CHAPTER

34 MULTITASK LEARNING

CHAPTER CONTENTS

When solving multiple related learning tasks, solving them together simultaneously by sharing information is expected to give a better solution than solving them independently. This is the basic idea of *multitask learning* [24]. In this chapter, practically useful multitask learning methods are introduced.

Let us consider T tasks indexed by $t = 1, \ldots, T$ and assume that each input-output paired training sample $(\boldsymbol{x}_i, y_i)$ is accompanied with task index t_i:

$$\{(\boldsymbol{x}_i, y_i, t_i) \mid t_i \in \{1, \ldots, T\}\}_{i=1}^{n}.$$

34.1 TASK SIMILARITY REGULARIZATION

In this section, a multitask learning method with *task similarity regularization* [40] is introduced.

34.1.1 FORMULATION

For the tth learning task, let us employ a linear-in-parameter model,

$$\sum_{j=1}^{b} \theta_{t,j} \phi_j(\boldsymbol{x}) = \boldsymbol{\theta}_t^\top \boldsymbol{\phi}(\boldsymbol{x}),$$

An Introduction to Statistical Machine Learning. DOI: 10.1016/B978-0-12-802121-7.00045-5

where $\boldsymbol{\theta}_t = (\theta_{t,1}, \ldots, \theta_{t,b})^\top$ is the parameter vector for the tth task and the basis functions $\boldsymbol{\phi}(\boldsymbol{x})$ are *common* to all tasks.

The idea of task similarity regularization is to impose the parameters $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_T$ to take similar values and learn all parameters,

$$\boldsymbol{\theta} = (\boldsymbol{\theta}_1^\top, \ldots, \boldsymbol{\theta}_T^\top)^\top \in \mathbb{R}^{bT},$$

simultaneously. Let us employ the *ℓ_2-regularized LS* (see Section 23.2) and learn $\boldsymbol{\theta}$ so that the following $J(\boldsymbol{\theta})$ is minimized:

$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{n}\left(y_i - \boldsymbol{\phi}(\boldsymbol{x}_i)^\top \boldsymbol{\theta}_{t_i}\right)^2 + \frac{1}{2}\sum_{t=1}^{T}\lambda_t \|\boldsymbol{\theta}_t\|^2 + \frac{1}{4}\sum_{t,t'=1}^{T}\gamma_{t,t'}\|\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t'}\|^2,$$

where $\lambda_t \geq 0$ is the ℓ_2-regularization parameter for the tth task, and $\gamma_{t,t'} \geq 0$ is the similarity between the tth task and the t'th task. If $\gamma_{t,t'} = 0$ for all $t, t' = 1, \ldots, T$, the third term in $J(\boldsymbol{\theta})$ disappears. Then there is no interaction between tasks and thus this corresponds to merely learning T tasks separately. On the other hand, if $\gamma_{t,t'} > 0$, $\boldsymbol{\theta}_t$ and $\boldsymbol{\theta}_{t'}$ are imposed to be close to each other and thus information can be implicitly shared between the tth task and the t'th task. If all $\gamma_{t,t'}$ are large enough, all solutions are imposed to be the same and thus a single solution that is common to all tasks is obtained from all training samples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$.

34.1.2 ANALYTIC SOLUTION

$J(\boldsymbol{\theta})$ can be compactly expressed as

$$J(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{\Psi}\boldsymbol{\theta}\|^2 + \frac{1}{2}\boldsymbol{\theta}^\top(\boldsymbol{C} \otimes \boldsymbol{I}_b)\boldsymbol{\theta}, \tag{34.1}$$

where

$$\begin{aligned}
\boldsymbol{y} &= (y_1, \ldots, y_n)^\top \in \mathbb{R}^n, \\
\boldsymbol{\Psi} &= (\boldsymbol{\psi}_{t_1}(\boldsymbol{x}_1), \ldots, \boldsymbol{\psi}_{t_n}(\boldsymbol{x}_n))^\top \in \mathbb{R}^{n \times bT}, \\
\boldsymbol{\psi}_t(\boldsymbol{x}) &= \left(\boldsymbol{0}_{b(t-1)}^\top, \boldsymbol{\phi}(\boldsymbol{x})^\top, \boldsymbol{0}_{b(T-t)}^\top\right)^\top \in \mathbb{R}^{bT}.
\end{aligned}$$

$\boldsymbol{C}$ is the $T \times T$ matrix defined as

$$C_{t,t'} = \begin{cases} \lambda_t + \sum_{t''=1}^{T}\gamma_{t,t''} - \gamma_{t,t} & (t = t'), \\ -\gamma_{t,t'} & (t \neq t'), \end{cases}$$

and $\otimes$ denotes the *Kronecker product*, i.e. for $\boldsymbol{E} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{F} \in \mathbb{R}^{p \times q}$,

$$\boldsymbol{E} \otimes \boldsymbol{F} = \begin{pmatrix} E_{1,1}\boldsymbol{F} & \cdots & E_{1,n}\boldsymbol{F} \\ \vdots & \ddots & \vdots \\ E_{m,1}\boldsymbol{F} & \cdots & E_{m,n}\boldsymbol{F} \end{pmatrix} \in \mathbb{R}^{mp \times nq}.$$

