_{\text{in}} = 0.0385$ and $E_{\text{out}} = 5.2772$, while for the 2nd order fit, $E_{\text{in}} = 0.4419$ and $E_{\text{out}} = 0.5655$. Thus with hypothesis $\mathcal{H}_{10}$ overfitting has occurred over hypothesis $\mathcal{H}_2$, where $\mathcal{H}_r$ denotes the set of polynomials of order r. We see that, between $\mathcal{H}_{10}$ and $\mathcal{H}_2$, a learning model with hypothesis set $\mathcal{H}_2$ provides generalization performance for a given 50th order target function. On the other hand, the optimal hypothesis from $\mathcal{H}_2$ (the red curve shown in Fig. 4.8) differs considerably from the target function (the black curve in Fig. 4.8). The difference between these

curves is called *deterministic noise*. The term comes from the fact that it is not stochastic because the data was assumed to be perfectly accurate, however it cannot be modeled by an $\mathcal{H}_2$ hypothesis just like the stochastic noise in the data that cannot be modeled by $\mathcal{H}_2$.

Next, we looked at three cases, each with a fixed data size N and 10^{th} and 2^{nd} order fits are applied to approximate a target polynomial whose order varied from $Q = 10$ to $Q = 46$ with an increment of 2. The averaged overfit over 2000 runs of the three cases with $N = 15$, 20, and 25 are shown in Fig. 4.9. It is observed that (i) overfitting grows with target complexity Q; (ii) overfitting reduces as data size N increases. ■

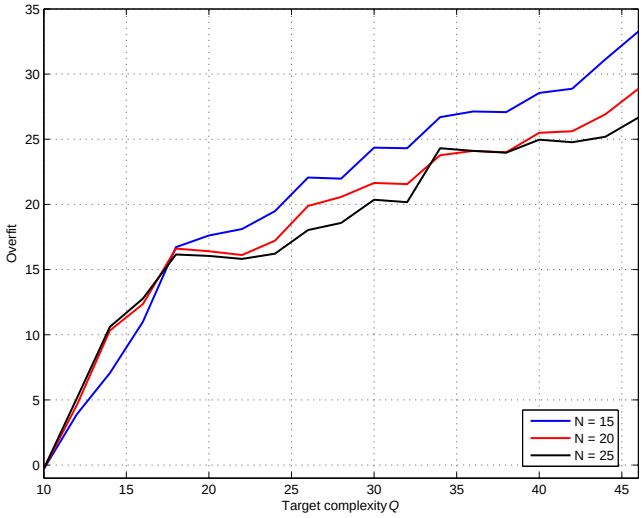


Fig. 4.9 Overfit as a function of target complexity Q. The noise level σ was set to 0; the order of the target function varied from $Q = 10$ and $Q = 46$. The overfit shown here was the averaged $E_{\text{out}}[g_{10}(x)] - E_{\text{out}}[g_2(x)]$ over 2000 runs.

Summarizing the observations made through the two examples, the primary causes of overfitting are stochastic noise that degrades the *quality* of training data and deterministic noise due to higher target complexity. The size of the data (i.e., *quantity*) has proven to play an important role in the overfitting phenomenon – increasing the data size in general reduces or even eliminates overfitting. In other words, "what matters is how the model complexity matches the *quantity and quality* of the data we have, not how it matches the target function [8]."

4.2 Regularization Techniques

We have seen that overfitting is closely related to size of training data as well as target complexity. Regularization is a general technique to address overfitting problems that has found effective regardless of whether the overftting is due to insufficient data size or high target complexity. One way to understand the machinery of regularization is through the VC-dimension analysis. Recall Eq. (2.30), namely,

$$E_{\text{out}}(h) \leq E_{\text{in}}(h) + \Omega(N, \mathcal{H}, \delta) \tag{4.7}$$

where

$$\Omega(N,\mathcal{H},\delta)=\sqrt{\frac{8}{N}\ln\frac{4m_{\mathcal{H}}(2N)}{\delta}}\le\sqrt{\frac{8}{N}\ln\frac{4(2N)^{d_{vc}}+4}{\delta}}$$

is controlled by data size N, confidence parameter δ, and VC-dimension d_{vc}. In a typical learning problem N and δ are fixed, hence $\Omega(N,\mathcal{H},\delta)$ depends only on d_{vc} which indicates the complexity of hypothesis $\mathcal{H}$. Thus (4.7) becomes

$$E_{\text{out}}(g)\le E_{\text{in}}(g)+\Omega(\mathcal{H}) \tag{4.8}$$

Roughly speaking, *regularized learning* minimizes a combination of $E_{\text{in}}(h)$ with $\Omega(h)$ over $h\in\mathcal{H}$ instead of minimizing the in-sample error $E_{\text{in}}(h)$ alone, that is

$$\underset{h}{\text{minimize}}\;\; E_{\text{in}}(h)+\lambda\,\Omega(h) \tag{4.9}$$

where $\Omega(h)$ is a penalty term for the complexity of hypothesis h, and $\lambda\ge 0$ is a parameter that controls the amount of regularization. From (4.8), it is quite clear that regularization is justified as it effectively prevents $E_{\text{out}}(h)$ from being too large.

A. *Regularization with L_2-Penalty*

Example 4.3 *Use regularization to handle insufficient data size*

Again we consider Example 4.1 illustrated in Figs. 4.1 to 4.3, where the target function was a 10^{th} order polynomial and a 10^{th} order LS was found to overfit when it applied to a set of 15 noisy data points. Here we re-formulate the problem as a regularized LS problem. Let $f(x)$ be the target function and $h(x,\boldsymbol{w})$ be the 10^{th} order hypothesis polynomial with coefficient vector $\boldsymbol{w}=[w_0\;w_1\cdots w_{10}]^T$, i.e.,

$$h(x,\boldsymbol{w})=w_0+w_1x+\cdots+w_{10}x^{10}$$

We employ an L_2-loss function to measure the in-sample error

$$E_{\text{in}}(\boldsymbol{w})=\sum_{n=1}^{N}|\,h(x_n,\boldsymbol{w})-f(x_n)\,|^2=\sum_{n=1}^{N}|\,h(x_n,\boldsymbol{w})-y_n\,|^2 \tag{4.10}$$

and an L_2 penalty term $\Omega(h)=\|\boldsymbol{w}\|_2^2$. In this way, scheme (4.10) is realized by a regularized least-squares formulation as

$$\underset{\boldsymbol{w}}{\text{minimize}}\;\; E_{\text{in}}(\boldsymbol{w})+\lambda\,\|\boldsymbol{w}\|_2^2 \tag{4.11}$$

Taking a perspective from VC-analysis, the solution weight $\boldsymbol{w}^*$ of problem (4.11) shall keep $E_{\text{in}}(\boldsymbol{w})$ sufficiently small subject to $\boldsymbol{w}$ being in a limited vicinity of the origin in the $\boldsymbol{w}$ space thereby reducing the *effective* VC dimension [8] of $\mathcal{H}$. It follows from (4.8) that such a $\boldsymbol{w}^*$ is expected to offer a reduced E_{out} relative to that achieved by minimizing $E_{\text{in}}(\boldsymbol{w})$ alone.

With an L_2-loss $E_{\text{in}}(\boldsymbol{w})$ in (4.10), there is a closed-form solution for problem (4.11) which we derive as follows. Let $\boldsymbol{x}_n=[1\;\;x_n\;\;x_n^2\cdots x_n^{10}]^T$, $\boldsymbol{y}=[y_1\;\;y_2\;\;\cdots\;\;y_N]^T$, and write $h(x_n,\boldsymbol{w})=\boldsymbol{x}_n^T\boldsymbol{w}$, the in-sample error in (4.10) becomes $\|\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y}\|_2^2$ with $\boldsymbol{X}\in R^{N\times 11}$ whose nth row is $\boldsymbol{x}_n^T$, the objective function in (4.11) is given by

$$\boldsymbol{w}^T\left(\boldsymbol{X}^T\boldsymbol{X}+\lambda\boldsymbol{I}_{11}\right)\boldsymbol{w}-2\boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{y}+\text{const}$$

whose global minimizer is given by

$$\boldsymbol{w}^* = \left(\boldsymbol{X}^T\boldsymbol{X} + \lambda \boldsymbol{I}_{11}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{4.12}$$

For the example at hand, we applied (4.11) with $\lambda = 0.015$ for a regularized 10^{th} order LS fit. The performance is shown in Fig. 4.10 which on comparing with the 10^{th} order LS fit without regularization (see Fig. 4.2) demonstrates considerable improvement. Actually, the overfit has been eliminated as can be seen from the table below.

