60 PERCEPTRON

In this and the next chapter we discuss two binary classification schemes known as Perceptron and support vector machines. In contrast to logistic regression, these methods do not approximate the conditional pdf, $f_{\boldsymbol{\gamma}|h}(\gamma|h)$, nor the joint pdf, $f_{\boldsymbol{\gamma},\boldsymbol{h}}(\gamma,h)$. Instead, both schemes are examples of *deterministic methods* that operate directly on data realizations $\{\gamma(n),h_n\}$ and learn from the training data how to discriminate between classes. As the derivations will show, these methods rely on geometric arguments to construct hyperplanes that separate the data into classes. The Perceptron algorithm, discussed in this chapter, is one of the earliest iterative solutions devised for binary classification problems. Its development led to a flurry of interest in learning methods culminating with various techniques for cascading elementary units into the form of neural networks for more sophisticated solutions. We will motivate Perceptron from first principles for linearly separable data and comment on its convergence properties and limitations. In comparison to logistic regression, which continually updates its weight iterate w_n in response to data, the Perceptron algorithm limits its updates only to data points that are misclassified. This results in a simpler implementation albeit at a cost. For instance, we will find that Perceptron is not able to complement its classification decision with a confidence level, as was the case with logistic regression.

60.1 LINEAR SEPARABILITY

Assume we are given N independent realizations $\{\gamma(n),h_n\}$, where $\gamma(n)\in\{\pm1\}$ is the binary label corresponding to the $n-$th feature vector $h_n\in\mathbb{R}^M$. The objective is to construct a classifier $c(h):\mathbb{R}^M\rightarrow\{\pm1\}$ that maps features vectors h into their labels. One popular classification structure is the set of "affine-based" classifiers defined by

$$c(h)\;\triangleq\;\text{sign}\left(h^{\mathsf{T}}w-\theta\right) \tag{60.1}$$

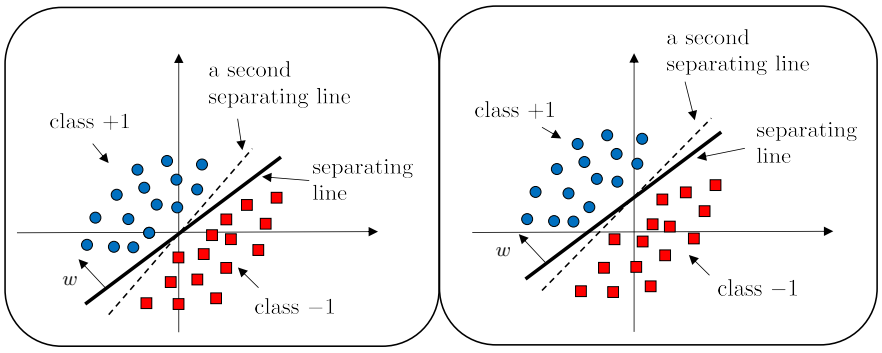
where each classifier is parameterized by a vector $w\in\mathbb{R}^M$ and an offset parameter $\theta\in\mathbb{R}$. The qualification "affine" or, more simply "linear", refers to the relation $h^{\mathsf{T}}w-\theta$ that appears inside the sign operation. Any point h that lies on the hyperplane defined by (w,θ) satisfies $h^{\mathsf{T}}w-\theta=0$. On the other hand,

points lying on one side of the hyperplane satisfy $h^{\mathsf{T}}w - \theta > 0$ while points lying on the other side satisfy $h^{\mathsf{T}}w - \theta < 0$ — see Fig. 60.1. The sign operation therefore allows us to identify where a given h lies in relation to the hyperplane $h^{\mathsf{T}}w - \theta = 0$. This class of "linear" classifiers is very useful in practice even in situations when the data cannot be well-separated by "linear" structures. This is because they will serve as building blocks for more elaborate classifiers. We will illustrate this situation later in Example 63.1 and also in future Sec. 63.2 when we discuss kernel methods.

We will say that a given dataset $\{\gamma(n), h_n\}$ is *linearly separable* when linear classifiers of the form (60.1) exist that are able to separate the data into its two classes with one class lying on one side of the hyperplane and the other class lying on the other side of the hyperplane. This situation is illustrated in Fig. 60.1 for the case $M = 2$. In two-dimensional spaces, a hyperplane is simply a line (it will be a plane in $\mathbb{R}^3$ when $M = 3$). The figure shows two situations depending on whether the separating line passes through the origin or not. It is clear from the figure that separating lines are *not unique*, especially since the slopes of the lines can be altered in many ways and still succeed in separating the data into two classes. Once a separating hyperplane is chosen, with parameters denoted by $(w^\star, \theta^\star)$, then the classifier can be used to assign feature vectors h into one class or the other by performing the following check:

$$\begin{cases} \text{if } h^{\mathsf{T}}w^\star < \theta^\star, \text{ assign } h \text{ to class } -1 \\ \text{if } h^{\mathsf{T}}w^\star \geq \theta^\star, \text{ assign } h \text{ to class } +1 \end{cases} \tag{60.2}$$

In the sequel we derive the Perceptron algorithm, which will provide one way to determine a separating hyperplane $(w^\star, \theta^\star)$ from the training data $\{\gamma(n), h_n\}$.



separating line **passes** through the origin separating line **does not pass** through the origin

Figure 60.1 Illustration of linearly separable data in $\mathbb{R}^2$. The separating line on the left passes through the origin (i.e., it has a zero offset parameter), while the separating line on the right does not pass through the origin. The vector w represents the normal direction to the line.

60.2 PERCEPTRON EMPIRICAL RISK

Assuming the $N-$data points $\{\gamma(n), h_n\}$ are *linearly separable*, our objective is to construct a hyperplane $(w^\star, \theta^\star)$ that separates the data into its two classes. We pursue a geometric argument.

