CHAPTER

SUPPORT VECTOR CLASSIFICATION 27

CHAPTER CONTENTS

In the previous chapter, LS regression was shown to be used also for classification. However, due to the nonmonotonicity of the ℓ_2-loss function, classification by ℓ_2-loss minimization is rather unnatural as a surrogate for the 0/1-loss. In this chapter, a classification algorithm called the *support vector machine* is introduced, which uses a more natural loss function called the *hinge loss*.

27.1 MAXIMUM MARGIN CLASSIFICATION

In this section, a classification algorithm called the *support vector machine* is introduced, which is based on the *margin maximization principle* [115].

27.1.1 HARD MARGIN SUPPORT VECTOR CLASSIFICATION

Let us consider a linear-in-input binary classifier:

$$f_{\boldsymbol{w},\gamma}(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + \gamma, \tag{27.1}$$

where $\boldsymbol{w}$ and γ are the parameters of the model which correspond to the *normal vector* and the *intercept* of the decision boundary, respectively (Fig. 27.1). If $\boldsymbol{w}$ and

An Introduction to Statistical Machine Learning. DOI: 10.1016/B978-0-12-802121-7.00038-8

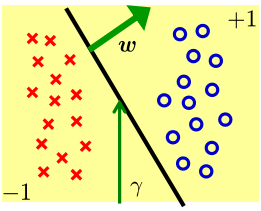
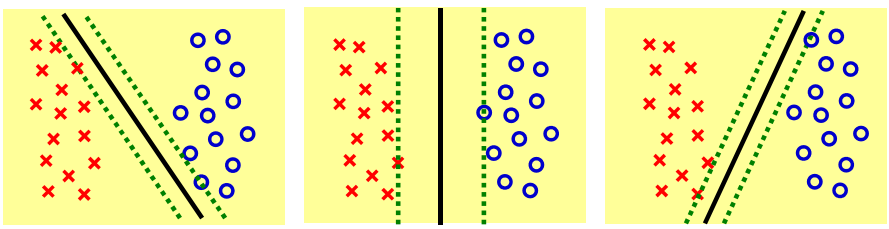


FIGURE 27.1

Linear-in-input binary classifier $f_{\boldsymbol{w},\gamma}(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + \gamma$. $\boldsymbol{w}$ and γ are the normal vector and the intercept of the decision boundary, respectively.



(a) Small margin (b) Large margin (c) Small margin

FIGURE 27.2

Decision boundaries that separate all training samples correctly.

γ are learned so that margins for all training samples are positive, i.e.

$$f_{\boldsymbol{w},\gamma}(\boldsymbol{x}_i) y_i = (\boldsymbol{w}^\top \boldsymbol{x}_i + \gamma) y_i > 0, \quad \forall i = 1, \ldots, n,$$

all training samples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ can be correctly classified. Since the open set $(\boldsymbol{w}^\top \boldsymbol{x}_i + \gamma) y_i > 0$ is not easy to handle mathematically, let us convert this constraint to a closed set based on the fact that the scale of $\boldsymbol{w}$ and γ is arbitrary:

$$(\boldsymbol{w}^\top \boldsymbol{x}_i + \gamma) y_i \geq 1, \quad \forall i = 1, \ldots, n.$$

If there exists $(\boldsymbol{w}, \gamma)$ such that the above condition is fulfilled, the set of training samples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ is said to be *linearly separable*. For a linearly separable set of training samples, there usually exist infinitely many decision boundaries that correctly separate all training samples. Here, let us choose the one that separates all training samples with the *maximum margin*, where the *margin* for a set of training samples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ is defined as the minimum of the normalized margin

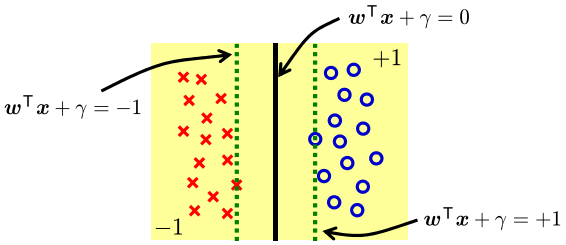


FIGURE 27.3

Decision boundary of hard margin support vector machine. It goes through the center of positive and negative training samples, $\boldsymbol{w}^\top \boldsymbol{x}_+ + \gamma = +1$ for some positive sample $\boldsymbol{x}_+$ and $\boldsymbol{w}^\top \boldsymbol{x}_- + \gamma = -1$ for some negative sample $\boldsymbol{x}_-$.

$m_i = (\boldsymbol{w}^\top \boldsymbol{x}_i + \gamma) y_i / \|\boldsymbol{w}\|$ for $i = 1, \ldots, n$ (Fig. 27.2):

$$\min_{i=1,\ldots,n} \left\{ \frac{(\boldsymbol{w}^\top \boldsymbol{x}_i + \gamma) y_i}{\|\boldsymbol{w}\|} \right\} = \frac{1}{\|\boldsymbol{w}\|}.$$

Geometrically, the margin for a set of training samples corresponds to the half distance between two extreme decision boundaries $\boldsymbol{w}^\top \boldsymbol{x} + \gamma = +1$ and $\boldsymbol{w}^\top \boldsymbol{x} + \gamma = -1$ (Fig. 27.3). The classifier that maximizes the above margin (or equivalently minimizes the squared inverse margin) is called the *hard margin support vector machine* [18]:

$$\min_{\boldsymbol{w},\gamma} \frac{1}{2} \|\boldsymbol{w}\|^2 \quad \text{subject to } (\boldsymbol{w}^\top \boldsymbol{x}_i + \gamma) y_i \geq 1, \quad \forall i = 1, \ldots, n.$$