Then the minimizer $\widehat{\boldsymbol{\theta}}$ of $J(\boldsymbol{\theta})$ satisfies

$$(\boldsymbol{\Psi}^\top \boldsymbol{\Psi} + \boldsymbol{C} \otimes \boldsymbol{I}_b)\boldsymbol{\theta} = \boldsymbol{\Psi}^\top \boldsymbol{y}, \tag{34.2}$$

and $\widehat{\boldsymbol{\theta}}$ can be obtained analytically as

$$\widehat{\boldsymbol{\theta}} = \left(\boldsymbol{\Psi}^\top \boldsymbol{\Psi} + \boldsymbol{C} \otimes \boldsymbol{I}_b\right)^{-1} \boldsymbol{\Psi}^\top \boldsymbol{y}. \tag{34.3}$$

34.1.3 EFFICIENT COMPUTATION FOR MANY TASKS

The size of matrix $(\boldsymbol{\Psi}^\top \boldsymbol{\Psi} + \boldsymbol{C} \otimes \boldsymbol{I}_b)$ is $bT \times bT$, and thus directly computing the solution by the above analytic form is not tractable if the number of tasks, T, is large. However, since the rank of matrix $\boldsymbol{\Psi}^\top \boldsymbol{\Psi}$ is at most n, the solution can be computed efficiently if $n < bT$.

More specifically, the solution $\widehat{\boldsymbol{\theta}}_t^\top \boldsymbol{\phi}(\boldsymbol{x})$ for the tth task can be expressed as

$$\widehat{\boldsymbol{\theta}}_t^\top \boldsymbol{\phi}(\boldsymbol{x}) = \widehat{\boldsymbol{\theta}}^\top \boldsymbol{\psi}_t(\boldsymbol{x}) = \boldsymbol{y}^\top \boldsymbol{A}^{-1} \boldsymbol{b}_t. \tag{34.4}$$

Here, $\boldsymbol{A}$ and $\boldsymbol{b}_t$ are the $n \times n$ matrix and n-dimensional vector defined as

$$\begin{aligned}
A_{i,i'} &= [\boldsymbol{\Psi}(\boldsymbol{C}^{-1} \otimes \boldsymbol{I}_b)\boldsymbol{\Psi}^\top + \boldsymbol{I}_n]_{i,i'} \\
&= [\boldsymbol{C}^{-1}]_{t_i,t_{i'}} \boldsymbol{\phi}(\boldsymbol{x}_i)^\top \boldsymbol{\phi}(\boldsymbol{x}_{i'}) + \begin{cases} 1 & (i = i'), \\ 0 & (i \neq i'), \end{cases} \\
b_{t,i} &= [\boldsymbol{\Psi}(\boldsymbol{C}^{-1} \otimes \boldsymbol{I}_b)\boldsymbol{\psi}_t(\boldsymbol{x})]_i \\
&= [\boldsymbol{C}^{-1}]_{t,t_i} \boldsymbol{\phi}(\boldsymbol{x}_i)^\top \boldsymbol{\phi}(\boldsymbol{x}),
\end{aligned}$$

where $[\boldsymbol{C}^{-1}]_{t,t'}$ denotes the (t,t')th element of matrix $\boldsymbol{C}^{-1}$. Since the size of matrix $\boldsymbol{A}$ and the dimensionality of vector $\boldsymbol{b}_t$ are independent of the number of tasks, T, Eq. (34.4) would be computationally more efficient than Eq. (34.3) if $n < bT$. Moreover, since $\boldsymbol{A}^{-1}$ is independent of task index t, it needs to be computed only once across all tasks. Note that the following formulas regarding the Kronecker product were utilized in the derivation of Eq. (34.4):

$$\begin{aligned}
\boldsymbol{\Psi}\left(\boldsymbol{\Psi}^\top \boldsymbol{\Psi} + \boldsymbol{C} \otimes \boldsymbol{I}_b\right)^{-1} &= (\boldsymbol{\Psi}(\boldsymbol{C} \otimes \boldsymbol{I}_b)^{-1}\boldsymbol{\Psi}^\top + \boldsymbol{I}_n)^{-1}\boldsymbol{\Psi}(\boldsymbol{C} \otimes \boldsymbol{I}_b)^{-1}, \\
(\boldsymbol{C} \otimes \boldsymbol{I}_b)^{-1} &= \boldsymbol{C}^{-1} \otimes \boldsymbol{I}_b.
\end{aligned}$$

A MATLAB code for multitask LS is provided in Fig. 34.1, and its behavior is illustrated in Fig. 34.2. This shows that multitask classification outperforms single-task classification.