Let (w, θ) denote the parameters of some *generic* hyperplane. By definition, all vectors $h \in \mathbb{R}^M$ that lie on this hyperplane satisfy the equation

$$h^{\mathsf{T}} w - \theta = 0 \tag{60.3}$$

Moreover, the vector w is called the *normal direction* to the hyperplane. This is because if we consider any two vectors (h_a, h_b) on the hyperplane, i.e.,

$$h_a^{\mathsf{T}} w - \theta = 0, \quad h_b^{\mathsf{T}} w - \theta = 0 \tag{60.4}$$

then by subtracting we find that

$$(h_a - h_b)^{\mathsf{T}} w = 0 \tag{60.5}$$

so that w is orthogonal to the difference of any two vectors lying in the hyperplane — recall Fig. 56.6. Accordingly, with every hyperplane defined by the parameters (w, θ), we associate the unit-norm normal direction:

$$\text{unit-norm normal direction} \;=\; w/\|w\| \tag{60.6}$$

We wish to determine parameters (w, θ) such that any data pair $(\gamma(n), h_n)$ in the training set will be correctly classified by this hyperplane, namely, such that

$$\begin{cases} h_n^{\mathsf{T}} w - \theta > 0, & \text{when } \gamma(n) = +1 \\ h_n^{\mathsf{T}} w - \theta < 0, & \text{when } \gamma(n) = -1 \end{cases} \tag{60.7}$$

In the first case, h_n will lie on one side of the hyperplane, while in the second case it will lie on the other side. We can combine these two conditions into a single relation by writing that the choice for (w, θ) should enforce the following condition for all training data points, $n = 0, 1, \ldots, N-1$:

$$\gamma(n)\left(h_n^{\mathsf{T}} w - \theta\right) > 0, \qquad (\textbf{correct classification}) \tag{60.8}$$

Geometric construction

Pick an arbitrary vector h_a that belongs to the hyperplane (w, θ), i.e.,

$$h_a^{\mathsf{T}} w - \theta = 0 \tag{60.9}$$

The distance from any training feature h_n to the hyperplane w can be determined by projecting the vector difference $(h_n - h_a)$ onto the unit-norm direction $w/\|w\|$ and retaining the absolute value of this projection — see Fig. 60.2:

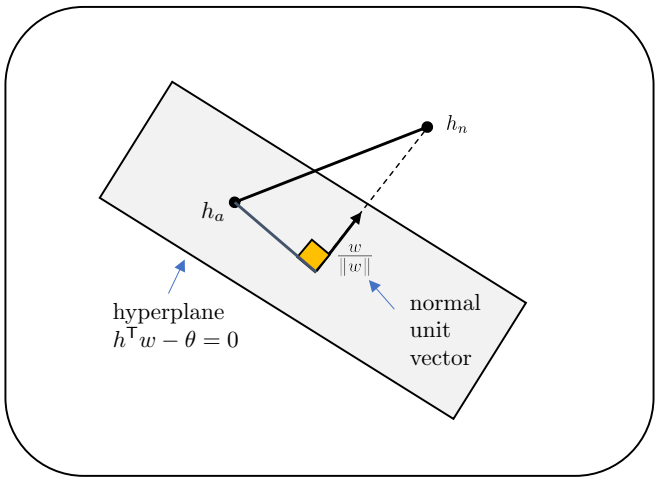


Figure 60.2 Distance from h_n to the separating hyperplane can be obtained by computing, for any h_a, the inner product of the difference $(h_n - h_a)$ and the unit-norm vector, $w/\|w\|$.

$$\begin{aligned}
\text{distance from } h_n \text{ to hyperplane} &= \left|(h_n - h_a)^{\mathsf{T}}\frac{w}{\|w\|}\right| \\
&= \left|(h_n^{\mathsf{T}}w - h_a^{\mathsf{T}}w)\frac{1}{\|w\|}\right| \\
&\overset{(60.9)}{=} \left|(h_n^{\mathsf{T}}w - \theta)\frac{1}{\|w\|}\right| \\
&\overset{(a)}{=} \left|\gamma(n)(h_n^{\mathsf{T}}w - \theta)\frac{1}{\|w\|}\right| \qquad (60.10)
\end{aligned}$$

where we added $\gamma(n)$ in step (a) because $|\gamma(n)| = 1$. We know from (60.8) that if $(\gamma(n), h_n)$ is misclassified by (w, θ), then $\gamma(n)\left(h_n^{\mathsf{T}}w - \theta\right) < 0$. When this occurs, the distance expression becomes:

$$\begin{aligned}
&\text{distance from a misclassified point } h_n \text{ to } (w, \theta) \\
&= -\gamma(n)\left(h_n^{\mathsf{T}}w - \theta\right)\frac{1}{\|w\|} \qquad (60.11)
\end{aligned}$$

If we add the distances of all misclassified points to the hyperplane (their index set is denoted by $\mathcal{M}$) we get:

$$-\frac{1}{\|w\|}\left(\sum_{n\in\mathcal{M}}\gamma(n)\left(h_n^{\mathsf{T}}w - \theta\right)\right) \qquad (60.12)$$

The scaling by $1/\|w\|$ is irrelevant since we can always re-normalize the separating hyperplane (w, θ) by scaling its w to have unit norm and by scaling θ

similarly, i.e., we can always replace any (w,θ) in (60.3) by $(w/\|w\|,\theta/\|w\|)$. As such, we will remove the scaling by $1/\|w\|$ from (60.12) and consider instead the sum:

$$\mathcal{S} = -\sum_{n\in\mathcal{M}} \gamma(n)\left(h_n^{\mathsf{T}} w - \theta\right) \tag{60.13}$$