27.1.2 SOFT MARGIN SUPPORT VECTOR CLASSIFICATION

The hard margin support vector machine requires linear separability, which may not always be satisfied in practice. The *soft margin support vector machine* [30] relaxes this requirement by allowing error $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_n)^\top$ for margins (Fig. 27.4):

$$\begin{aligned} &\min_{\boldsymbol{w},\gamma,\boldsymbol{\xi}} \left[\frac{1}{2} \|\boldsymbol{w}\|^2 + C \sum_{i=1}^{n} \xi_i \right] \\ &\text{subject to } (\boldsymbol{w}^\top \boldsymbol{x}_i + \gamma) y_i \geq 1 - \xi_i, \; \xi_i \geq 0, \quad \forall i = 1, \ldots, n, \end{aligned} \tag{27.2}$$

where $C > 0$ is a tuning parameter that controls the margin errors. The margin error $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_n)^\top$ is also referred to as *slack variables* in optimization. Larger C

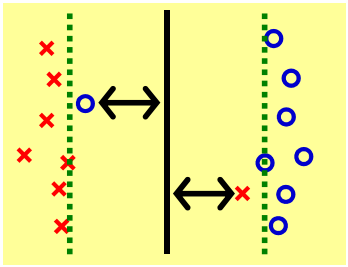


FIGURE 27.4

Soft margin support vector machine allows small margin errors.

makes the margin error $\sum_{i=1}^{n} \xi_i$ small and then soft margin support vector machine approaches hard margin support vector machine.

Below, the soft margin support vector machine may be merely called the support vector machine.

27.2 DUAL OPTIMIZATION OF SUPPORT VECTOR CLASSIFICATION

The optimization problem of support vector classification (27.2) takes the form of *quadratic programming* (Fig. 27.5), where the objective is a quadratic function and constraints are linear. Since quadratic programming has been extensively studied in the optimization community and various practical algorithms are available, which can be readily used for obtaining the solution of support vector classification.

However, although original linear-in-input model (27.1) contains only $d + 1$ parameters (i.e. $\boldsymbol{w} \in \mathbb{R}^d$ and $\gamma \in \mathbb{R}$), quadratic optimization problem (27.2) contains additional parameter $\boldsymbol{\xi} \in \mathbb{R}^n$. Thus, the total number of parameters to be optimized is $n + d + 1$, which does not scale well to large data sets.

The *Lagrange function* (Fig. 23.5) of optimization problem (27.2) is given by

$$L(\boldsymbol{w}, \gamma, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i \left((\boldsymbol{w}^\top \boldsymbol{x}_i + \gamma) y_i - 1 + \xi_i\right) - \sum_{i=1}^{n} \beta_i \xi_i.$$

Then Lagrange dual optimization problem is given by

$$\max_{\boldsymbol{\alpha}, \boldsymbol{\beta}} \inf_{\boldsymbol{w}, \gamma, \boldsymbol{\xi}} L(\boldsymbol{w}, \gamma, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \quad \text{subject to} \quad \boldsymbol{\alpha} \geq \boldsymbol{0}, \quad \boldsymbol{\beta} \geq \boldsymbol{0}.$$

Quadratic programming is the optimization problem defined for matrices $\boldsymbol{F}$ and $\boldsymbol{G}$ and vectors $\boldsymbol{f}$ and $\boldsymbol{g}$ as

$$\min_{\boldsymbol{\theta}} \left[\frac{1}{2} \boldsymbol{\theta}^\top \boldsymbol{F} \boldsymbol{\theta} + \boldsymbol{f}^\top \boldsymbol{\theta} \right] \quad \text{subject to } \boldsymbol{G}\boldsymbol{\theta} \leq \boldsymbol{g},$$

where vector inequality $\boldsymbol{G}\boldsymbol{\theta} \leq \boldsymbol{g}$ denotes the elementwise inequalities:

$$\begin{pmatrix} a \\ b \end{pmatrix} \leq \begin{pmatrix} c \\ d \end{pmatrix} \Longleftrightarrow \begin{cases} a \leq c, \\ b \leq d. \end{cases}$$

Matrix $\boldsymbol{F}$ is supposed to be positive, i.e. all eigenvalues (see Fig. 6.2) are positive. When $\boldsymbol{F}$ is *ill-conditioned* in the sense that a very small eigenvalue exists, a small positive constant may be added to the diagonal elements of $\boldsymbol{F}$ to improve numerical stability.

FIGURE 27.5

Quadratic programming.

The first-order optimality condition of $\inf_{\boldsymbol{w},\gamma,\boldsymbol{\xi}} L(\boldsymbol{w},\gamma,\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\beta})$ yields

$$\begin{aligned} \frac{\partial L}{\partial \boldsymbol{w}} = 0 \quad &\Longrightarrow \quad \boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i, \\ \frac{\partial L}{\partial \gamma} = 0 \quad &\Longrightarrow \quad \sum_{i=1}^{n} \alpha_i y_i = 0, \\ \frac{\partial L}{\partial \xi_i} = 0 \quad &\Longrightarrow \quad \alpha_i + \beta_i = C, \quad \forall i = 1, \ldots, n, \end{aligned}$$

showing that $\boldsymbol{w}$ and $\boldsymbol{\beta}$ can be expressed by $\boldsymbol{\alpha}$. Furthermore, the condition $\alpha_i + \beta_i = C$ allows us to eliminate the margin error $\boldsymbol{\xi}$ from the Lagrange function.