In the above multitask method, the task similarity $\gamma_{t,t'}$ was assumed to be known. When $\gamma_{t,t'}$ is unknown, it may be alternately learned as described in Fig. 34.3.

```
n=2; T=6; y=[ones(n/2,T); -ones(n/2,T)];
x=[randn(2,n,T); ones(1,n,T)]; r(1,1,:)=pi*[1:T]/T/10;
c=repmat(cos(r),[1 n/2]); x(1,:,:)=x(1,:,:)+[c -c];
s=repmat(sin(r),[1 n/2]); x(2,:,:)=x(2,:,:)+[s -s];
Ci=inv(-ones(T,T)+diag(T*ones(T,1)+0.01));
a=repmat([1:T],[n 1]); a=a(:); m=20; X=linspace(-4,4,m);
b=repmat(X,[m 1]); bt=b'; XX=[b(:)'; bt(:)'; ones(1,m^2)];
yAi=y(:)'*inv(Ci(a,a).*(x(:,:)'*x(:,:))+eye(n*T));

figure(1); clf; colormap([1 0.7 1; 0.7 1 1]);
for k=1:T
%  Y=yAi*(repmat(Ci(a,k),[1 3]).*x(:,:)')*XX;
  q=x(:,:,k); Y=((q*q'+0.01*eye(3))\(q*y(:,k)))'*XX;
  subplot(2,3,k); hold on; contourf(X,X,reshape(Y,m,m));
  plot(x(1,y(:,k)==1,k),x(2,y(:,k)==1,k),'bo');
  plot(x(1,y(:,k)==-1,k),x(2,y(:,k)==-1,k),'rx');
  plot(99*sin(r(k))*[1 -1],99*cos(r(k))*[-1 1],'k--');
  axis([-4 4 -4 4]);
end
```

FIGURE 34.1

MATLAB code for multitask LS.

34.2 MULTIDIMENSIONAL FUNCTION LEARNING

In this section, the problem of *multidimensional function learning* is discussed, which can be regarded as a special case of multitask learning.

34.2.1 FORMULATION

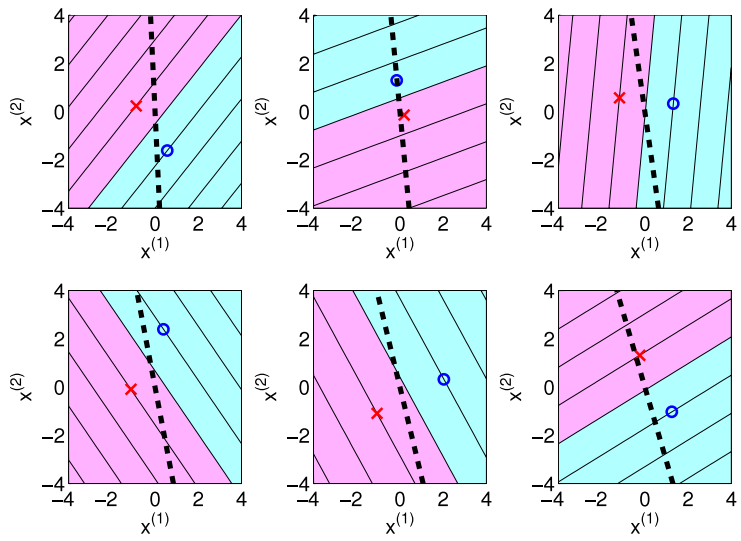
Let us consider the problem of learning a T-dimensional function,

$$\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots, f_T(\boldsymbol{x}))^\top,$$

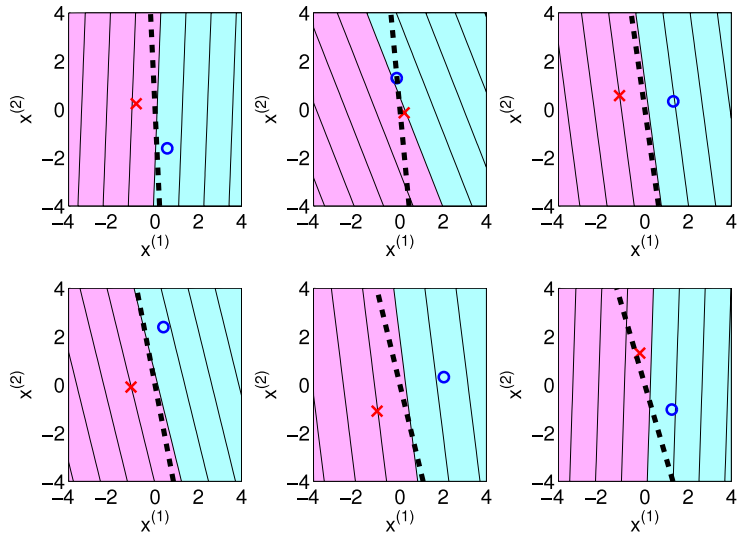
from input-output paired training samples:

$$\{(\boldsymbol{x}_i, \boldsymbol{y}_i) \mid \boldsymbol{x}_i \in \mathbb{R}^d,\ \boldsymbol{y}_i = (y_i^{(1)}, \ldots, y_i^{(T)})^\top \in \mathbb{R}^T\}_{i=1}^n.$$