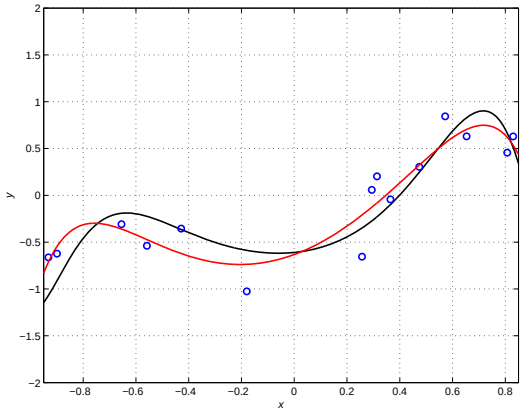
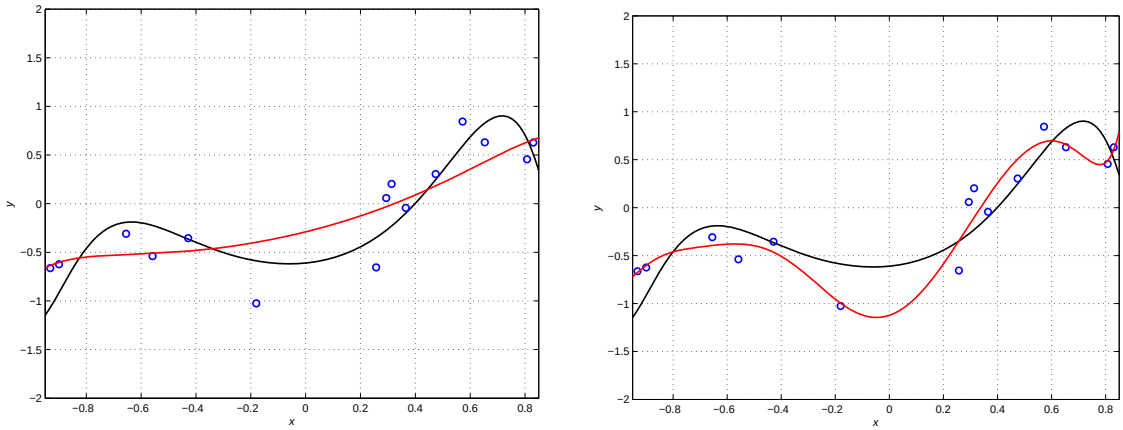


Fig. 4.10 L_2-regularized 10^{th} order LS fit (red curve).

	E_{in}	E_{out}
2^{nd} order LS fit without regularization	0.0834	0.1544
10^{th} order LS fit without regularization	0.0201	0.3250
10^{th} order LS fit with L_2-regularization ($\lambda = 0.015$)	0.0408	0.1218

The value of regularization parameter λ greatly affects solution's performance. As shown in the figures below, a λ too large (*over-regularization*, see Fig. 4.11a) or a λ too small (*under-regularization*, see Fig. 4.11b) leads to an increased out-of-sample error E_{out} relative to that with an optimal λ. The table enclosed below lists the numerical values of E_{in} and E_{out} for the three regularized solutions. It follows that using an appropriate value of λ is of practical importance. We shall come back to this issue in the next section where an approach based on cross validation is studied. ■



(a) (b)

Figure 4.11 (a) A over-regularized 10^{th} order LS fit with $\lambda = 0.3$ and (b) a under-regularized 10^{th} order LS fit with $\lambda = 10^{-6}$.

	E_{in}	E_{out}
10^{th} order LS fit with L_2-regularization ($\lambda = 0.3$)	0.0820	0.1515
10^{th} order LS fit with L_2-regularization ($\lambda = 0.015$)	0.0408	0.1218
10^{th} order LS fit with L_2-regularization ($\lambda = 10^{-6}$)	0.0257	0.1728

The MATLAB code used in this example is given below.

```
% Regularized least-squares polynomial fitting.
% Input:
% N: Data size.
% Q: Order of the target polynomial.
% M: Order of the fitting polynomial.
% lam: value of regularization parameter lambada.
% sig: standard deviation of the noise.
% st1: initial state for data samples.
% st2: initial state for target polynomial.
% st3: initial state for noise.
% Output:
% ws: Coefficients of f fitting polynomial.
% p: Coeff. of target polynomial.
% (x,y): Training data without noise.
% (x,yn): Training data with noise.
% Ein: In-sample error.
% Eout: Out-of-sample error.
% Written by W.-S. Lu, University of Victoria.
% Last modified: Feb. 20, 2015.
% Example:
% [ws,p,x,y,yn,Ein,Eout] = ex4_1a_re(15,10,10,0.015,0.3,148,119,18);
function [p,x,y,yn,Ein,Eout] = ex4_1a_re(N,Q,M,lam,sig,st1,st2,st3)
rand('state',st1)
x = sort(2*(rand(N,1)-0.5));
randn('state',st2)
p = 5*randn(1,Q+1);
y = polyval(p,x);
randn('state',st3)
w = sig*randn(N,1);
yn = y + w;
V = fliplr(vander(x));
X = V(:,1:(M+1));
I = eye(M+1,M+1);
w = (inv(X'*X + lam*I))*(X'*yn);
ws = flipud(w);
zn = polyval(ws,x);
xt = -0.95:1.8/1000:0.85;
yt = polyval(p,xt);
zt = polyval(ws,xt);
figure(1)
plot(xt,yt,'k-','linewidth',1.5)
hold on
plot(x,yn,'bo','linewidth',1.5)
```

```
plot(xt,zt,'r-','linewidth',1.5')
grid
axis([-0.95 0.85 -2 2])
xlabel('\itx')
ylabel('\ity')
hold off
Ein = (norm(yn-zn)^2)/N;
randn('state',st3)
wn = sig*randn(1,1001);
ytn = yt + wn;
Eout = (norm(ytn-zt)^2)/1001;
disp('in-sample error and out-of-sample error:')
[Ein Eout]
```

Example 4.4 *Use regularization to handle high target complexity*

We now re-examine Example 4.2 illustrated in Fig. 4.7 where the target function was a 50th order polynomial and overfitting occurred when a 10th order LS fir was applied to a set of 15 data points that are selected from [–1, 1] at random and free of noise contamination. Again we re-formulate the problem as a regularized LS problem as we did in the first example. Because the LS fit also involves a 10th order polynomial, the regularized solution assumes the same form as in (4.12), namely,

$$\boldsymbol{w}^* = \left(\boldsymbol{X}^T \boldsymbol{X} + \lambda \boldsymbol{I}_{11} \right)^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

With $\lambda = 0.01$, a regularized 10th order LS fit was obtained, whose performance is shown in Fig. 4.12. On comparing with the 10th order LS fit without regularization (see Fig. 4.2), the regularized solution demonstrates considerable improvement. The table below gives numerical results that compares the regularized 10th order LS fit with those obtained from 2nd order and 10th order LS fits without regularization. ■

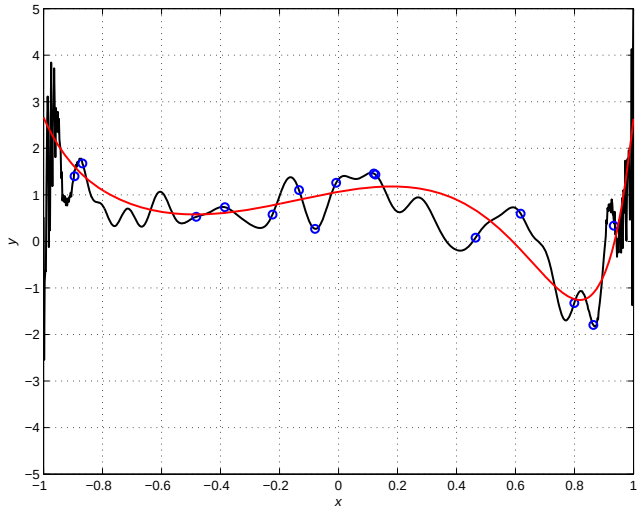


Fig. 4.12 L_2-regularized 10th order LS fit for a 50th order target function using 15 noise-free random data points.