Summarizing the above computations, the Lagrange dual optimization problem is simplified as

$$\begin{aligned} \widehat{\boldsymbol{\alpha}} = \operatorname*{argmax}_{\boldsymbol{\alpha}} & \left[\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j \right] \\ & \text{subject to } \sum_{i=1}^{n} \alpha_i y_i = 0,\ 0 \leq \alpha_i \leq C \quad \text{for } i = 1, \ldots, n. \end{aligned} \tag{27.3}$$

This is a quadratic programming problem which only contains n variables to be optimized, and therefore it may be solved more efficiently than original optimization problem (27.2). The solution $\widehat{\boldsymbol{\alpha}}$ of Lagrange dual optimization problem (27.3) gives the solution of support vector classification as

$$\widehat{\boldsymbol{w}} = \sum_{i=1}^{n} \widehat{\alpha}_i y_i \boldsymbol{x}_i.$$

The solution of intercept $\widehat{\gamma}$ can be obtained by using $\boldsymbol{x}_i$ such that $0 < \widehat{\alpha}_i < C$ as

$$\widehat{\gamma} = y_i - \sum_{j:\widehat{\alpha}_i > 0} \widehat{\alpha}_j y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j. \tag{27.4}$$

The derivation of $\widehat{\gamma}$ above will be explained in Section 27.3. When the linear-in-input model without the intercept,

$$f_{\boldsymbol{w},\gamma}(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x},$$

is used, the constraint,

$$\sum_{i=1}^{n} \alpha_i y_i = 0,$$

can be removed from Lagrange dual optimization problem (27.3).

While LS classification could not perfectly classify the training samples plotted in Fig. 26.5, support vector classification can achieve perfect separation (Fig. 27.6). Among 200 dual parameters $\{\alpha_i\}_{i=1}^n$, 197 parameters take zero and only 3 parameters specified by the square in the plot take nonzero values. The reason for this sparsity induction will be explained in the next section.

The quadratic programming approach to obtaining the solution of support vector classification explained above is handy and useful to understand the properties of support vector classification. However, its scalability to large-scale data is limited and more advanced optimization techniques have been extensively studied, e.g. [25].

27.3 SPARSENESS OF DUAL SOLUTION

The ℓ_1-constraint was shown to induce a sparse solution in Chapter 24. On the other hand, as shown in Fig. 27.6, the dual solution $\widehat{\boldsymbol{\alpha}}$ of support vector classification tends to be sparse even though the ℓ_1-constraint is not used.

To explain the sparsity induction mechanism of support vector classification, let us investigate the optimality conditions called the KKT conditions (Fig. 27.7). Dual variables and constraints satisfy the following conditions called *complementary slackness*:

$$\alpha_i(m_i - 1 + \xi_i) = 0 \quad \text{and} \quad \beta_i \xi_i = 0, \ \ \forall i = 1, \ldots, n,$$

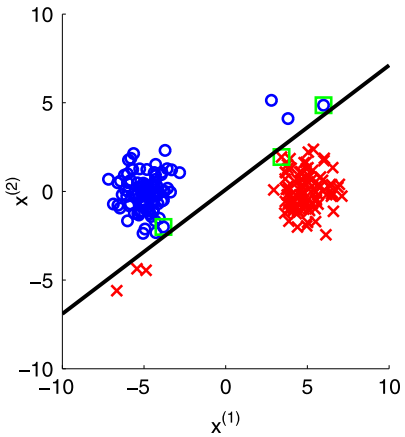


FIGURE 27.6

Example of linear support vector classification. Among 200 dual parameters $\{\alpha_i\}_{i=1}^n$, 197 parameters take zero and only 3 parameters specified by the square in the plot take nonzero values.

where $m_i = (\boldsymbol{w}^\top \boldsymbol{x}_i + \gamma) y_i$ denotes the margin for the ith training sample $(\boldsymbol{x}_i, y_i)$. Furthermore, $\frac{\partial L}{\partial \xi_i} = 0$ yields

$$\alpha_i + \beta_i = C.$$

Summarizing the above conditions, the dual variable α_i and margin m_i satisfy the following relations (Fig. 27.8):

- $\alpha_i = 0$ implies $m_i \geq 1$: the ith training sample $\boldsymbol{x}_i$ is on the margin border or inside the margin and is correctly classified.
- $0 < \alpha_i < C$ implies $m_i = 1$: $\boldsymbol{x}_i$ is on the margin border and correctly classified.
- $\alpha_i = C$ implies $m_i \leq 1$: $\boldsymbol{x}_i$ is on the margin border or outside the margin. If $\xi_i > 1$, $m_i < 0$ and then $\boldsymbol{x}_i$ is misclassified.
- $m_i > 1$ implies $\alpha_i = 0$: if $\boldsymbol{x}_i$ is inside the margin, $\alpha_i = 0$.
- $m_i < 1$ implies $\alpha_i = C$: if $\boldsymbol{x}_i$ is outside the margin, $\alpha_i = C$.