In order to reduce classification errors on the $N-$training data points, we would like to keep this sum small. We can rewrite the above expression in an equivalent manner that incorporates all training points as follows:

$$\mathcal{S} = \sum_{n=0}^{N-1} \max\Big\{0, -\gamma(n)\left(h_n^{\mathsf{T}} w - \theta\right)\Big\} \tag{60.14}$$

where, by comparing against zero, we are in effect only keeping the contributions arising from the misclassified points $(\gamma(n), h_n)$. If we scale by $1/N$ we arrive at the *empirical risk* function that is associated with the Perceptron construction, namely,

$$(w^\star,\theta^\star) \triangleq \operatorname*{argmin}_{w\in\mathbb{R}^M,\theta\in\mathbb{R}} \left\{ P(w) \triangleq \frac{1}{N}\sum_{n=0}^{N-1} \max\Big\{0, -\gamma(n)(h_n^{\mathsf{T}} w - \theta)\Big\}\right\} \tag{60.15}$$

Clearly, when the data is linearly separable, a hyperplane $(w^\star,\theta^\star)$ exists that separates the data correctly and reduces the sum of misclassified distances in (60.13) to zero. If we invoke ergodicity, we find that $P(w)$ motivates the following *stochastic risk* function:

$$\frac{1}{N}\sum_{n=0}^{N-1} \max\Big\{0, -\gamma(n)(h_n^{\mathsf{T}} w - \theta)\Big\} \;\overset{N\to\infty}{\longrightarrow}\; \mathbb{E}\,\max\Big\{0, -\boldsymbol{\gamma}(\boldsymbol{h}^{\mathsf{T}} w - \theta)\Big\} \tag{60.16}$$

so that the Perceptron construction can also be interpreted as solving the following Bayesian inference problem:

$$w^o = \operatorname*{argmin}_{w\in\mathbb{R}^M,\theta\in\mathbb{R}} \left\{ \mathbb{E}\,\max\Big\{0, -\boldsymbol{\gamma}(\boldsymbol{h}^{\mathsf{T}} w - \theta)\Big\}\right\} \tag{60.17}$$

where the expectation is over the joint distribution of $(\boldsymbol{\gamma},\boldsymbol{h})$.

Online recursion

Problem (60.15) can now be solved by a variety of stochastic optimization methods, already discussed in previous chapters, such as using stochastic subgradient algorithms and variations thereof. It is sufficient to illustrate the construction by considering one solution method. We will therefore focus on stochastic subgradient implementations, with and without regularization, that rely on instantaneous subgradient approximations. The sampling of the data in the stochastic implementation can also be done with or without replacement.

In practice, the optimization problem (60.15) is modified to incorporate regularization for the reasons already explained in Chapter 51, such as reducing ill-conditioning, reducing the possibility of overfitting, and endowing $w^\star$ with desirable properties such as having a small norm or sparse structure. For illustration, we will consider Perceptron risks under ℓ_2-regularization and replace (60.15) by

$$(w^\star,\theta^\star) \triangleq \underset{w\in\mathbb{R}^M,\theta\in\mathbb{R}}{\operatorname{argmin}} \left\{\rho\|w\|^2 + \frac{1}{N}\sum_{n=0}^{N-1}\max\Big\{0,-\gamma(n)(h_n^{\mathsf{T}}w-\theta)\Big\}\right\} \tag{60.18}$$

where ρ is a nonnegative scalar. Using the result of Example 16.9, we show in (60.19) a listing for a regularized Perceptron algorithm for solving (60.18). The notation $\mathbb{I}[x]$ refers to the indicator function that is equal to one when condition x is true and zero otherwise.

Regularized Perceptron for minimizing (60.18)

given dataset $\{\gamma(m),h_m\}_{m=0}^{N-1}$ or streaming data $(\gamma(n),h_n)$;
start from an arbitrary initial condition, $\boldsymbol{w}_{-1}$.
repeat until convergence over $n\geq 0$:
  select at random or receive a sample $(\boldsymbol{\gamma}(n),\boldsymbol{h}_n)$ at iteration n;
  $\widehat{\boldsymbol{\gamma}}(n) = \boldsymbol{h}_n^{\mathsf{T}}\boldsymbol{w}_{n-1} - \boldsymbol{\theta}(n-1)$
  $\boldsymbol{\theta}(n) = \boldsymbol{\theta}(n-1) - \mu\boldsymbol{\gamma}(n)\,\mathbb{I}\,[\boldsymbol{\gamma}(n)\widehat{\boldsymbol{\gamma}}(n)\leq 0]$
  $\boldsymbol{w}_n = (1-2\mu\rho)\boldsymbol{w}_{n-1} + \mu\boldsymbol{\gamma}(n)\boldsymbol{h}_n\,\mathbb{I}\,[\boldsymbol{\gamma}(n)\widehat{\boldsymbol{\gamma}}(n)\leq 0]$
end
return $w^\star\leftarrow\boldsymbol{w}_n$, $;\theta^\star\leftarrow\boldsymbol{\theta}(n)$;
classify a feature h by using the sign of $\widehat{\gamma} = h^{\mathsf{T}}w^\star-\theta^\star$. (60.19)