If each dimension of output $\boldsymbol{y}$ is regarded as a task, the problem of multidimensional function learning can be regarded as multitask learning. The difference is that multidimensional function learning shares input points across all tasks, while input points are generally different in multitask learning (Fig. 34.4). Thus, the number of training samples in multidimensional function learning is actually nT in the context of multitask learning.



(a) Single-task classification



(b) Multitask classification

FIGURE 34.2

Examples of multitask LS. The dashed lines denote true decision boundaries and the contour lines denote learned results.

Multidimensional function learning for classification is specifically called *multilabel classification*, which can be regarded as a generalization of *multiclass*

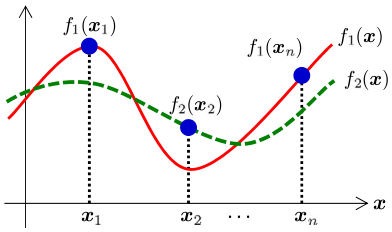
1. Initialize task similarity, e.g. $\gamma_{t,t'} = \gamma > 0, \forall t, t'$.
2. Learn parameter $\boldsymbol{\theta}$ based on the current task similarity $\gamma_{t,t'}$.
3. Update task similarity $\gamma_{t,t'}$ based on the similarity between $\boldsymbol{\theta}_t$ and $\boldsymbol{\theta}_{t'}$, e.g.

$$\gamma_{t,t'} = \eta \exp(-\kappa \|\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t'}\|^2),$$

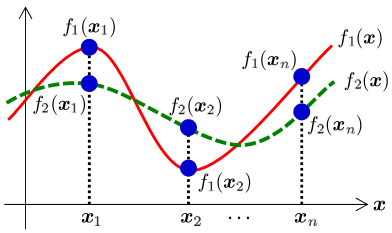
   where $\eta \geq 0$ and $\kappa \geq 0$ are tuning parameters and may be determined by cross validation.
4. Iterate 2–3 until convergence.

FIGURE 34.3

Alternate learning of task similarity $\gamma_{t,t'}$ and solution $\boldsymbol{\theta}$.



(a) Multitask regression



(b) Multidimensional regression

FIGURE 34.4

Multidimensional function learning.

classification (see Section 26.3). A pattern $\boldsymbol{x}$ belongs to one of the classes exclusively in multiclass classification, while $\boldsymbol{x}$ can belong to multiple classes simultaneously in multilabel classification. For example, an image can belong to the classes of "human," "dog," and "building" at the same time in image recognition, and a sound clip can belong to the classes of "conversation," "noise," and "background music" at the

same time in audio recognition. In multilabel classification, co-occurrence of multiple classes can be utilized by multitask learning. For example, an image containing a dog often contains a human at the same time.

However, in multidimensional function learning, the number of parameters is bT and the *actual* number of samples is nT, both of which are proportional to the number of tasks, T. Thus, both the solutions explained in Section 34.1.2 and Section 34.1.3 are not computationally efficient in multidimensional function learning.

34.2.2 EFFICIENT ANALYTIC SOLUTION

Let us arrange the parameter vectors $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_T$, the output vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$, and the basis vectors $\boldsymbol{\phi}(\boldsymbol{x}_1), \ldots, \boldsymbol{\phi}(\boldsymbol{x}_n)$ into matrices as

$$\begin{aligned}\boldsymbol{\Theta} &= (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_T) \in \mathbb{R}^{b \times T},\\ \boldsymbol{Y} &= (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)^\top \in \mathbb{R}^{n \times T},\\ \boldsymbol{\Phi} &= (\boldsymbol{\phi}(\boldsymbol{x}_1), \ldots, \boldsymbol{\phi}(\boldsymbol{x}_n))^\top \in \mathbb{R}^{n \times b}.\end{aligned}$$

Then, Eq. (34.2), which multitask solutions should satisfy, can be rewritten as

$$\boldsymbol{\Phi}^\top \boldsymbol{\Phi} \boldsymbol{\Theta} + \boldsymbol{\Theta} \boldsymbol{C} = \boldsymbol{\Phi}^\top \boldsymbol{Y}.$$

This form is known as the *continuous Sylvester equation*, which often arises in control theory [94], and can be solved efficiently, as explained in Fig. 34.5.