A training input sample $\boldsymbol{x}_i$ such that $\alpha_i > 0$ is called a *support vector*, because those points "support" the decision boundary. In other words, nonsupport vectors do not affect the classification boundary. Support vector $\boldsymbol{x}_i$ such that $0 < \alpha_i < C$ is on the margin border (see Fig. 27.8 again) and satisfy

$$m_i = (\boldsymbol{w}^\top \boldsymbol{x}_i + \gamma) y_i = 1.$$

The solution of 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, satisfies the following *Karush-Kuhn-Tucker* (KKT) conditions:

$$\frac{\partial L}{\partial \boldsymbol{t}} = \boldsymbol{0}, \quad \boldsymbol{g}(\boldsymbol{t}) \leq \boldsymbol{0}, \quad \boldsymbol{\lambda} \geq \boldsymbol{0},$$
$$\text{and } \lambda_i g_i(\boldsymbol{t}) = 0, \ \forall i = 1, \ldots, n,$$

where $L(\boldsymbol{t}, \boldsymbol{\lambda}) = f(\boldsymbol{t}) + \boldsymbol{\lambda}^\top \boldsymbol{g}(\boldsymbol{t})$ is the the Lagrange function (Fig. 23.5) and $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_p)^\top$ are the Lagrange multipliers. The last condition $\lambda_i g_i(\boldsymbol{t}) = 0$ is called *complementary slackness* because at least either of λ_i or $g_i(\boldsymbol{t})$ is zero.

FIGURE 27.7

KKT optimality conditions.

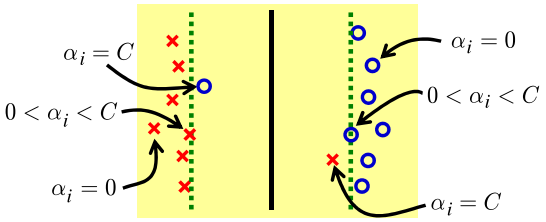


FIGURE 27.8

When $\alpha_i = 0$, $\boldsymbol{x}_i$ is inside the margin and correctly classified. When $0 < \alpha_i < C$, $\boldsymbol{x}_i$ is on the margin border (the dotted lines) and correctly classified. When $\alpha_i = C$, $\boldsymbol{x}_i$ is outside the margin, and if $\xi_i > 1$, $m_i < 0$ and thus $\boldsymbol{x}_i$ is misclassified.

This implies that the intercept γ satisfies

$$\gamma = 1/y_i - \boldsymbol{w}^\top \boldsymbol{x}_i = y_i - \boldsymbol{w}^\top \boldsymbol{x}_i, \ \forall i : 0 < \alpha_i < C,$$

from which the solution $\widehat{\gamma}$ in Eq. (27.4) is obtained.

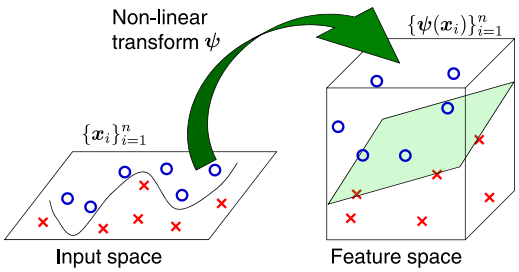


FIGURE 27.9

Nonlinearization of support vector machine by kernel trick.

27.4 NONLINEARIZATION BY KERNEL TRICK

In this section, support vector classification introduced for linear-in-input model (27.1) is extended to nonlinear models. More specifically, training input samples $\{\boldsymbol{x}_i\}_{i=1}^n$ are transformed to a feature space using a nonlinear function $\boldsymbol{\psi}$ (Fig. 27.9). Then, for the transformed samples $\{\boldsymbol{\psi}(\boldsymbol{x}_i)\}_{i=1}^n$, the linear support vector machine is trained. A linear decision boundary obtained in the feature space is a nonlinear decision boundary obtained in the original input space.

If the dimensionality of the feature space is higher than the original input space, it is more likely that training samples are linearly separable. However, too high dimensionality causes large computation costs.

To cope with this problem, a technique called the *kernel trick* [30, 89] is useful. In the optimization problem of the linear support vector machine, Eq. (27.3), training input samples $\{\boldsymbol{x}_i\}_{i=1}^n$ appear only in terms of their inner product:

$$\boldsymbol{x}_i^\top \boldsymbol{x}_j = \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle.$$

Similarly, in the above nonlinear support vector machine, training input samples $\{\boldsymbol{x}_i\}_{i=1}^n$ appear only in

$$\langle \boldsymbol{\psi}(\boldsymbol{x}_i), \boldsymbol{\psi}(\boldsymbol{x}_j) \rangle.$$

This means that the nonlinear support vector machine is trainable even if $\boldsymbol{\psi}(\boldsymbol{x}_i)$ and $\boldsymbol{\psi}(\boldsymbol{x}_j)$ are unknown, as long as their inner product $\langle \boldsymbol{\psi}(\boldsymbol{x}_i), \boldsymbol{\psi}(\boldsymbol{x}_j) \rangle$ is known.

The kernel trick is to directly specify the inner product by a kernel function $K(\cdot, \cdot)$ as

$$\langle \boldsymbol{\psi}(\boldsymbol{x}), \boldsymbol{\psi}(\boldsymbol{x}') \rangle = K(\boldsymbol{x}, \boldsymbol{x}').$$

This shows that, as long as the kernel function can be computed with a computation cost independent of the dimensionality of the feature space, the computation cost

of training the support vector machine is independent of the dimensionality of the feature space. As a kernel function, the *polynomial kernel*,

$$K(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^\top \boldsymbol{x}' + a)^p,$$

and the *Gaussian kernel*,

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

are popular choices, where a is a positive real scalar, p is a positive integer, and h is a positive real scalar. It is known that the feature vector $\boldsymbol{\psi}$ that corresponds to the Gaussian kernel is actually *infinite dimensional*. Therefore, without the kernel trick, the solution cannot be computed explicitly.