We can simplify the notation by extending the feature and weight vectors as follows:

$$h\leftarrow\begin{bmatrix}1\\ h\end{bmatrix},\quad w\leftarrow\begin{bmatrix}-\theta\\ w\end{bmatrix} \tag{60.20}$$

so that the recursions in (60.19) can be rewritten more compactly in the following manner where the offset parameter is now implicit:

$$\begin{cases}\widehat{\boldsymbol{\gamma}}(n) = \boldsymbol{h}_n^{\mathsf{T}}\boldsymbol{w}_{n-1}\\ \boldsymbol{w}_n = A\boldsymbol{w}_{n-1} + \Big(\mu\boldsymbol{\gamma}(n)\,\mathbb{I}\,[\boldsymbol{\gamma}(n)\widehat{\boldsymbol{\gamma}}(n)\leq 0]\Big)\boldsymbol{h}_n,\quad n\geq 0\end{cases} \tag{60.21}$$

and the diagonal matrix A depends on the regularization parameter:

$$A \triangleq \begin{bmatrix}1 & \\ & (1-2\mu\rho)I_M\end{bmatrix} \tag{60.22}$$

When a mini-batch of size B is used, the Perceptron recursion is replaced by

$$\begin{cases} \text{select } B \text{ data samples } \{\boldsymbol{\gamma}(b), \boldsymbol{h}_b\} \text{ at random} \\ \boldsymbol{\gamma}(b) = \boldsymbol{h}_b^{\mathsf{T}} \boldsymbol{w}_{n-1}, \;\; b = 0, 1, \ldots, B-1 \\ \boldsymbol{w}_n = A\boldsymbol{w}_{n-1} + \displaystyle\sum_{b=0}^{B-1} \Big(\mu \boldsymbol{\gamma}(b)\, \mathbb{I}\left[\boldsymbol{\gamma}(b)\widehat{\boldsymbol{\gamma}}(b) \le 0\right] \Big) \boldsymbol{h}_b, \;\; n \ge 0 \end{cases} \tag{60.23}$$

On the other hand, in the absence of regularization ($\rho = 0$), we obtain the classical Perceptron update:

$$\boldsymbol{w}_n = \boldsymbol{w}_{n-1} + \mu \boldsymbol{\gamma}(n) \boldsymbol{h}_n\, \mathbb{I}\left[\boldsymbol{\gamma}(n)\widehat{\boldsymbol{\gamma}}(n) \le 0\right], \;\; n \ge 0 \tag{60.24}$$

which can be rewritten in the equivalent form

$$\boxed{\boldsymbol{w}_n = \boldsymbol{w}_{n-1} \;+\; \mu \boldsymbol{\gamma}(n) \boldsymbol{h}_n, \;\; \text{if } \; \boldsymbol{\gamma}(n)\widehat{\boldsymbol{\gamma}}(n) \le 0} \tag{60.25}$$

That is, the weight iterate is updated from $\boldsymbol{w}_{n-1}$ to $\boldsymbol{w}_n$ only when the data point $(\boldsymbol{\gamma}(n), \boldsymbol{h}_n)$ is misclassified, i.e., when the signs of $\boldsymbol{\gamma}(n)$ and $\widehat{\boldsymbol{\gamma}}(n)$ do not match, in which case the vectors $\boldsymbol{\gamma}(n)\boldsymbol{h}_n$ and $\boldsymbol{w}_{n-1}$ will have a non-positive inner product. This situation is illustrated in Fig. 60.3. We therefore find that the Perceptron iteration (60.25) perturbs $\boldsymbol{w}_{n-1}$ to $\boldsymbol{w}_n$ in order to obtain a vector $\boldsymbol{w}_n$ that is more correlated with $\boldsymbol{\gamma}(n)\boldsymbol{h}_n$.

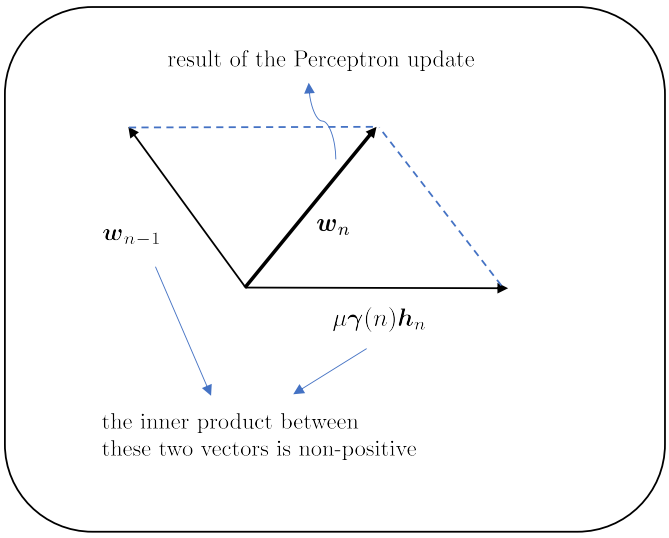


Figure 60.3 Iteration (60.25) updates $\boldsymbol{w}_{n-1}$ to $\boldsymbol{w}_n$ in order to obtain a vector $\boldsymbol{w}_n$ that is more correlated with $\boldsymbol{\gamma}(n)\boldsymbol{h}_n^{\mathsf{T}}$.

It is useful to note that the inequality condition $\boldsymbol{\gamma}(n)\widehat{\boldsymbol{\gamma}}(n) \le 0$ in (60.25) cannot be replaced by the strict inequality $\boldsymbol{\gamma}(n)\widehat{\boldsymbol{\gamma}}(n) < 0$. This is because if we start from the initial condition $\boldsymbol{w}_{-1} = 0$, as is typical in Perceptron implementations,

then $\widehat{\gamma}(0) = \boldsymbol{h}_0^{\mathsf{T}}\boldsymbol{w}_{-1} = 0$ and the recursion will never update the weight iterate. Moreover, in most implementations of the Perceptron algorithm, the step-size parameter is set to $\mu = 1$, which leads to — see the explanation after (63.31) and also Prob. 60.1:

$$\boldsymbol{w}_n = \boldsymbol{w}_{n-1} + \boldsymbol{\gamma}(n)\boldsymbol{h}_n, \quad \text{if } \boldsymbol{\gamma}(n)\widehat{\boldsymbol{\gamma}}(n) \le 0 \tag{60.26}$$

Example 60.1 (**Binary classification using Perceptron**) Figure 60.4 shows a collection of 150 feature samples $\boldsymbol{h}_n \in \mathbb{R}^2$ whose classes ± 1 are known beforehand: 120 samples are selected for training and 30 samples are selected for testing. The data arises from the dimensionally reduced iris dataset from Example 57.3; we denoted the two-dimensional reduced feature vectors by the notation h'_n in that example. We denote them by h_n here. We employ the two classes shown in the bottom plot of Fig. 57.5 and denote them by $\gamma(n) \in \{\pm 1\}$. We extend the feature data and weight vector according to (60.20).