	E_{in}	E_{out}
2nd order LS fit without regularization	0.4419	0.5655
10th order LS fit without regularization	0.0385	5.2772
10th order LS fit with L_2-regularization ($\lambda = 0.01$)	0.1644	0.3287

B. *Regularization with L_1-Penalty for Sparse Solutions*

B.1 *L_2-L_1 Minimization*

This part requires some background knowledge on convex functions with Lipschitz continuous gradients. A continuously differentiable function $u(\boldsymbol{x})$ with $\boldsymbol{x} \in R^n$ is said to have Lipschitz continuous gradient if

$$\left\|\nabla u(\boldsymbol{x}) - \nabla u(\boldsymbol{y})\right\|_2 \le L\left\|\boldsymbol{x} - \boldsymbol{y}\right\|_2 \tag{4.13}$$

holds for any $\boldsymbol{x}$ and $\boldsymbol{y}$, where $L > 0$ is a (Lipschitz) constant. An example of such function is $u(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$ whose gradient is given by $\nabla u(\boldsymbol{x}) = \boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^T\boldsymbol{b}$, hence

$$\left\|\nabla u(\boldsymbol{x}) - \nabla u(\boldsymbol{y})\right\|_2 = \left\|\boldsymbol{A}^T\boldsymbol{A}(\boldsymbol{x} - \boldsymbol{y})\right\|_2 \le \lambda_{\max}(\boldsymbol{A}^T\boldsymbol{A})\left\|\boldsymbol{x} - \boldsymbol{y}\right\|_2$$

Therefore $u(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$ has Lipschitz continuous gradient with $L = \lambda_{\max}(\boldsymbol{A}^T\boldsymbol{A})$.

Recall that a smooth function $u(\boldsymbol{x})$ is convex if and only if all tangent lines of the function lie underneath the function's graph. An analytical form of the above statement is

$$u(\boldsymbol{x}) \ge u(\boldsymbol{y}) + \nabla^T u(\boldsymbol{y})(\boldsymbol{x} - \boldsymbol{y})$$

In other words, *a convex if globally bounded from below by a linear function*. Now if in addition $u(\boldsymbol{x})$ is convex with a Lipschitz gradient, then $u(\boldsymbol{x})$ is also *globally bounded from above* by a convex quadratic function with Hessian $L \cdot \boldsymbol{I}$, namely,

$$u(\boldsymbol{y}) + \nabla^T u(\boldsymbol{y})(\boldsymbol{x} - \boldsymbol{y}) \le u(\boldsymbol{x}) \le u(\boldsymbol{y}) + \nabla^T u(\boldsymbol{y})(\boldsymbol{x} - \boldsymbol{y}) + \frac{L}{2}\left\|\boldsymbol{x} - \boldsymbol{y}\right\|_2^2 \tag{4.14}$$

Proof: We write

$$u(\boldsymbol{x}) = u(\boldsymbol{y}) + \int_0^1 \nabla^T u(\boldsymbol{y} + \tau(\boldsymbol{x} - \boldsymbol{y}))\,(\boldsymbol{x} - \boldsymbol{y})\,d\tau$$

$$= u(\boldsymbol{y}) + \nabla^T u(\boldsymbol{y})\,(\boldsymbol{x} - \boldsymbol{y}) + \int_0^1 [\nabla u(\boldsymbol{y} + \tau(\boldsymbol{x} - \boldsymbol{y})) - \nabla u(\boldsymbol{y})]^T (\boldsymbol{x} - \boldsymbol{y})\,d\tau$$

Hence

$$\left|u(\boldsymbol{x}) - u(\boldsymbol{y}) - \nabla^T u(\boldsymbol{y})\,(\boldsymbol{x} - \boldsymbol{y})\right|$$

$$= \left|\int_0^1 [\nabla u(\boldsymbol{y} + \tau(\boldsymbol{x} - \boldsymbol{y})) - \nabla u(\boldsymbol{y})]^T (\boldsymbol{x} - \boldsymbol{y})\,d\tau\right|$$

$$\le \int_0^1 \left|[\nabla u(\boldsymbol{y} + \tau(\boldsymbol{x} - \boldsymbol{y})) - \nabla u(\boldsymbol{y})]^T (\boldsymbol{x} - \boldsymbol{y})\right| d\tau$$

$$\le \int_0^1 \left\|\nabla u(\boldsymbol{y}+\tau(\boldsymbol{x}-\boldsymbol{y}))-\nabla u(\boldsymbol{y})\right\|_2 \cdot \left\|\boldsymbol{x}-\boldsymbol{y}\right\|_2 d\tau$$

$$\le \int_0^1 \tau L \left\|\boldsymbol{x}-\boldsymbol{y}\right\|_2^2 d\tau = \frac{L}{2}\left\|\boldsymbol{x}-\boldsymbol{y}\right\|_2^2$$

The proof is complete. ∎

If we denote the upper bound in (4.14) by $q(\boldsymbol{x}, \boldsymbol{y})$, we can write $q(\boldsymbol{x}, \boldsymbol{y})$ as a complete square plus a constant:

$$q(\boldsymbol{x},\boldsymbol{y}) = u(\boldsymbol{y}) + \nabla^T u(\boldsymbol{y})(\boldsymbol{x}-\boldsymbol{y}) + \frac{L}{2}\left\|\boldsymbol{x}-\boldsymbol{y}\right\|_2^2 = \frac{L}{2}\left\|\boldsymbol{x}-\left(\boldsymbol{y}-\tfrac{1}{L}\nabla u(\boldsymbol{y})\right)\right\|_2^2 + \text{const} \qquad (4.15)$$

From (4.15) the minimizer of $q(\boldsymbol{x}, \boldsymbol{y})$ can readily be found as

$$\boldsymbol{x}^* = \boldsymbol{y} - \tfrac{1}{L}\nabla u(\boldsymbol{y}) \qquad (4.16)$$

Fig. 4.13 illustrates the above analysis with an example of $u(x) = \frac{1}{2}(x-1)^2 - 4\log x + 3$ which is shown as a black solid curve. It is easy to verify that $u(x)$ is convex with a Lipschitz gradient (first-order derivative) with $L = 2$ over interval [2, 4]. At point $y = 3$, the dashed blue line was produced by the linear lower bound (see the left-hand side of (4.14)), namely,

$$u(y) + \nabla^T u(y)(x-y) = 0.6056 + 0.6667(x-3) = 0.6667x - 1.3944$$

while the dashed red curve was generated by the convex quadratic upper bound (see the right-hand side of (4.14)), i.e.,

$$u(y) + \nabla^T u(y)(x-y) + \frac{L}{2}\left\|x-y\right\|_2^2 = 0.6056 + 0.6667(x-3) + (x-3)^2$$

Notice the two bounds are tightest at point $y = 3$. By (4.16), the convex quadratic upper bound function has a unique global minimizer at $x^* = 3 - 0.6667/2 = 2.6667$ which is much closer to the true minimizer of $u(x)$ relative to what the "preceding iterate" $y = 3$ is to the true minimizer. Together, Eqs. (4.15) and (4.16) play a fundamental role in the development of *proximal point algorithms* which we shall study below.