A MATLAB code of support vector classification for the Gaussian kernel is provided in Fig. 27.10, and its behavior is illustrated in Fig. 27.11. This shows that nonlinearly entangled data can be successfully classified by support vector classification with the Gaussian kernel.

Since the kernel support vector machine only uses the kernel value $K(\boldsymbol{x}, \boldsymbol{x}')$, as long as the kernel function is defined, nonvectorial pattern $\boldsymbol{x}$ such as sequences, trees, and graphs can be handled [46]. Note that the kernel trick can be applied not only to support vector machines but also to any algorithms that handle training samples only in terms of their inner products, for example, *clustering* (Chapter 37) and *dimensionality reduction* (Section 36.1).

27.5 MULTICLASS EXTENSION

As explained in Section 26.3, a multiclass classification problem can be reduced to a set of binary classification problems. In this section, a direct formulation of *multiclass support vector classification* [34] is provided.

Let us classify a test sample $\boldsymbol{x}$ into class $\widehat{y}$ as

$$\widehat{y} = \operatorname*{argmax}_{y=1,\ldots,c} \boldsymbol{w}_y^\top \boldsymbol{x},$$

where $\boldsymbol{w}_y$ denotes the parameter vector for class y. In this situation, it is desirable that $\boldsymbol{w}_{\widehat{y}}^\top \boldsymbol{x}$ is larger than $\boldsymbol{w}_y^\top \boldsymbol{x}$ for $y \neq \widehat{y}$ with a *large margin*. Based on this idea, *hard margin* multiclass support vector classification for separable data is formulated as

$$\min_{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_c} \frac{1}{2} \sum_{y=1}^{c} \|\boldsymbol{w}_y\|^2$$
$$\text{subject to } \boldsymbol{w}_{y_i}^\top \boldsymbol{x}_i - \boldsymbol{w}_y^\top \boldsymbol{x}_i \geq 1, \quad \forall i = 1,\ldots,n, \forall y \neq y_i.$$

Similarly to the binary case, the above formulation can be extended to *soft margin* multiclass support vector classification using the margin error $\boldsymbol{\xi} = (\xi_1,\ldots,\xi_n)^\top$ as

```
n=200; a=linspace(0,4*pi,n/2);
u=[a.*cos(a) (a+pi).*cos(a)]'+rand(n,1);
v=[a.*sin(a) (a+pi).*sin(a)]'+rand(n,1);
x=[u v]; y=[ones(1,n/2) -ones(1,n/2)]';

x2=sum(x.^2,2); hh=2*1^2; l=0.01;
k=exp(-(repmat(x2,1,n)+repmat(x2',n,1)-2*x*x')/hh);
Q=(y*y').*k+0.01*eye(n); q=-ones(n,1);
H=[-eye(n); eye(n)]; h=[zeros(n,1); ones(n,1)/(2*l)];
a=quadprog(Q,q,H,h); t=y.*a;

m=100; X=linspace(-15,15,m)'; X2=X.^2;
U=exp(-(repmat(u.^2,1,m)+repmat(X2',n,1)-2*u*X')/hh);
V=exp(-(repmat(v.^2,1,m)+repmat(X2',n,1)-2*v*X')/hh);
figure(1); clf; hold on; axis([-15 15 -15 15]);
contourf(X,X,sign(V'*(U.*repmat(t,1,m))));
plot(x(y==1,1),x(y==1,2),'bo');
plot(x(y==-1,1),x(y==-1,2),'rx');
colormap([1 0.7 1; 0.7 1 1]);
```

FIGURE 27.10

MATLAB code of support vector classification for Gaussian kernel. `quadprog.m` included in Optimization Toolbox is required. Free alternatives to `quadprog.m` are available, e.g. from http://www.mathworks.com/matlabcentral/fileexchange/.

$$\min_{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_c,\boldsymbol{\xi}} \left[\frac{1}{2} \sum_{y=1}^{c} \|\boldsymbol{w}_y\|^2 + C \sum_{i=1}^{n} \xi_i \right]$$
$$\text{subject to } \boldsymbol{w}_{y_i}^\top \boldsymbol{x}_i - \boldsymbol{w}_y^\top \boldsymbol{x}_i \geq t_{i,y} - \xi_i, \quad \forall i = 1,\ldots,n, \; \forall y = 1,\ldots,c,$$

where

$$t_{i,y} = \begin{cases} 0 & (y = y_i), \\ 1 & (y \neq y_i). \end{cases}$$

Note that the constraint $\boldsymbol{w}_{y_i}^\top \boldsymbol{x}_i - \boldsymbol{w}_y^\top \boldsymbol{x}_i \geq t_{i,y} - \xi_i$ is reduced to $\xi_i \geq 0$ when $y = y_i$.