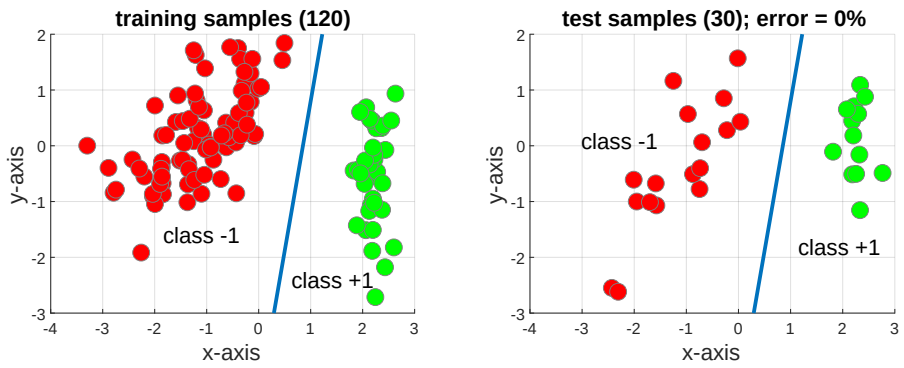


Figure 60.4 The plots show 120 data points used for training (*left*) and 30 data points used for testing (*right*). The separating line is obtained by running the Perceptron algorithm (60.27a)–(60.27b) five times over the training data.

We use 120 samples to train the Perceptron classifier by running 5 passes over the data:

$$\widehat{\gamma}(n) = h_n^{\mathsf{T}} w_{n-1} \tag{60.27a}$$

$$w_n = w_{n-1} + \gamma(n)h_n, \quad \text{if } \gamma(n)\widehat{\gamma}(n) \le 0 \tag{60.27b}$$

During each pass of the algorithm, the data $\{\gamma(n), h_n\}$ is randomly reshuffled and the algorithm is re-run starting from the weight iterate obtained at the end of the previous pass. The line in the figure shows the separating curve obtained in this manner with parameters (after undoing the extension (60.20)):

$$w^\star = \begin{bmatrix} 2.3494 \\ -0.4372 \end{bmatrix}, \quad \theta^\star = 2.0 \tag{60.28}$$

It is seen that the separation curve is able to classify all test vectors and leads to a 0% empirical error rate.

60.3 TERMINATION IN FINITE STEPS

One useful property of the Perceptron algorithm (60.25) is that it terminates in a *finite number* of steps for linearly separable data $\{\gamma(n), h_n\}$. To see this, recall first that linear separability means that there exists at least one vector $w^\star$ that is able to separate the data into two classes and satisfy

$$\exists\, w^\star \text{ such that } \gamma(n) h_n^{\mathsf{T}} w^\star > 0, \quad \text{for } n = 0, 1, \ldots, N-1 \tag{60.29}$$

where we are assuming that the feature data and the weight vector have been extended according to (60.20). When this happens, there will exist at least one point h in the training data that will be closest to the hyperplane $w^\star$. The distance from this closest point to the hyperplane is called the *margin*. This situation is illustrated in Fig. 60.5. Using expression (60.10) with $w = w^\star$ and setting $\theta = 0$ (since h_n and $w^\star$ are assumed to have been extended), we find that the margin can be found by computing:

$$m(w^\star) \triangleq \min_{0 \le n \le N-1} \left\{ \frac{\gamma(n) h_n^{\mathsf{T}} w^\star}{\|w^\star\|} \right\} \overset{(60.29)}{=} \min_{0 \le n \le N-1} \left\{ \frac{|h_n^{\mathsf{T}} w^\star|}{\|w^\star\|} \right\} \tag{60.30}$$

Observe that the margin is dependent on the choice of $w^\star$. The next result shows that the performance of the Perceptron algorithm is sensitive to the margin. Specifically,the number of mis-classifications encountered by the algorithm is inversely proportional to $m^2(w^\star)$ so that larger margins are preferable. Although the result guarantees convergence in a finite number of steps, the number of steps required can still be large because the margin can be small.

Lemma 60.1. (Finite number of errors) *Assume the $N-$size dataset $\{\gamma(m), h_m\}$ is linearly separable, i.e., there exists at least one vector $w^\star$ satisfying (60.29) and denote its margin by $m(w^\star)$. Assume further that the feature vectors are bounded, say, $\|h_n\| \le H$ for all n. The Perceptron algorithm (60.25) is applied continuously over the data, including possibly multiple passes, as needed. At any iteration t, the total number of erroneous misclassifications encountered by the algorithm until that point in time is bounded by*

$$|\mathcal{M}_t| \;\le\; \frac{H^2}{m^2(w^\star)} \tag{60.31}$$

Since the Perceptron algorithm updates only when misclassifications occur, it follows from this result that the algorithm will only perform a finite number of updates.