A MATLAB code for multidimensional regression is provided in Fig. 34.6, and its behavior is illustrated in Fig. 34.7. This shows that multitask regression outperforms single-task regression.

34.3 MATRIX REGULARIZATION

The multitask learning method introduced in Section 34.1 explicitly used task similarity $\gamma_{t,t'}$ to control the amount of information sharing. In this section, another multitask learning approach is introduced, which does not involve task similarity.

34.3.1 PARAMETER MATRIX REGULARIZATION

The basic idea is to share information across multiple tasks by regularizing the *parameter matrix*:

$$\boldsymbol{\Theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_T) \in \mathbb{R}^{b \times T}.$$

More specifically, for a squared loss function, the multitask learning criterion to be minimized with respect to $\boldsymbol{\Theta}$ is given by

$$\frac{1}{2} \sum_{i=1}^{n} \left(y_i - \boldsymbol{\phi}(\boldsymbol{x}_i)^\top \boldsymbol{\theta}_{t_i} \right)^2 + \lambda R(\boldsymbol{\Theta}),$$

where $R(\boldsymbol{\Theta})$ denotes some regularization functional for $\boldsymbol{\Theta}$.

The *continuous Sylvester equation* for some matrices $\boldsymbol{A} \in \mathbb{R}^{b \times b}$, $\boldsymbol{B} \in \mathbb{R}^{T \times T}$, and $\boldsymbol{Z} \in \mathbb{R}^{b \times T}$ with respect to $\boldsymbol{\Theta} \in \mathbb{R}^{b \times T}$ is given by

$$\boldsymbol{A\Theta} + \boldsymbol{\Theta B} = \boldsymbol{Z}.$$

The use of the *Kronecker product* and the *vectorization operator* (see Fig. 6.5) allows us to rewrite the above equation as

$$(\boldsymbol{I}_T \otimes \boldsymbol{A} + \boldsymbol{B} \otimes \boldsymbol{I}_b)\text{vec}(\boldsymbol{\Theta}) = \text{vec}(\boldsymbol{Z}),$$

where $\boldsymbol{I}_T$ denotes the $T \times T$ identity matrix. This shows that the continuous Sylvester equation with respect to a $b \times T$ matrix can be seen as a linear equation with respect to a bT-dimensional vector, which is computationally expensive to solve naively. However, the continuous Sylvester equation can be solved more efficiently. For example, let $u_1, \ldots, u_b$ and $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_b$ be *eigenvalues* and *eigenvectors* of $\boldsymbol{A}$ (see Fig. 6.2), and let $v_1, \ldots, v_T$ and $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_T$ be eigenvalues and eigenvectors of $\boldsymbol{B}$, respectively. Then, when $u_j + v_t \neq 0$ for all $j = 1, \ldots, b$ and $t = 1, \ldots, T$, the solution $\widehat{\boldsymbol{\Theta}}$ of the above continuous Sylvester equation with respect to $\boldsymbol{\Theta}$ can be obtained analytically as

$$\widehat{\boldsymbol{\Theta}} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_b)\boldsymbol{Q}(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_T)^\top,$$

where $\boldsymbol{Q}$ is the $b \times T$ matrix defined as

$$Q_{j,t} = \frac{\boldsymbol{u}_j^\top \boldsymbol{Z} \boldsymbol{v}_t}{u_j + v_t}.$$

FIGURE 34.5

Continuous Sylvester equation.

If the squared *Frobenius norm* $\|\boldsymbol{\Theta}\|_{\text{Frob}}^2$ is used for regularization, no information is shared across different tasks since this is equivalent to the sum of ℓ_2-norms of parameter vectors $\boldsymbol{\theta}_t$:

$$\|\boldsymbol{\Theta}\|_{\text{Frob}}^2 = \sum_{t=1}^{T} \sum_{j=1}^{b} \Theta_{t,j}^2 = \sum_{t=1}^{T} \|\boldsymbol{\theta}_t\|^2.$$

Thus, a more intricate norm should be used for matrix regularization.