The *Lagrange dual* (Fig. 23.5) of the above optimization problem is given by

$$\max_{\alpha_{1,1},\ldots,\alpha_{n,c}} \left[\sum_{i=1}^{n} \alpha_{i,y_i} - \frac{1}{2} \sum_{i,j=1}^{n} \sum_{y=1}^{c} \alpha_{i,y} \alpha_{j,y} \boldsymbol{x}_i^\top \boldsymbol{x}_j \right]$$

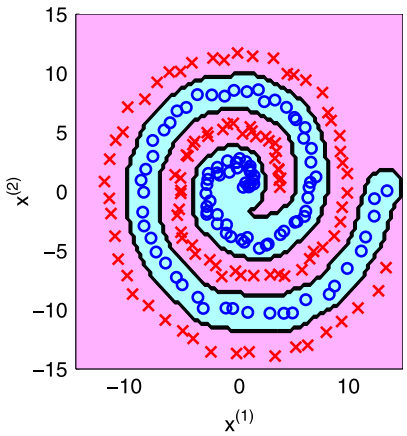


FIGURE 27.11

Example of support vector classification for Gaussian kernel.

$$\text{subject to } \sum_{y=1}^{c} \alpha_{i,y} = 0, \quad \forall i = 1,\ldots,n,$$
$$\alpha_{i,y} \leq C(1 - t_{i,y}), \quad \forall i = 1,\ldots,n, \ \forall y = 1,\ldots,c.$$

This is a quadratic programming problem and thus the solution $\widehat{\alpha}_{i,y}$ can be obtained by standard optimization software.

From the dual solution $\widehat{\alpha}_{i,y}$, the primal solution $\widehat{\boldsymbol{w}}_y$ can be obtained as

$$\widehat{\boldsymbol{w}}_y = \sum_{i=1}^{n} \widehat{\alpha}_{i,y} \boldsymbol{x}_i.$$

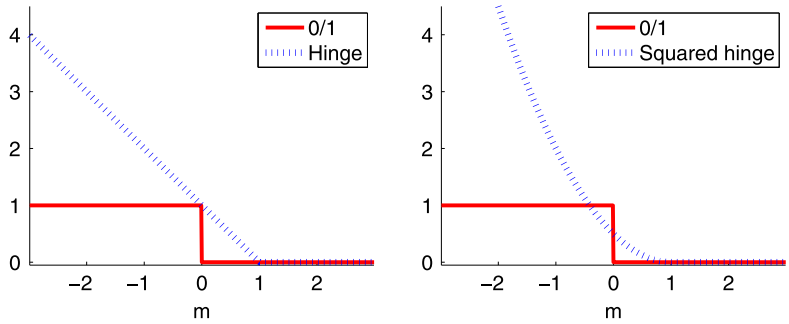
Then a test sample $\boldsymbol{x}$ is classified into class $\widehat{y}$ as

$$\widehat{y} = \operatorname*{argmax}_{y=1,\ldots,c} \sum_{i=1}^{n} \widehat{\alpha}_{i,y} \boldsymbol{x}_i^\top \boldsymbol{x}.$$

Through this dual formulation, the *kernel trick* introduced in Section 27.4 can be applied to nonlinearize multiclass support vector classification by replacing $\boldsymbol{x}^\top \boldsymbol{x}'$ with $K(\boldsymbol{x}, \boldsymbol{x}')$.

27.6 LOSS MINIMIZATION VIEW

So far, support vector classification was derived based on margin maximization and the kernel trick. Although such a derivation is quite different from LS classification



(a) Hinge loss (b) Squared hinge loss

FIGURE 27.12

Hinge loss and squared hinge loss.

introduced in Chapter 26, support vector classification can actually be regarded as an extension of LS classification. In this section, support vector classification is interpreted as loss minimization. Another view of support vector classification in L_1-distance approximation is given in Section 39.1.4.

27.6.1 HINGE LOSS MINIMIZATION

Let us consider the *hinge loss* as a surrogate for the 0/1-loss (Fig. 27.12(a)):

$$\max\{0, 1-m\} = \begin{cases} 1-m & (m \leq 1), \\ 0 & (m > 1), \end{cases}$$

where m denotes the margin $m = f_{\boldsymbol{\theta}}(\boldsymbol{x})y$. The hinge loss is zero for $m \geq 1$, which is the same as the 0/1-loss. On the other hand, for $m < 1$, the hinge loss takes a positive value $1-m$ and its slope is negative.

For the kernel model with intercept γ,

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

let us consider hinge loss minimization with generalized ℓ_2-regularization:

$$\min_{\boldsymbol{\theta},\gamma} \left[\sum_{i=1}^{n} \max\{0, 1 - f_{\boldsymbol{\theta},\gamma}(\boldsymbol{x}_i)y_i\} + \frac{\lambda}{2}\boldsymbol{\theta}^\top \boldsymbol{K}\boldsymbol{\theta} \right],$$

where $K_{i,j} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$. As illustrated in Fig. 27.13, the hinge loss corresponds to the maximum of $1-m$ and 0:

$$\max\{0, 1-m\} = \min_{\xi} \xi \quad \text{subject to } \xi \geq 1-m,\ \xi \geq 0.$$

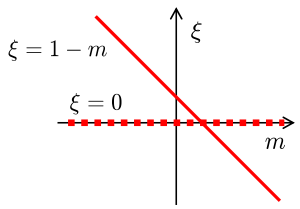


FIGURE 27.13

Hinge loss as maximizer of $1 - m$ and 0.