Proof: We refer to (60.25) and assume, without loss of generality, that the algorithm starts from the initial condition $w_{-1} = 0$; if w_{-1} is nonzero, then we should incorporate its value into the derivation below. Let $\mathcal{M}_t$ denote the collection of all iteration indexes for which the algorithm encounters a misclassification (i.e., when it performs

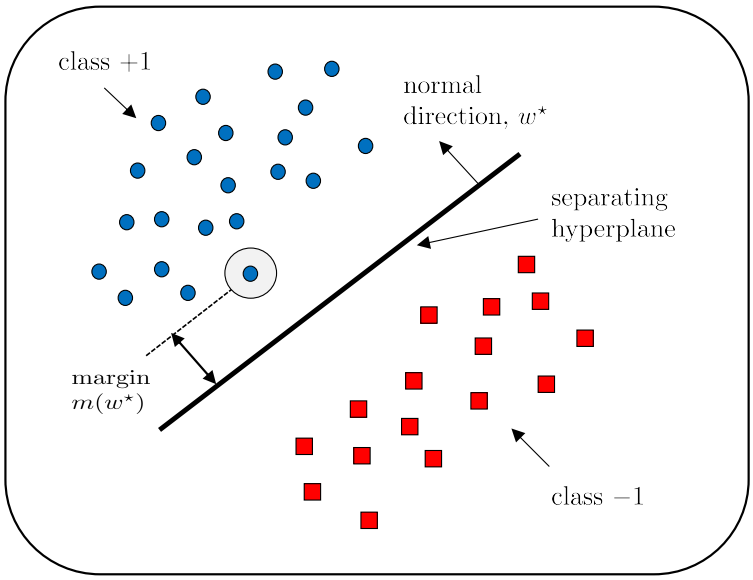


Figure 60.5 The closest feature vector to the separating hyperplane $w^\star$ is highlighted inside a circle with its distance to $w^\star$ representing the margin, $m(w^\star)$.

an update). Iterating (60.25), we find that at any iteration t:

$$w_t = \mu \left(\sum_{n \in \mathcal{M}_t} \gamma(n) h_n \right) \tag{60.32}$$

where the sum is over the set of misclassified data up to time t, i.e., points for which

$$\gamma(n) h_n^{\mathsf{T}} w_{n-1} \leq 0, \quad \text{for } n \in \mathcal{M}_t \tag{60.33}$$

and $(\gamma(n), h_n)$ is the sample pair selected at the $n-$th iteration. Computing the inner product of w_t with $w^\star$ we get

$$\begin{aligned} w_t^{\mathsf{T}} w^\star &= \mu \left(\sum_{n \in \mathcal{M}_t} \gamma(n) h_n^{\mathsf{T}} w^\star \right) \\ &\overset{(60.29)}{=} \mu \left(\sum_{n \in \mathcal{M}_t} |h_n^{\mathsf{T}} w^\star| \right) \\ &\overset{(60.30)}{\geq} \mu \, m(w^\star) \, \|w^\star\| \, |\mathcal{M}_t| \end{aligned} \tag{60.34}$$

in terms of the cardinality of the set $\mathcal{M}_t$. It follows that

$$\|w_t\|^2 \, \|w^\star\|^2 \geq |w_t^{\mathsf{T}} w^\star|^2 \geq \mu^2 m^2(w^\star) \, \|w^\star\|^2 \, |\mathcal{M}_t|^2 \tag{60.35}$$

where we applied the Cauchy-Schwarz inequality for the inner product of two vectors, which states that $|a^{\mathsf{T}} b| \leq \|a\| \, \|b\|$. We then arrive at the lower bound:

$$\|w_t\|^2 \geq \mu^2 m^2(w^\star) \, |\mathcal{M}_t|^2 \tag{60.36}$$

We can similarly derive an upper bound for $\|w_t\|^2$ as follows. We return to the Perceptron recursion (60.25) and note that, for any step t where an update occurs:

$$\begin{aligned}\|w_t\|^2 &= \|w_{t-1} \;+\; \mu\gamma(t)h_t\|^2 \\ &= \|w_{t-1}\|^2 \;+\; \mu^2\gamma^2(t)\|h_t\|^2 \;+\; 2\mu\gamma(t)h_t^{\mathsf{T}}w_{t-1} \\ &\le \|w_{t-1}\|^2 \;+\; \mu^2\gamma^2(t)\|h_t\|^2 \quad \text{(because of (60.33))} \\ &= \|w_{t-1}\|^2 \;+\; \mu^2\|h_t\|^2 \qquad \text{(since } \gamma^2(t)=1\text{)} \end{aligned} \tag{60.37}$$

Iterating starting from w_{-1}, we find that

$$\|w_t\|^2 \;\le\; \mu^2\left(\sum_{n\in\mathcal{M}_t}\|h_n\|^2\right) \tag{60.38}$$

and we arrive at the upper bound

$$\|w_t\|^2 \;\le\; \mu^2 H^2\,|\mathcal{M}_t| \tag{60.39}$$

Combining this result with (60.36) we conclude that

$$\mu^2 m^2(w^\star)\,|\mathcal{M}_t|^2 \;\le\; \|w_t\|^2 \;\le\; \mu^2 H^2\,|\mathcal{M}_t| \tag{60.40}$$

These bounds are valid as long as the lower bound is smaller than the upper bound, i.e.,

$$\mu^2 m^2(w^\star)\,|\mathcal{M}_t|^2 \;\le\; \mu^2 H^2\,|\mathcal{M}_t| \tag{60.41}$$

which is only satisfied if the *number of updates* (and, hence, the number of erroneous decisions) is bounded according to (60.31). This result holds irrespective of the value of t and is also independent of the feature dimension, M. The bound in (60.31) confirms termination of the Perceptron algorithm after a finite number of iterations for linearly separable data.