For example, the *trace norm* (see Fig. 24.10) tends to produce a *low-rank* solution:

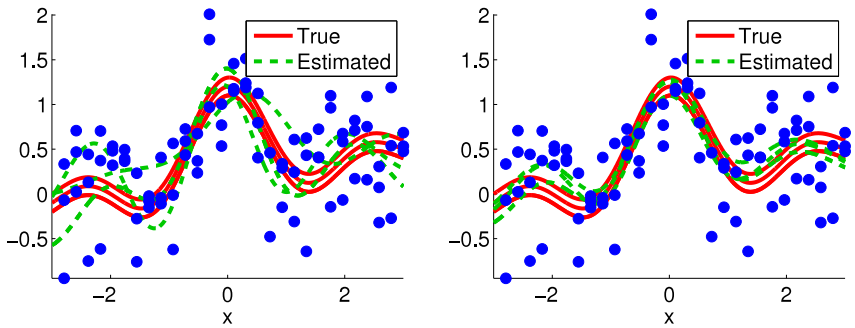
```
n=30; x=linspace(-3,3,n)'; pix=pi*x; T=3;
y=repmat(sin(pix)./(pix)+0.1*x,1,T)+0.1*repmat([1:T],n,1);
y=y+0.5*randn(n,T); N=1000; X=linspace(-3,3,N)'; piX=pi*X;
Y=repmat(sin(piX)./(piX)+0.1*X,1,T)+0.1*repmat([1:T],N,1);
G=10*ones(T,T); %G=zeros(T,T);
l=0.1; C=l*eye(T)+diag(sum(G))-G; hh=1; x2=x.^2;
k=exp(-(repmat(x2,1,n)+repmat(x2',n,1)-2*x*x')/hh);
K=exp(-(repmat(X.^2,1,n)+repmat(x2',N,1)-2*X*x')/hh);

[U,u]=eig(k'*k); [V,v]=eig(C);
Q=U'*k'*y*V./(repmat(diag(u),1,T)+repmat(diag(v)',n,1));
S=U*Q*V'; F=K*S;

figure(1); clf; hold on; axis([-inf inf -inf inf])
plot(X,Y,'r-'); plot(X,F,'g--'); plot(x,y,'bo');
```

FIGURE 34.6

MATLAB code for multidimensional regression.



(a) Single-task regression (b) Multitask regression

FIGURE 34.7

Examples of multidimensional regression.

$$\|\mathbf{\Theta}\|_{\text{tr}} = \sum_{k=1}^{\min(b,T)} \sigma_k,$$

where σ_k is a *singular value* of $\mathbf{\Theta}$ (see Fig. 22.2). Since a low-rank solution in multitask learning confines parameters of each task in a common subspace, information can be shared across different tasks [6]. In Section 23.1, the method

of *subspace-constrained LS* was introduced, which confines the LS solution in a given subspace. Multitask learning based on the trace norm can be regarded as automatically learning the subspace with the help of other learning tasks.

Another possibility of matrix regularization for multitask learning is to use the $\ell_{2,1}$-norm $\|\boldsymbol{\Theta}\|_{2,1}$:

$$\|\boldsymbol{\Theta}\|_{2,1} = \sum_{t=1}^{T} \sqrt{\sum_{j=1}^{b} \Theta_{t,j}^2},$$

which tends to give a *row-wise* sparse solution (see Section 24.4.4). This means that $\theta_{1,j}, \ldots, \theta_{T,j}$ tend to simultaneously vanish for several $j \in \{1, \ldots, b\}$, and thus *feature selection* can be performed across different tasks, if linear basis function $\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{x}$ is used [8] (see also Section 24.3).

Below, a practical optimization method for multitask LS with trace norm regularization is introduced.

34.3.2 PROXIMAL GRADIENT FOR TRACE NORM REGULARIZATION

Since the trace norm $\|\boldsymbol{\Theta}\|_{\mathrm{tr}}$ is a convex function, the global optimal solution to multitask LS with trace norm regularization can be obtained, for example, by the *proximal gradient method* described in Fig. 34.8.

More specifically, the proximal gradient method updates the parameter matrix $\boldsymbol{\Theta}$ from some initial value as

$$\boldsymbol{\Theta} \longleftarrow \mathrm{prox}\Big(\boldsymbol{\Theta} - \varepsilon \nabla L(\boldsymbol{\Theta})\Big), \tag{34.5}$$

where $\varepsilon > 0$ is the step size. In Eq. (34.5), $L(\boldsymbol{\Theta})$ is the loss function for all training samples:

$$L(\boldsymbol{\Theta}) = \frac{1}{2} \sum_{i=1}^{n} \Big(y_i - \boldsymbol{\phi}(\boldsymbol{x}_i)^\top \boldsymbol{\Theta} \boldsymbol{e}_{t_i}\Big)^2,$$

where $\boldsymbol{e}_t$ denotes the T-dimensional vector with all zeros but the tth element being one, i.e. $\boldsymbol{\Theta}\boldsymbol{e}_t = \boldsymbol{\theta}_t$. $\nabla L(\boldsymbol{\Theta})$ is the gradient of L given by

$$\nabla L(\boldsymbol{\Theta}) = \sum_{i=1}^{n} \Big(\boldsymbol{\phi}(\boldsymbol{x}_i)^\top \boldsymbol{\Theta} \boldsymbol{e}_{t_i} - y_i\Big) \boldsymbol{\phi}(\boldsymbol{x}_i) \boldsymbol{e}_{t_i}^\top,$$

where the following matrix derivative formula is used:

$$\frac{\partial \boldsymbol{\phi}(\boldsymbol{x}_i)^\top \boldsymbol{\Theta} \boldsymbol{e}_{t_i}}{\partial \boldsymbol{\Theta}} = \boldsymbol{\phi}(\boldsymbol{x}_i) \boldsymbol{e}_{t_i}^\top.$$