Then the optimization problem of regularized hinge loss minimization is expressed as

$$\min_{\boldsymbol{\theta},\gamma,\boldsymbol{\xi}} \left[\sum_{i=1}^{n} \xi_i + \frac{\lambda}{2} \boldsymbol{\theta}^\top \boldsymbol{K} \boldsymbol{\theta} \right]$$
$$\text{subject to } \xi_i \geq 1 - f_{\boldsymbol{\theta},\gamma}(\boldsymbol{x}_i) y_i,\ \xi_i \geq 0, \quad \forall i = 1, \ldots, n. \tag{27.5}$$

Let us recall the primal optimization problem of kernel support vector classification (see (27.2)):

$$\min_{\boldsymbol{w},\gamma,\boldsymbol{\xi}} \left[\frac{1}{2} \|\boldsymbol{w}\|^2 + C \sum_{i=1}^{n} \xi_i \right]$$
$$\text{subject to } \xi_i \geq 1 - (\boldsymbol{w}^\top \boldsymbol{\psi}(\boldsymbol{x}_i) + \gamma) y_i,\ \xi_i \geq 0, \quad \forall i = 1, \ldots, n.$$

Letting $C = 1/\lambda$ and $\boldsymbol{w} = \sum_{j=1}^{n} \theta_j \boldsymbol{\psi}(\boldsymbol{x}_j)$ and using the kernel trick $\boldsymbol{\psi}(\boldsymbol{x}_i)^\top \boldsymbol{\psi}(\boldsymbol{x}_j) = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ in the above optimization problem show that support vector classification is actually equivalent to ℓ_2-regularized hinge loss minimization given by Eq. (27.5).

This loss minimization interpretation allows us to consider various extensions of support vector classification, which are described below.

27.6.2 SQUARED HINGE LOSS MINIMIZATION

For margin $m = f_{\boldsymbol{\theta}}(\boldsymbol{x}) y$, let us consider the *squared hinge loss*:

$$\frac{1}{2} \left(\max \left\{ 0, 1 - m \right\} \right)^2 = \begin{cases} \frac{1}{2}(1 - m)^2 & (m \leq 1), \\ 0 & (m > 1), \end{cases}$$

which is differentiable everywhere (Fig. 27.12(b)). Then ℓ_2-regularized squared hinge loss minimization,

$$\min_{\boldsymbol{\theta}} \left[\frac{1}{2} \sum_{i=1}^{n} \left(\max \left\{ 0, 1 - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) y_i \right\} \right)^2 + \frac{\lambda}{2} \|\boldsymbol{\theta}\|^2 \right],$$

1. Initialize parameter $\boldsymbol{\theta}$, e.g. randomly or by ℓ_2-regularized LS as

$$\boldsymbol{\theta} \longleftarrow (\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1} \boldsymbol{\Phi}^\top \boldsymbol{y}.$$

2. Compute the retarget matrix $\boldsymbol{V}$ from the current solution $\boldsymbol{\theta}$ as

$$\boldsymbol{V} = \mathrm{diag}\,(v_1, \ldots, v_n),$$

where

$$v_i = \boldsymbol{\theta}^\top \boldsymbol{\phi}(\boldsymbol{x}_i) y_i + \max(0, 1 - \boldsymbol{\theta}^\top \boldsymbol{\phi}(\boldsymbol{x}_i) y_i).$$

3. Compute the solution $\boldsymbol{\theta}$ based on the retarget matrix $\boldsymbol{V}$:

$$\boldsymbol{\theta} \longleftarrow (\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1} \boldsymbol{\Phi}^\top \boldsymbol{V} \boldsymbol{y}.$$

4. Iterate 2–3 until convergence.

FIGURE 27.14

Iterative retargeted LS for ℓ_2-regularized squared hinge loss minimization.

may be solved by a simple stochastic gradient algorithm introduced in Section 15.3.

Another approach to obtaining the solution of ℓ_2-regularized squared hinge loss minimization is to put

$$v = m + \max\{0, 1 - m\}$$

and express the squared hinge loss as

$$\begin{aligned}\Big(\max\{0, 1 - m\}\Big)^2 &= (v - m)^2 \\ &= \Big(v - f_{\boldsymbol{\theta}}(\boldsymbol{x}) y\Big)^2 = \Big(vy - f_{\boldsymbol{\theta}}(\boldsymbol{x})\Big)^2,\end{aligned}$$

where $y = \pm 1$ is used. Since v is unknown, let us consider an iterative procedure and replace it with an estimate $\widehat{v}$ computed from the current solution. Then the solution of ℓ_2-regularized squared hinge loss minimization can be obtained by *iterative retargeted LS* (see Fig. 27.14).

A MATLAB code of iterative retargeted LS for ℓ_2-regularized squared hinge loss minimization is provided in Fig. 27.15, and its behavior is illustrated in Fig. 27.16. This shows that nonlinearly entangled data can be successfully classified almost in the same way as ordinary support vector classification with the Gaussian kernel (see Fig. 27.11).

```
n=200; a=linspace(0,4*pi,n/2);
u=[a.*cos(a) (a+pi).*cos(a)]'+rand(n,1);
v=[a.*sin(a) (a+pi).*sin(a)]'+rand(n,1);
x=[u v]; y=[ones(1,n/2) -ones(1,n/2)]';

x2=sum(x.^2,2); hh=2*1^2; l=1;
k=exp(-(repmat(x2,1,n)+repmat(x2',n,1)-2*x*x')/hh);
A=inv(k'*k+l*eye(n))*k'; t=rand(n,1);
for o=1:1000
  z=(k*t).*y; w=z+max(0,1-z); t0=A*(w.*y);
  if norm(t-t0)<0.001, break, end
  t=t0;
end

m=100; X=linspace(-15,15,m)'; X2=X.^2;
U=exp(-(repmat(u.^2,1,m)+repmat(X2',n,1)-2*u*X')/hh);
V=exp(-(repmat(v.^2,1,m)+repmat(X2',n,1)-2*v*X')/hh);
figure(1); clf; hold on; axis([-15 15 -15 15]);
contourf(X,X,sign(V'*(U.*repmat(t,1,m))));
plot(x(y==1,1),x(y==1,2),'bo');
plot(x(y==-1,1),x(y==-1,2),'rx');
colormap([1 0.7 1; 0.7 1 1]);
```

FIGURE 27.15

MATLAB code of iterative retargeted LS for ℓ_2-regularized squared hinge loss minimization.