■

60.4 POCKET PERCEPTRON

When the training data $\{\gamma(m), h_m\}$ is *not* linearly separable, the Perceptron iteration (60.25) will not terminate and the weight vector will continue to update and possibly move from a "good" to a "bad" solution, i.e., from a hyperplane that separates well a large fraction of the data to another hyperplane that performs poorly on the same data. One variation that improves the behavior of Perceptron under these circumstances is to introduce a *pocket* variable to keep track of the best iterate. At the conclusion of the training phase, the pocket variable provides the desired weight estimate. Pocket Perceptron operates as follows.

Let $w_p \in \mathbb{R}^M$ denote the weight iterate that is saved in the "pocket". We set its value initially to some random vector (e.g., the zero vector) and evaluate its empirical error rate over the N training data points, $\{\gamma(m), h_m\}$ (i.e., we compute the fraction of incorrect classifications by w_p). We denote this value by

$$R_p \;\triangleq\; \frac{1}{N}\sum_{m=0}^{N-1}\mathbb{I}\left[\gamma(m)h_m^{\mathsf{T}}w_p \le 0\right] \tag{60.42}$$

At any subsequent iteration of index n, the Perceptron recursion (60.26) updates w_{n-1} to a new value w_n only when w_{n-1} misclassifies h_n. Each time an update occurs, from w_{n-1} to w_n, we compute the empirical error rate of the new iterate over the entire training dataset:

$$R(w_n) \triangleq \frac{1}{N}\sum_{m=0}^{N-1} \mathbb{I}\left[\gamma(m)h_m^{\mathsf{T}}w_m \le 0\right] \tag{60.43}$$

and compare it against R_p in order to decide whether to replace the pocket variable by the new value, w_n:

$$\text{if } R(w_n) < R_p \text{ then } w_p \leftarrow w_n \text{ and } R_p \leftarrow R(w_n) \tag{60.44}$$

At the end of the training phase, the hyperplane that is selected as the final classifier is the one that has been saved in the pocket, i.e., w_p. One inconvenience of this implementation is that it assumes, at every iteration, that the algorithm has access to the entire training data to assess the empirical error rates, $R(w_n)$.

Pocket Perceptron algorithm for binary classification

given dataset $\{\gamma(m), h_m\}_{m=0}^{N-1}$;
assume vectors are extended according to (60.20);
start from initial conditions, $\boldsymbol{w}_{-1} = \boldsymbol{w}_p = 0_M, R_p = 1$.
repeat until convergence over $n \ge 0$:
  select at random $(\boldsymbol{\gamma}(n), \boldsymbol{h}_n)$ at iteration n;
  $\widehat{\boldsymbol{\gamma}}(n) = \boldsymbol{h}_n^{\mathsf{T}}\boldsymbol{w}_{n-1}$
  if $\boldsymbol{\gamma}(n)\widehat{\boldsymbol{\gamma}}(n) \le 0$:
    $\boldsymbol{w}_n = \boldsymbol{w}_{n-1} + \boldsymbol{\gamma}(n)\boldsymbol{h}_n$
    $R(\boldsymbol{w}_n) = \dfrac{1}{N}\displaystyle\sum_{m=0}^{N-1} \mathbb{I}\left[\boldsymbol{\gamma}(m)\boldsymbol{h}_m^{\mathsf{T}}\boldsymbol{w}_n \le 0\right]$
    if $R(\boldsymbol{w}_n) < R_p$
      $\boldsymbol{w}_p \leftarrow \boldsymbol{w}_n$
      $R_p \leftarrow R(\boldsymbol{w}_n)$
    end
  end
end
return $w^\star \longleftarrow w_p$

(60.45)

Example 60.2 (**Binary classification using pocket Perceptron**) Figure 60.6 illustrates the behavior of the pocket algorithm on training samples that are not linearly separable. The data arises from the dimensionally-reduced iris dataset from Example 57.3; we denoted the two-dimensional reduced feature vectors by the notation h_n' in that example. We denote them by h_n here. We consider the situation involving three classes shown in the top plot of Fig. 57.5, and extract the data corresponding to classes $r = 1$ (versicolor) and $r = 2$ (virginica) — see, for example, the bottom rightmost plot in

Fig. 59.9. We denote these two classes by $\gamma(n) \in \{\pm 1\}$. There are a total of 100 data samples; we select 80 samples for training and 20 samples for testing.

We use the 80 samples to train the traditional Perceptron classifier (60.26) and the pocket Perceptron classifier (60.45), both under extensions (60.20). In each case, we run 5 passes of the algorithm over the training data using random reshuffling. The lines in Fig. 60.6 show the separating curves obtained in this manner with parameters

$$w^\star = \begin{bmatrix} 4.2001 \\ -0.4662 \end{bmatrix}, \quad \theta^\star = -5.0, \quad \text{(traditional Perceptron)} \tag{60.46}$$

$$w^\star = \begin{bmatrix} 3.9984 \\ -1.6366 \end{bmatrix}, \quad \theta^\star = -5.0, \quad \text{(pocket Perceptron)} \tag{60.47}$$

The resulting empirical error rates on the test data are 20% for Perceptron (4 misclassifications in 20 test samples) and 10% for Pocket Perceptron (2 misclassifications in 20 test samples). The empirical error rates over the training data are 16.25% and 10%, respectively.