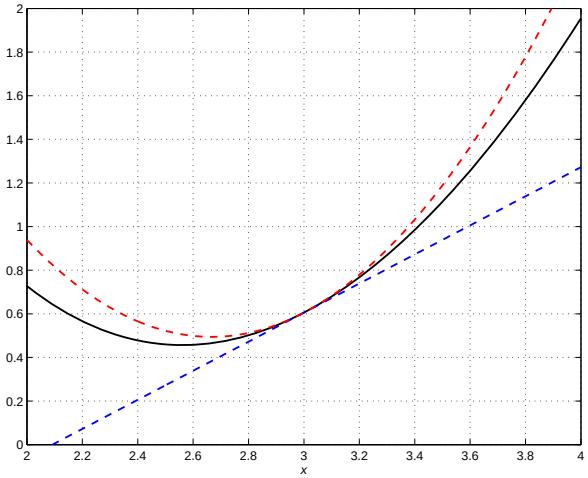


Fig. 4.13 An illustration of property (4.14).

We now develop a solution method for the L_2-L_1 problem

$$\text{minimize} \quad F(\boldsymbol{x}) = \tfrac{1}{2} \| \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \|_2^2 + \lambda \| \boldsymbol{x} \|_1 \tag{4.17}$$

where $\boldsymbol{A}$ is an *arbitrary* matrix. As the first step of the solution method, we consider a simple special case of (4.17) with matrix $\boldsymbol{A}$ being the *identity matrix*, thus the problem at hand becomes

$$\text{minimize} \quad F(\boldsymbol{x}) = \tfrac{1}{2} \| \boldsymbol{x} - \boldsymbol{b} \|_2^2 + \lambda \| \boldsymbol{x} \|_1 \tag{4.18}$$

The objective function in (4.18) can be expressed as

$$F(\boldsymbol{x}) = \sum_{i=1}^{n} \left[\tfrac{1}{2}(x_i - b_i)^2 + \lambda | x_i | \right] \tag{4.19}$$

Therefore, minimizing $F(\boldsymbol{x})$ in (4.19) can be accomplished by minimizing n single-variable problems of the form

$$\underset{x}{\text{minimize}} \quad \tfrac{1}{2}(x - b)^2 + \lambda | x | \tag{4.20}$$

Fig. 4.14 illustrates the component functions in problem (4.20), assuming $b = 1$ and $\lambda = 0.6$. It is easy to verify that for arbitrary b and $\lambda > 0$, the global minimizer of (4.20) is given by

$$x^* = \begin{cases} \text{sign}(b)(|b| - \lambda) & \text{if } |b| \geq \lambda \\ 0 & \text{if } |b| < \lambda \end{cases} \tag{4.21}$$

Based on (4.21), the global minimizer of problem (4.18) is given by

$$\boldsymbol{x}^* = \text{sign}(\boldsymbol{b}) \cdot \max\{|\boldsymbol{b}| - \lambda, \boldsymbol{0}\} \tag{4.22a}$$

with the expression on the right-hand side performed *componentwisely*. From (4.22a) we see that the solution can be found by "shrinking" componentwisely data vector $\boldsymbol{b}$ by λ towards the origin for those b_i with magnitude larger than λ and setting other components to zero. Hence the method is known as a *soft-shrinkage* method. In what follows, we use the notation $\mathcal{S}_\lambda(\boldsymbol{b})$ to denote the right-hand side of (4.22), i.e.,

$$\mathcal{S}_\lambda(\boldsymbol{b}) \triangleq \text{sign}(\boldsymbol{b}) \cdot \max\{|\boldsymbol{b}| - \lambda, \boldsymbol{0}\} \tag{4.22b}$$

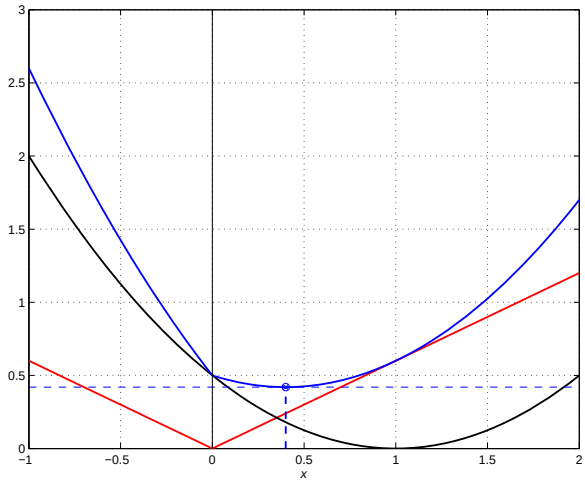


Fig. 4.14 The blue curve represents the objective function in (4.20) with $b = 1$ and $\lambda = 0.6$. Its 1st and 2nd terms are displayed in black and red, respectively. The function reaches its global minimum at $x = 0.4$, verifying the formula in (4.21).

We remark that the solution method described above is non-iterative and only requires a small amount of computations. This is made possible because the objective function in (4.18) is *separable* in its variables (as can be seen explicitly from (4.19)) so that one can treat the problem as a set of simple one-variable convex problems.

We are now ready to tackle the general L_2-L_1 problem in (4. 17). The objective functions in (4.17) is convex and of the form $F(\boldsymbol{x}) = u(\boldsymbol{x}) + \psi(\boldsymbol{x})$ where $u(\boldsymbol{x}) = \frac{1}{2} \| \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \|_2^2$ possesses Lipschitz continuous gradient because $\nabla u(\boldsymbol{x}) = \boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})$ with Lipschitz constant $L = \lambda_{\max}(\boldsymbol{A}^T\boldsymbol{A})$. Recall (4.14) that offers such a $u(\boldsymbol{x})$ with a global upper bound that is strongly convex and quadratic with *separate* variables and is tight at point $\boldsymbol{x}_k$. We take this as a motivation to consider sequentially minimizing the so-called *proximal-point function* for objective function $F(\boldsymbol{x})$. In the kth iteration with point $\boldsymbol{x}_k$ given, the proximal-point function is defined as

$$v(\boldsymbol{x}, \boldsymbol{x}_k) = u(\boldsymbol{x}_k) + \nabla^T u(\boldsymbol{x}_k)(\boldsymbol{x} - \boldsymbol{x}_k) + \tfrac{L}{2} \| \boldsymbol{x} - \boldsymbol{x}_k \|_2^2 + \lambda \| \boldsymbol{x} \|_1 \tag{4.23}$$

which is an upper bound of $F(\boldsymbol{x})$. To this end, we combine the first three terms of $v(\boldsymbol{x}, \boldsymbol{x}_k)$ as

$$u(\boldsymbol{x}_k) + \nabla^T u(\boldsymbol{x}_k)(\boldsymbol{x} - \boldsymbol{x}_k) + \tfrac{L}{2} \| \boldsymbol{x} - \boldsymbol{x}_k \|_2^2 = \tfrac{L}{2} \left\| \boldsymbol{x} - \left(\boldsymbol{x}_k - \tfrac{1}{L} \nabla u(\boldsymbol{x}_k) \right) \right\|_2^2 + \text{constant}$$

hence up to a constant the proximal-point function becomes

$$v(\boldsymbol{x}, \boldsymbol{x}_k) = \tfrac{L}{2} \left\| \boldsymbol{x} - \left(\boldsymbol{x}_k - \tfrac{1}{L} \nabla u(\boldsymbol{x}_k) \right) \right\|_2^2 + \lambda \| \boldsymbol{x} \|_1 \tag{4.24}$$

or simply

$$v(\boldsymbol{x}, \boldsymbol{x}_k) = \tfrac{L}{2} \left\| \boldsymbol{x} - \boldsymbol{b}_k \right\|_2^2 + \lambda \| \boldsymbol{x} \|_1 \tag{4.25a}$$

where

$$\boldsymbol{b}_k = \boldsymbol{x}_k - \tfrac{1}{L} \nabla u(\boldsymbol{x}_k) \tag{4.25b}$$

On comparing (4.25) with (4.17), we see that in (4.25) the term $\frac{L}{2} \left\| \boldsymbol{x} - \boldsymbol{b}_k \right\|_2^2$ replaces the term $\frac{1}{2} \| \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \|_2^2$, making (4.25) considerably easier to solve relative to that of (4.17), because the variables in (4.25a) are *separate*. The algorithmic steps of this method are sketched below.