The *proximal operator* $\mathrm{prox}(\boldsymbol{\Theta})$ in Eq. (34.5) is given by

$$\mathrm{prox}(\boldsymbol{\Theta}) = \operatorname*{argmin}_{\boldsymbol{U}} \left(\|\boldsymbol{U}\|_{\mathrm{tr}} + \frac{1}{2\varepsilon\lambda} \|\boldsymbol{U} - \boldsymbol{\Theta}\|_{\mathrm{Frob}}^2 \right),$$

Let us consider the following optimization problem:

$$\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) + R(\boldsymbol{\theta}),$$

where $L(\boldsymbol{\theta})$ is a convex differentiable function and $R(\boldsymbol{\theta})$ is a closed convex function. The *proximal gradient method* finds the minimizer of $L(\boldsymbol{\theta}) + R(\boldsymbol{\theta})$ by updating $\boldsymbol{\theta}$ from some initial value as

$$\boldsymbol{\theta} \longleftarrow \operatorname*{argmin}_{\boldsymbol{u}} \left(R(\boldsymbol{u}) + L(\boldsymbol{\theta}) + \nabla L(\boldsymbol{\theta})^{\top}(\boldsymbol{u} - \boldsymbol{\theta}) + \frac{1}{2\varepsilon}\|\boldsymbol{u} - \boldsymbol{\theta}\|^2 \right),$$

where $\nabla L(\boldsymbol{\theta})$ is the gradient of L with respect to $\boldsymbol{\theta}$ and $\varepsilon > 0$ corresponds to the step size. If the above minimization with respect to $\boldsymbol{u}$ can be solved efficiently (e.g. analytically), the proximal gradient method is computationally efficient. The above update rule can be equivalently expressed as

$$\boldsymbol{\theta} \longleftarrow \operatorname{prox}_{\varepsilon R}\Big(\boldsymbol{\theta} - \varepsilon \nabla L(\boldsymbol{\theta})\Big),$$

where $\operatorname{prox}_R(\boldsymbol{\theta})$ is called the *proximal operator* defined as

$$\operatorname{prox}_R(\boldsymbol{\theta}) = \operatorname*{argmin}_{\boldsymbol{u}} \left(R(\boldsymbol{u}) + \frac{1}{2}\|\boldsymbol{u} - \boldsymbol{\theta}\|^2 \right).$$

This implies that the proximal gradient method can be regarded as a generalization of the projected gradient method. Indeed, when $R(\boldsymbol{\theta})$ is the *indicator function* for some set $\mathcal{S}$,

$$R(\boldsymbol{\theta}) = \begin{cases} 0 & (\boldsymbol{\theta} \in \mathcal{S}), \\ \infty & (\boldsymbol{\theta} \notin \mathcal{S}), \end{cases}$$

the proximal operator is reduced to an ordinary projection operator and thus the proximal gradient method is a projected gradient method.

FIGURE 34.8

Proximal gradient method.

where $\|\boldsymbol{\Theta}\|_{\text{Frob}}$ denotes the *Frobenius norm*:

$$\|\boldsymbol{\Theta}\|_{\text{Frob}} = \sqrt{\sum_{k_1,k_2=1}^{b_1,b_2} \Theta_{k_1,k_2}^2} = \sqrt{\operatorname{tr}\left(\boldsymbol{\Theta}\boldsymbol{\Theta}^{\top}\right)}.$$

```
n=2; T=6; y=[ones(n/2,T); -ones(n/2,T)];
x=[randn(2,n,T); ones(1,n,T)]; r(1,1,:)=pi*[1:T]/T/10;
c=repmat(cos(r),[1 n/2]); x(1,:,:)=x(1,:,:)+[c -c];
s=repmat(sin(r),[1 n/2]); x(2,:,:)=x(2,:,:)+[s -s];
t0=randn(3,T); e=0.1; l=4;
for o=1:1000
  for k=1:T
    gt(:,k)=x(:,:,k)*(x(:,:,k)'*t0(:,k)-y(:,k));
  end
  [U,S,V]=svd(t0-e*gt,'econ');
  S=diag(max(0,diag(S)-e*l)); t=U*S*V';
  if norm(t-t0)<0.001, break, end
  t0=t;
end

figure(1); clf; colormap([1 0.7 1; 0.7 1 1]);
m=20; X=linspace(-4,4,m); b=repmat(X,[m 1]); bt=b';
for k=1:T
  Y=t(:,k)'*[b(:)'; bt(:)'; ones(1,m^2)];
  subplot(2,3,k); hold on; contourf(X,X,reshape(Y,m,m));
  plot(x(1,y(:,k)==1,k),x(2,y(:,k)==1,k),'bo');
  plot(x(1,y(:,k)==-1,k),x(2,y(:,k)==-1,k),'rx');
  plot(99*sin(r(k))*[1 -1],99*cos(r(k))*[-1 1],'k--');
  axis([-4 4 -4 4]);
end
```