27.6.3 RAMP LOSS MINIMIZATION

As illustrated in Fig. 27.12, the (squared) hinge loss is not upper-bounded. For this reason, learning with such an unbounded loss tends to be strongly affected by an *outlier* (Fig. 27.17(a)).

As discussed in Chapter 25, using a bounded loss can enhance the robustness against outliers. For margin $m = f_{\boldsymbol{\theta}}(\boldsymbol{x})y$, let us consider the *ramp loss* (Fig. 27.18(a)):

$$\min\left\{1, \max\left(0, 1-m\right)\right\} = \begin{cases} 1 & (m < 0), \\ 1-m & (0 \leq m \leq 1), \\ 0 & (m > 1). \end{cases}$$

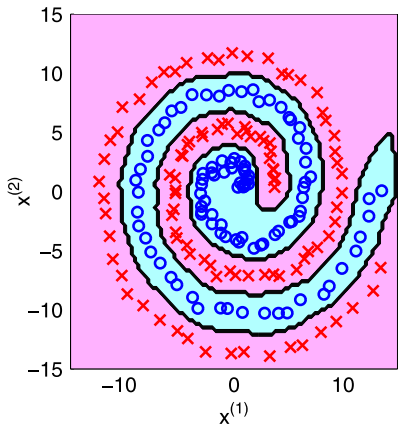
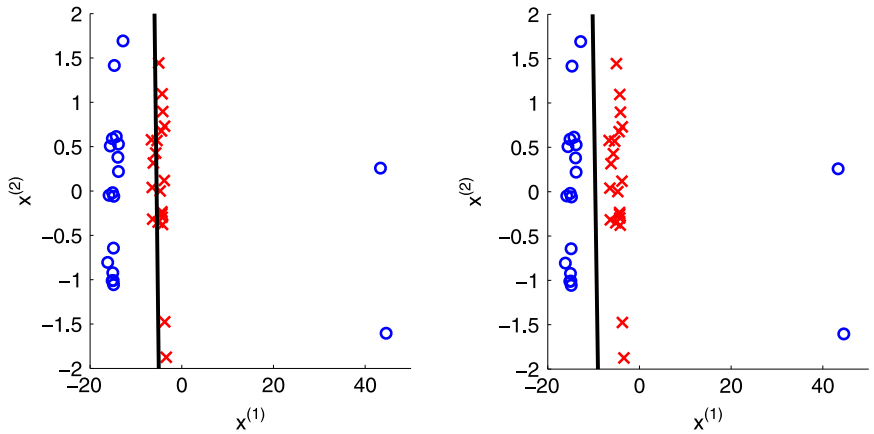


FIGURE 27.16

Example of ℓ_2-regularized squared hinge loss minimization.



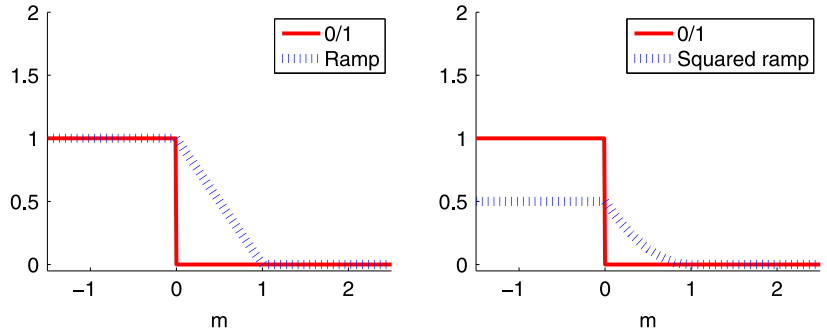
(a) Ordinary support vector classification (b) Robust support vector classification

FIGURE 27.17

Examples of support vector classification with outliers.

Due to the boundedness of the ramp loss, ramp loss minimization,

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \min\left\{1, \max\left(0, 1 - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) y_i\right)\right\},$$



(a) Ramp loss (b) Squared ramp loss

FIGURE 27.18

Ramp loss and squared ramp loss.

is more robust against outliers than hinge loss minimization, as illustrated Fig. 27.17(b). However, since the ramp loss is a nonconvex function, obtaining the global optimal solution is not straightforward. A ramp loss variant of the support vector machine is also referred to as the *robust support vector machine*.

A variation of the ramp loss is the *squared ramp loss* (Fig. 27.18(b)):

$$\frac{1}{2}\left(\min\left\{1,\max\left(0,1-m\right)\right\}\right)^2 = \begin{cases} \frac{1}{2} & (m < 0), \\ \frac{1}{2}(1-m)^2 & (0 \leq m \leq 1), \\ 0 & (m > 1). \end{cases}$$

A local optimal solution of squared ramp loss minimization,

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \left(\min\left\{1,\max\left(0,1-f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)y_i\right)\right\}\right)^2,$$

can be obtained by *iteratively retargeted LS* in the same as squared hinge loss minimization (see Section 27.6.2). The only difference is that the definition of v in iteratively retargeted LS is replaced with

$$v = m + \min\{1,\max(0,1-m)\}.$$