Algorithm for (4.17) Using Sequential Minimization of Proximal-Point Functions

- Input: data $\boldsymbol{A}$ and $\boldsymbol{b}$, regularization parameter λ, initial point $\boldsymbol{x}_0$, Lipschitz constant L, and number of iterations K. Set $k = 0$.
- While $k < K$, do

  (1) Compute $\boldsymbol{x}_{k+1} = S_{\lambda/L} \left\{ \dfrac{1}{L} \boldsymbol{A}^T \left(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k \right) + \boldsymbol{x}_k \right\}$

  (2) Set $k = k + 1$;

  End
- Output solution $\boldsymbol{x}^* = \boldsymbol{x}_K$.

B.2 ***Regularization with L_1 Penalty***

Again we employ an L_2-loss function to measure the in-sample error

$$E_{\text{in}}(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N} | h(x_n, \boldsymbol{w}) - f(x_n) |^2 = \frac{1}{2}\sum_{n=1}^{N} | h(x_n, \boldsymbol{w}) - y_n |^2 = \frac{1}{2} \| \boldsymbol{X}\boldsymbol{w} - \boldsymbol{y} \|_2^2$$

but instead of using an L_2 penalty term, here we use an L_1 penalty $\Omega(h) = \| \boldsymbol{w} \|_1$ that leads to an L_1-regularized LS problem as

$$\underset{\boldsymbol{w}}{\text{minimize}} \quad \tfrac{1}{2} \| \boldsymbol{X}\boldsymbol{w} - \boldsymbol{y} \|_2^2 + \lambda \| \boldsymbol{w} \|_1 \tag{4.26}$$

Up to the difference in notation, the problems in (4.17) and (4.26) are identical, therefore the algorithm developed in Sec. 4.2.C.1 is directly applicable to (4.26).

Again we consider the first example studied in Sec. 4.2.A where the target function was a 10^{th} order polynomial and the problem was formulated as an L_2-regularized 10^{th}-order LS problem. Here we re-formulate the problem as an L_1-regularized LS problem instead, as seen in (4.26).

The data matrices $\boldsymbol{X}$ and $\boldsymbol{y}$ were identical to those from the example in Sec. 4.2.A. With $\lambda = 0.03$, the algorithm converged to the global minimizer of problem (4.26) after 10^4 iterations (soft-shrinkages). The numerical values of $\boldsymbol{w}^*$ are shown below together with the optimal solution obtained by solving the L_2-regularied LS problem for comparison purposes (in descending order). Performance of the L_1-regularized solution is depicted in Fig. 4.15. On comparing it with the L_2-regularied solution (see Fig. 4.10), the L_1 solution offers slightly reduced E_{out}. In addition, the sparse solution obtained by L_1 regularization helps identify a 6^{th}-order (6^{th}) polynomial model

L_1-regularied, $\lambda = 0.03$	L_2-regularied, $\lambda = 0.015$
0	-0.494990088844840
0	-0.173442588201798
0	-0.824953376793348
0	-0.262285486372259
-2.474821442899334	-1.121227504561928
-0.670748903773600	-0.352431834177799
0	-0.772568032119878
-0.034628447591051	-0.305879796539401
2.160179872543816	2.497167235882199
0.933464058245025	1.036020538249031
-0.585403475901743	-0.632437434927223

(with one zero coefficient), hence faster implementation. The MATLAB code used for the above simulation is enclosed below.

```
% L1-regularized least-squares polynomial fitting.
% Input:
% N: Data size.
% Q: Order of the target polynomial.
% M: Order of the fitting polynomial.
% lam: value of regularization parameter lambada.
% sig: standard deviation of the noise.
```

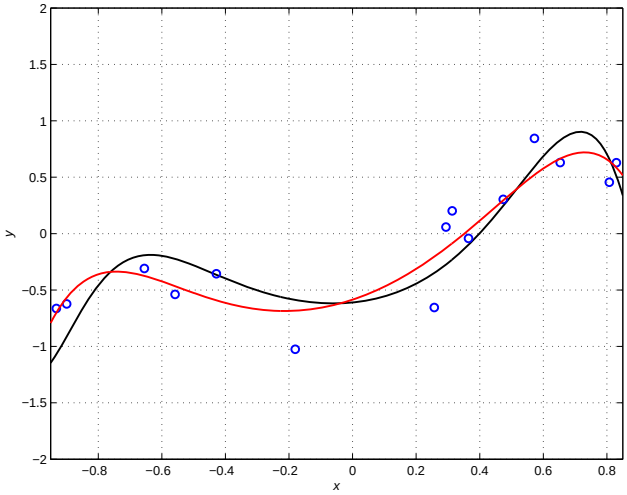


Figure 4.15 L_1-regularized 10^{th} order LS fit (red curve) with $\lambda = 0.03$.

	E_{in}	E_{out}
10^{th} order LS fit with L_2-regularization ($\lambda = 0.015$)	0.0408	0.1218
10^{th} order LS fit with L_1-regularization ($\lambda = 0.03$)	0.0436	0.1178

```
% st1: initial state for data samples.
% st2: initial state for target polynomial.
% st3: initial state for noise.
% K: number of iterations (soft shrinkage).
% Output:
% ws: Coeff. of fitting polynomial.
% Ein: In-sample error.
% Eout: Out-of-sample error.
% Written by W.-S. Lu, University of Victoria.  Last modified: March 16, 2015.
% Example:
% [ws1,Ein1,Eout1] = ex4_1a_re1a(15,10,10,0.03,0.3,148,119,18,10000);
function [ws,Ein,Eout] = ex4_1a_re1a(N,Q,M,lam,sig,st1,st2,st3,K)
rand('state',st1),
x = sort(2*(rand(N,1)-0.5));
randn('state',st2)
p = 5*randn(1,Q+1);
y = polyval(p,x);
randn('state',st3)
w = sig*randn(N,1);
yn = y + w;
V = fliplr(vander(x));
X = V(:,1:(M+1));
xx = X'*X;
L = max(eig(xx));
yh = (1/L)*X'*yn;
Xh = (1/L)*xx;
Lh = lam/L;
wk = zeros(M+1,1);
```

```
k = 0;
while k < K,
   bk = yh - Xh*wk + wk;
   abk = abs(bk) - Lh;
   wk = sign(bk).*(max(abk,0));
   k = k + 1;
end
ws = flipud(wk);
zn = polyval(ws,x);
xt = -0.95:1.8/1000:0.85;
yt = polyval(p,xt);
zt = polyval(ws,xt);
figure(1)
plot(xt,yt,'k-','linewidth',1.5)
hold on
plot(x,yn,'bo','linewidth',1.5)
plot(xt,zt,'r-','linewidth',1.5')
grid
axis([-0.95 0.85 -2 2])
xlabel('\itx')
ylabel('\ity')
hold off
Ein = (norm(yn-zn)^2)/N;
randn('state',st3)
wn = sig*randn(1,1001);
ytn = yt + wn;
Eout = (norm(ytn-zt)^2)/1001;
disp('in-sample error and out-of-sample error:')
[Ein Eout]
```

4.3 Cross Validation

Validation and cross validation are useful techniques for estimating E_{out} [3],[8]. These techniques can also be applied to model selection and estimation of regularization parameter λ.