FIGURE 34.9

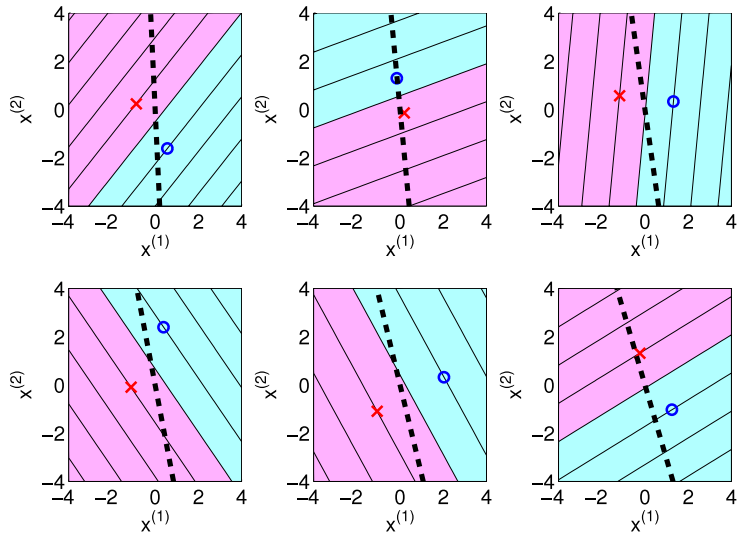
MATLAB code for multitask learning with trace norm regularization.

A notable fact is that the above proximal operator can be analytically expressed as follows [78]:

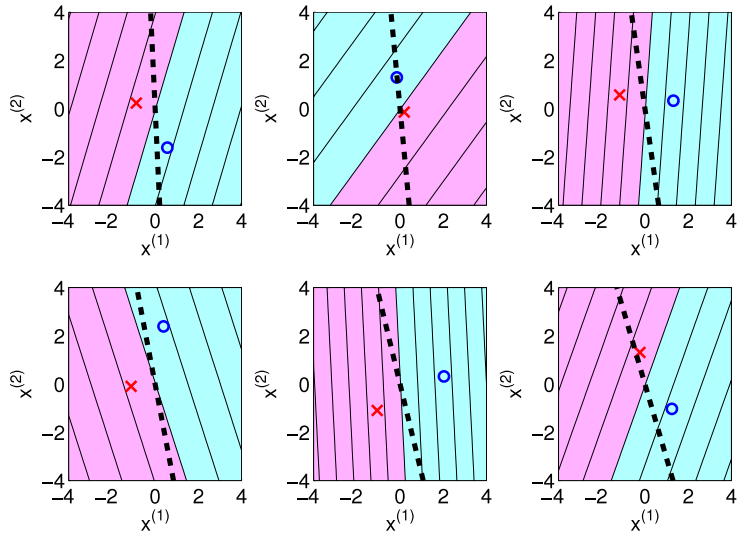
$$\text{prox}(\boldsymbol{\Theta}) = \sum_{k=1}^{\min(b_1,b_2)} \max(0, \sigma_k - \varepsilon\lambda)\boldsymbol{\psi}_k \boldsymbol{\phi}_k^\top,$$

where $\boldsymbol{\psi}_k$, $\boldsymbol{\phi}_k$, and σ_k are a left singular vector, a right singular vector, and a singular value of $\boldsymbol{\Theta}$, respectively:

$$\boldsymbol{\Theta} = \sum_{k=1}^{\min(b_1,b_2)} \sigma_k \boldsymbol{\psi}_k \boldsymbol{\phi}_k^\top.$$



(a) Single-task classification



(b) Multitask classification

FIGURE 34.10

Examples of multitask LS with trace norm regularization. The data set is the same as Fig. 34.2. The dashed lines denote true decision boundaries and the contour lines denote learned results.

This means that singular values less than $\varepsilon\lambda$ are rounded off to zero and other singular values are reduced by $\varepsilon\lambda$. This operation is called *soft thresholding*, which can be computed very efficiently.

A MATLAB code for multitask LS with trace norm regularization is provided in Fig. 34.9, and its behavior using the same data set as in Fig. 34.2 is illustrated in Fig. 34.10. This shows that multitask classification with trace norm regularization works reasonably well.