A. *Validation Set*

Given a data set $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, N\}$, we partition it into a training set $\mathcal{D}_{\text{train}}$ of size $N - K$ and a *validation set* $\mathcal{D}_{\text{val}}$ of size K. As expected, $\mathcal{D}_{\text{train}}$ is used in learning process like before, except that its size is reduced from N to $N - K$. We denote the final hypothesis identified based on $\mathcal{D}_{\text{train}}$ and a learning model of choice by $g^-(\boldsymbol{x})$. The role of the validation set is nearly the same as a test set in the sense that it is *independent* of the training data and it is used to compute a *validation error*, denoted by $E_{\text{val}}(g^-)$, which may be regarded as an estimate of the out-of-sample error of $g^-(\boldsymbol{x})$:

$$E_{\text{val}}(g^-) = \frac{1}{K} \sum_{\boldsymbol{x}_n \in \boldsymbol{\mathcal{D}}_{\text{val}}} e\left(g^-(\boldsymbol{x}_n), y_n\right) \tag{4.27}$$

where is the pointwise error measure. In the case of classification, this error measure becomes $[\![g^-(\boldsymbol{x}_n) - y_n]\!]$, while it is $\left(g^-(\boldsymbol{x}_n) - y_n\right)^2$ for regression. The expected validation error over all possible $\mathcal{D}_{\text{val}}$ turns out to be equal to $E_{\text{out}}(g^-)$ because

$$E_{\boldsymbol{\mathcal{D}}_{\text{val}}}\left[E_{\text{val}}(g^-)\right]=\frac{1}{K}\sum_{\boldsymbol{x}_n\in\boldsymbol{\mathcal{D}}_{\text{val}}}E_{\boldsymbol{\mathcal{D}}_{\text{val}}}\left[e\left(g^-(\boldsymbol{x}_n),y_n\right)\right]=\frac{1}{K}\sum_{\boldsymbol{x}_n\in\boldsymbol{\mathcal{D}}_{\text{val}}}E_{\text{out}}(g^-)=E_{\text{out}}(g^-) \tag{4.28}$$

Eq. (4.28) says that in average $E_{\text{val}}(g^-)$ is the same as the out-of-sample error at g^-. So how reliable is $E_{\text{val}}(g^-)$ in estimating $E_{\text{out}}(g^-)$ depends on the variance of $E_{\text{val}}(g^-)$. It can be shown that in a classification problem [8]

$$\text{var}\left[E_{\text{val}}(g^-)\right]\le\frac{1}{4K} \tag{4.29}$$

hence $E_{\text{val}}(g^-)$ as an estimate of $E_{\text{out}}(g^-)$ becomes more accurate as the size of the validation set gets larger. Another way to see the usefulness of $E_{\text{val}}(g^-)$ in estimating $E_{\text{out}}(g^-)$ is through the VC bound when it is applied to the scenario where $\mathcal{D}_{\text{val}}$ acts as the training set and the hypothesis set contains only a member g^-, thus $E_{\text{val}}(g^-)$ is equal to $E_{\text{in}}(g^-)$ and the VC bound gives

$$E_{\text{out}}(g^-)\le E_{\text{val}}(g^-)+O\left(\frac{1}{\sqrt{K}}\right) \tag{4.30}$$

From both (4.29) and (4.30), we see that the size K of the validation set has to be large for $E_{\text{val}}(g^-)$ to be a reliable estimate of $E_{\text{out}}(g^-)$. On the other hand, K has to be small such that the size $N - K$ of the training set is big enough for hypothesis g^- to offer a decent generalization performance. A rule of thumb is to set 20% of the data (i.e. $K = N/5$) for validation [8].

We now conclude this sub-section with a remark on "restoring" the data set $\mathcal{D}$. Here the restoration is the sense that once the validation set has been used in the estimation of E_{out} of hypothesis g^-, one does not have to take g^- as the final hypothesis. As long as $E_{\text{out}}(g^-)$ is considered acceptable, an improved (and final) hypothesis g can be found by learning that uses the entire data set $\mathcal{D}$ instead of using only a part of it with $N - K$ samples. From above analysis and (4.30), it follows that

$$E_{\text{out}}(g)\le E_{\text{out}}(g^-)\le E_{\text{val}}(g^-)+O\left(\frac{1}{\sqrt{K}}\right) \tag{4.31}$$

In summary, one purpose of utilizing a validation set is to guide the learning process and it is in this regard that a validation set differs from a test set [8].

B. ***Model Section by Validation***

An important use of validation is for model selection. Here the term model selection refers to a variety of things including the selection of linearity/nonlinearity of the learning model, complexity of the model perhaps in terms of order of the polynomials involved, value of regularization parameter, etc. A point to stress is that it is validation's ability to estimate the out-of-sample error -- not only for one model but for a given number of models that makes validation a popular approach for model selection.

For description simplicity, suppose a certain learning algorithm has been chosen so that model difference can be characterized in terms of hypothesis set. Let us now consider the problem of selecting a model from M models represented by M hypothesis sets $\mathcal{H}_1$, $\mathcal{H}_2$, …, $\mathcal{H}_M$. In what

follows we use a common data set $\mathcal{D}$ that is partitioned into a training set $\mathcal{D}_{\text{train}}$ and a validation set $\mathcal{D}_{\text{val}}$. For each hypothesis set, say $\mathcal{H}_m$, training set $\mathcal{D}_{\text{train}}$ is used to learn a final hypothesis g_m^- for which validation set $\mathcal{D}_{\text{val}}$ is used to evaluate error $E_m = E_{\text{val}}(g_m^-)$, which estimates the out-of-sample error $E_{\text{out}}(g_m^-)$. If m^* is the index such that E_{m^*} achieves the minimum of $\{E_m,\ m = 1, 2, \ldots, M\}$, then based on the validation errors model $\mathcal{H}_{m^*}$ is selected. To estimate $E_{\text{out}}(g_{m^*}^-)$, we consider a *new* learning problem with data set $\mathcal{D}_{\text{val}}$ and hypothesis set, $\mathcal{H}_{\text{val}}$, that contains M hypotheses: $\mathcal{H}_{\text{val}} = \{g_1^-, g_2^-, \cdots, g_M^-\}$ where each g_m^- was obtained above. Note that $E_{\text{val}}(g_{m^*}^-)$ is within the hypothesis set $\mathcal{H}_{\text{val}}$ and $E_{\text{val}}(g_{m^*}^-)$ can be viewed as "in-sample" error for $E_{\text{val}}(g_{m^*}^-)$. Applying the VC bound in this case leads to

$$E_{\text{out}}(g_{m^*}^-) \le E_{\text{val}}(g_{m^*}^-) + O\left(\sqrt{\frac{\log M}{K}}\right) \tag{4.32}$$

Once the model $\mathcal{H}_{m^*}$ is selected, as the last step of the learning process $\mathcal{H}_{m^*}$ is utilized together with the *entire* data set $\mathcal{D}$ in the learning to identify a final hypothesis g_{m^*}. From (4.32), it follows that

$$E_{\text{out}}(g_{m^*}) \le E_{\text{out}}(g_{m^*}^-) \le E_{\text{val}}(g_{m^*}^-) + O\left(\sqrt{\frac{\log M}{K}}\right) \tag{4.33}$$

C. ***Cross Validation***

From what described above we see that validation faces a dilemma when it comes to determine the size of validation set (K) versus the size of training set ($N - K$). In the extreme case of $K = 1$, the "final" hypothesis g^- obtained based on $N - 1$ data points and the final hypothesis g one would get based on the entire data set of N points are expected to be practically the same, consequently the difference between $E_{\text{out}}(g^-)$ and $E_{\text{out}}(g)$ is minimal. One the other hand, estimation of $E_{\text{out}}(g^-)$ as seen in (4.30) becomes unreliable as term $E_{\text{val}}(g^-)$ is obtained using only one data point and with $K = 1$ term $O\left(\frac{1}{\sqrt{K}}\right)$ becomes fairly large.

Cross validation (CV) is an effective approach to resolving the dilemma. In an M-fold CV [3], the data is split into M (approximately) equal-sized parts. One then carries out validation M times, each time using the mth data part as the validation set and the remaining data as the training set, for $m = 1, 2, \ldots, M$. This yields a total of M "final" hypotheses $\{g_1^-, g_2^-, \cdots, g_M^-\}$. Let $E_{\text{val}}(g_m^-)$ be the validation error obtained by evaluating the error of g_m^- over the mth data part (see (4.27)), the *CV estimate* is defined as the average value of $\{E_{\text{val}}(g_m^-),\ m = 1, 2, \ldots, M\}$, namely

$$E_{\text{cv}} = \frac{1}{M}\sum_{m=1}^{M} E_{\text{val}}(g_m^-) \tag{4.34}$$

In the particular case of $M = N$, a data set of size N is partitioned into N data parts, i.e. each part contains a single data point, the M-fold CV is reduced to the *leave-one-out* cross validation

(LOO-CV). Eq. (4.34) in this case becomes

$$E_{\text{cv}} = \frac{1}{N}\sum_{n=1}^{N} e_n, \quad e_n = e\left(g_n^-(\boldsymbol{x}_n), y_n\right) \tag{4.35}$$

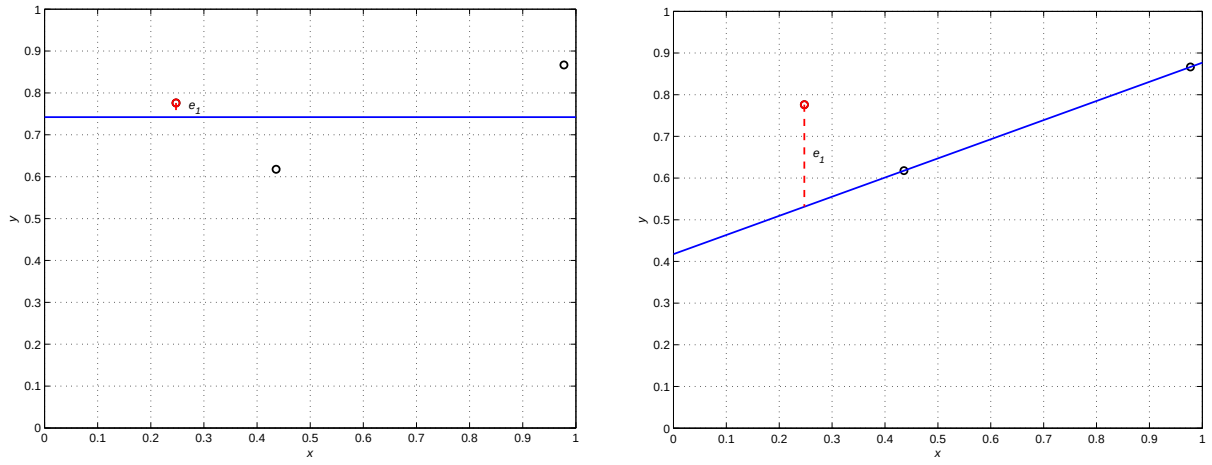
It can be shown [8] that for LOO-CV the expected E_{cv} with respect to all size-N data sets is the same as the expected $E_{\text{out}}(g)$ with respect to all size-(N–1) data sets, i.e., $\mathcal{E}_{\mathcal{D}_N}[E_{\text{cv}}] = \mathcal{E}_{\mathcal{D}_{N-1}}[E_{\text{out}}(g)]$. In other words, E_{cv} is an *unbiased* estimate of $\mathcal{E}_{\mathcal{D}_{N-1}}[E_{\text{out}}(g)]$. In practice the reliability of using E_{cv} to predict $E_{\text{out}}[g]$ has been found pretty high although little can be said theoretically.

Another use of CV is for model selection. Model selection was addressed in Sec. 4.3.B by validation. Here we approach the same problem via cross validation. Consider selecting a model from M models represented by M hypothesis sets $\mathcal{H}_1$, $\mathcal{H}_2$, …, $\mathcal{H}_M$. For each hypothesis set, say $\mathcal{H}_m$, cross validation is carried out to yield $E_{\text{cv}}^{(m)}$. Let m^* be the index with $E_{\text{cv}}^{(m^*)}$ reaching the minimum of $\{E_{\text{cv}}^{(m)}, m = 1, 2, \ldots, M\}$, then model $\mathcal{H}_{m^*}$ is selected. A final hypothesis g will then be identified from $\mathcal{H}_{m^*}$ using the entire data set, whose out-of-sample error $E_{\text{out}}(g)$ may be estimated by $E_{\text{cv}}^{(m^*)}$.

Example 4.5 [8]

Suppose we have a data set of three points $\mathcal{D} = \{(x_i, y_i), i = 1, 2, 3\}$ that are produced from a target function over [0, 1] that equals a constant c plus Gaussian noise with zero mean and variance σ^2 with $\sigma = 0.1$. We consider two models: $\mathcal{H}_1$:$\{h(x) = ax + b\}$ and $\mathcal{H}_2$: $\{h(x) = b\}$ where a and b are real-valued parameters. We apply cross validation to both models, the CV errors for the linear and constant models are denoted by $E_{\text{cv}}^{(L)}$ and $E_{\text{cv}}^{(C)}$, respectively. An example situation is illustrated in Fig. 4.16. Fig. 4.16(a) shows error term e_1 when CV was applied to the constant model and the first data pair (x_1, y_1) was taken out. Fig. 4.16(b) shows error term e_1 when CV was applied to the linear model and the first data pair (x_1, y_1) was taken out. It was found that $E_{\text{cv}}^{(C)} = 0.0238$ and $E_{\text{cv}}^{(L)} = 0.3278$. Because $E_{\text{cv}}^{(C)} < E_{\text{cv}}^{(L)}$, the cross validation helps make the right decision of selecting the constant model.

Yet another use of cross validation is for selecting regularization parameter λ. This is not surprising because λ is considered as a part of a learning model as it is involved in both a regularized objective function in a way that effectively imposes a constraint on "how much" of the hypothesis set can actually be used in searching a final hypothesis $g(\boldsymbol{x})$. Let $\{\lambda_m, m = 1, 2, \ldots, M\}$ be M values for the regularization parameter over a range that contains the optimal λ^*. We consider selecting a model from M models represented by $(\mathcal{H}, \lambda_1)$, $(\mathcal{H}, \lambda_2)$, …, $(\mathcal{H}, \lambda_M)$. For each hypothesis set, say $(\mathcal{H}, \lambda_m)$, cross validation is carried out to yield $E_{\text{cv}}^{(m)}$. Let m^* be the index



(a) (b)

Fig. 4.16 Instants of leave-one-out CV errors for (a) constant and (b) linear models.

with $E_{\text{cv}}^{(m^*)}$ reaching the minimum of $\{E_{\text{cv}}^{(m)}, m = 1, 2, \ldots, M\}$, then model $(\mathcal{H}, \lambda_{m^*})$ is selected. A final hypothesis g will then be identified from $(\mathcal{H}, \lambda_{m^*})$ using the entire data set, whose out-of-sample error $E_{\text{out}}(g)$ may be estimated by $E_{\text{cv}}^{(m^*)}$.

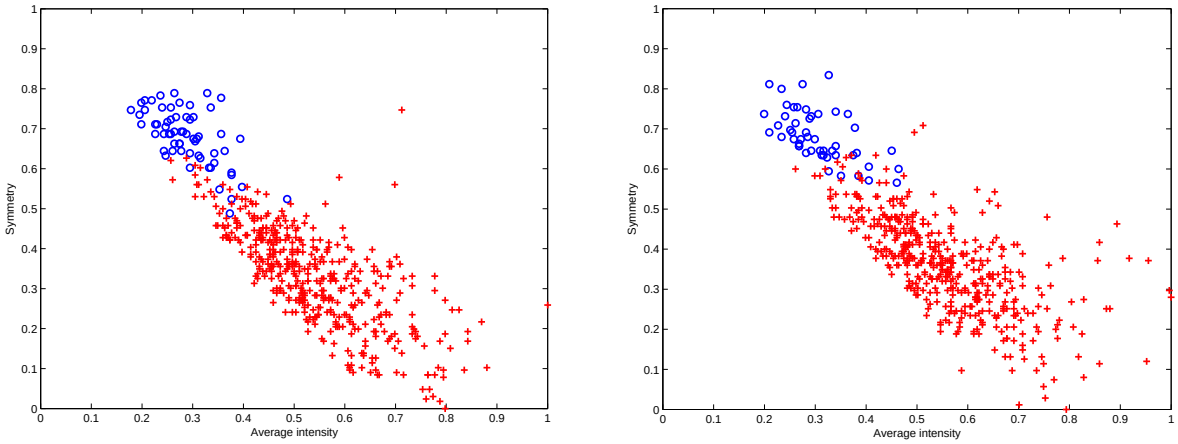
Example 4.6

This example illustrates the use of LOO-CV in estimating E_{out} in connection with model selection. Here we consider the problem of classifying a handwritten digit as either "1" or else. The *raw* training data consists of an array of size 784 by 500 representing 500 handwritten digits that were selected at random from MNIST database together with their labels. It has been noted that digit 1 has smaller average intensity relative to other digits and possesses symmetry measured by the difference between image "1" and the one by flipping it from left to right then from upside down will be small relative to the differences associated with other digits [8]. Based on these observations a 2-dimensional feature vector can be generated for each sample in the training set thus producing a training set of size 2 by 500. Fig. 4.17a shows the samples of digit 1 from the training data (marked as circles in blue) and the samples of all other digits marked as pluses in red in the input space with the average intensity as horizontal axes and "one-minus-(normalized) symmetry" as vertical axis.

It is clear that the problem at hand is a classification problem and, from Fig. 4.17 we see the classification can be done reasonably well by a linear method working with a nonlinear dilated feature space $\mathcal{Z}$. In this example we examine a variety of models by gradually increasing the number of features from the simplest case of one feature i.e. $\{1, x_1, x_2\}$ to twenty features, i.e.

$$\{1, x_1, x_2, x_1^2, x_1x_2, x_2^2, x_1^3, x_1^2x_2, \cdots, x_1^5, x_1^4x_2, x_1^3x_2^2, x_1^2x_2^3, x_1x_2^4, x_2^5\}$$

The method of logistic regression with BFGS-BT was chosen for the classification problem. For each chosen number of features, classification was carried out and E_{in}, E_{cv} for LOO-CV, and E_{out} were evaluated, where E_{out} was calculated using a set testing data of 500 samples outside the training set, depicted in Fig. 4.17b, that were selected at random from MNIST database consisting 50 "1"s and 450 other digits.



(a) (b)

Figure 4.17 (a) training data and (b) testing data for Example 4.6.

The profiles of E_{in}, E_{cv}, and E_{out} versus the number of feature used are shown in Fig. 4.18. It is

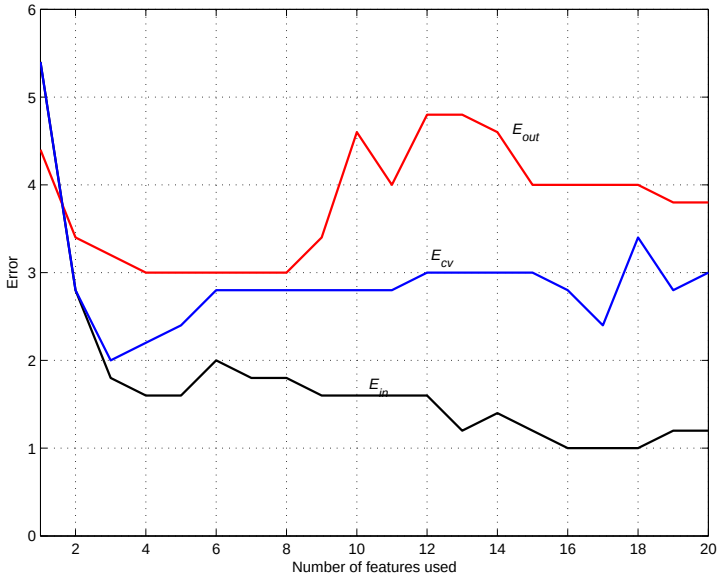


Figure 4.18 E_{in}, E_{cv}, and E_{out} versus the number of feature used for Example 4.6.

observed that the smallest E_{cv} values occurred when models using the first three or four features were employed and it was these models whose E_{out} were among the smallest. In this regard, a profile E_{cv}'s values versus model complexity is of help in selecting a right model for relative low E_{out}. It is also worth noticing that models with low E_{in} but high E_{cv} are not necessarily good ones as high E_{cv}'s usually indicate high E_{out}. ■

Problems

4.1 Polynomial models may be considered as a special family of linear models in dilated feature space $\mathcal{Z}$ under nonlinear transformation $\boldsymbol{z} = \Phi(\boldsymbol{x})$. So far conventional polynomials have been utilized in nonlinear transformations. Here we consider a linear model in space $\mathcal{Z}$ where Legendre polynomials up to order Q are utilized instead of a collection of monomials. Assume that there is only one basic feature, x, in the input space $\mathcal{X}$. Let

$$\boldsymbol{z} = \begin{bmatrix} 1 \\ L_1(x) \\ \vdots \\ L_Q(x) \end{bmatrix}$$

The hypothesis set $\mathcal{H}$ is given by

$$\mathcal{H} = \left\{ h : h(x) = \boldsymbol{w}^T \mathbf{z} = \sum_{q=0}^{Q} w_q L_q(x) \right\}$$

where $L_0(x) = 1$. For linear regression, one seeks to find a weight vector $\boldsymbol{w}^*$ that minimizes the L_2 in-sample error

$$E_{\text{in}}(\boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} \left(\boldsymbol{w}^T \mathbf{z}_n - y_n \right)^2$$

(i) Drive a closed-form formula for the global minimizer $\boldsymbol{w}^*$ of $E_{\text{in}}(\boldsymbol{w})$ where the Legendre polynomials involved are explicitly shown.
(ii) Now consider the case where the input space includes *two* basic features x_1 and x_2. To simplify the matter, you are asked to develop a closed-form formula for the minimizer $\boldsymbol{w}^*$ for $Q = 3$ by carrying out the following two steps.
Step 1: Recall in the single variable case the Legendre polynomials up to order Q are $L_0(x)$, $L_1(x)$, …, $L_Q(x)$. Write down all two-variable Legendre polynomials (in terms of single-variable Legendre polynomials) up to order $Q = 3$.
Step 2: Based on the outcome from Step 1, develop a closed-form formula for $\boldsymbol{w}^*$ where the single-variable Legendre polynomials are explicitly shown.

4.2 Recall that in Sec. 4.2.B.2 the modeling problem involving noisy data was addressed using an algorithm based on sequential minimization of proximal functions. Here you are asked to repeat the example (see pages 130 – 132) by using CVX instead of sequential minimization. This can be done by simply replacing the relevant part of the code `ex4_1a_re1a.m` with a much shorter block of CVX lines. Compare the two solutions in terms of E_{in} and E_{out} as well as solutions' sparsity.

4.3 Suppose we have a data set of three points $\mathcal{D} = \{(x_i, y_i), i = 1, 2, 3\}$ that are produced from a target function over [0, 1] that equals a linear function $f(x) = -0.6x + 0.8$ plus Gaussian noise with zero mean and variance σ^2 with $\sigma = 0.1$. The data set $\mathcal{D} = \{(x_i, y_i), i = 1, 2, 3\}$ is generated as follows:

```
rand('state',10),
```

```
x = rand(3,1)
randn('state',16)
w = 0.1*randn(3,1)
y0 = -0.6x + 0.8;
y = y0 + w;
```

We consider two models: $\mathcal{H}_0$: $\{h(x) = c\}$ and $\mathcal{H}_1$: $\{h(x) = ax + b\}$ where a, b, and c are real-valued parameters.

(i) Apply cross validation to both models and compute E_{cv} for the constant and linear models.

(ii) Based on the results from part (i), select a hypothesis.