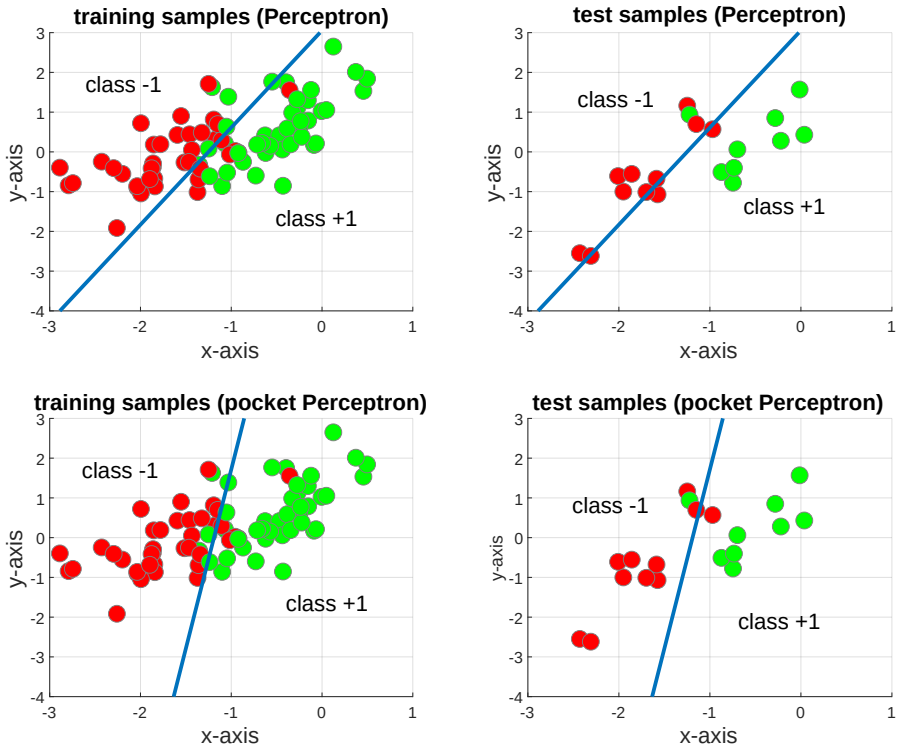


Figure 60.6 The plots show 80 training samples and 20 test samples (*top row*), and the resulting separation lines by means of the Perceptron classifier (60.26) and the pocket Perceptron classifier (60.45).

Example 60.3 (**Application to the heart disease data**) We reconsider the dimensionally-reduced heart disease dataset from Example 57.4. In particular, we consider the data samples shown in the bottom scatter plot of Fig. 57.6 where the feature vectors have

been reduced to dimension 3. We denote these features vectors by the notation $\{h_n\}$ in this example (as opposed to $\{h'_n\}$ used in Example 57.4); we also denote their dimension by $M = 3$. The data in that figure have been aggregated into two classes: presence of heart disease (which we now assign the label $+1$) and absence of heart disease (which we now assign the label -1).

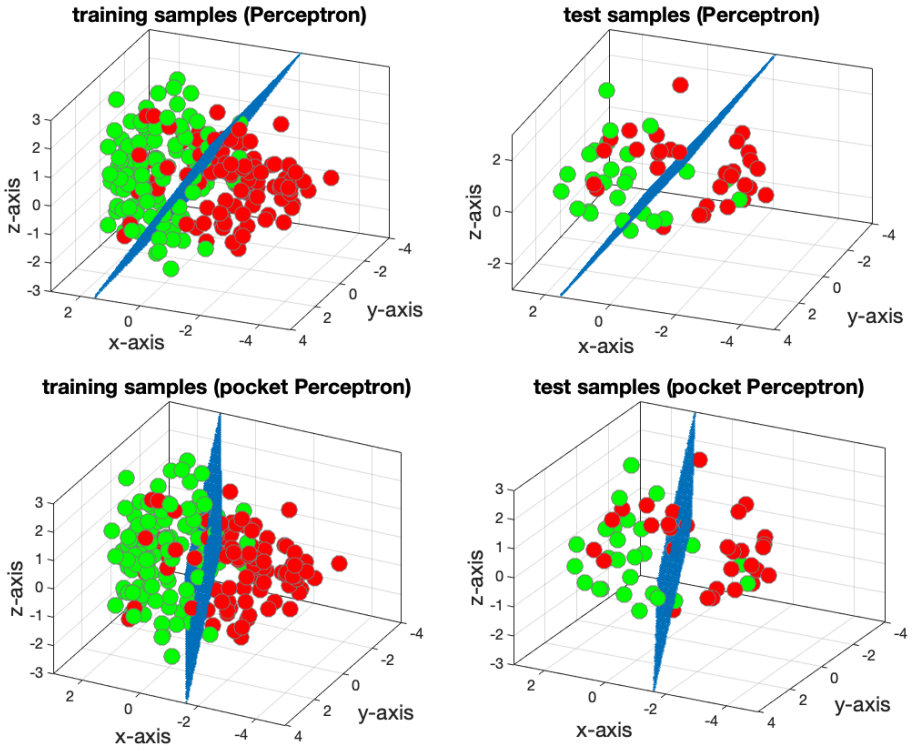


Figure 60.7 The plots show 238 training samples and 59 test samples in three-dimensional space (*top row*), and the resulting separation curves by means of the Perceptron classifier (60.26) and the pocket Perceptron classifier (60.45).

There are a total of 297 data samples; we select 238 samples for training and 59 samples for testing (that amounts to 20% of the total number of samples). We use the data to train the traditional Perceptron classifier (60.26) and the pocket Perceptron classifier (60.45), both under extensions (60.20). In each case, we run 50 passes of the algorithms over the training data using random reshuffling. The results are shown in Fig. 60.7. The hyperplanes in the figure show the separating curves obtained in this manner with parameters

$$w^\star = \begin{bmatrix} 5.2289 \\ 2.6399 \\ 1.0637 \end{bmatrix}, \quad \theta^\star = -1.0, \qquad (\textbf{traditional Perceptron}) \tag{60.48}$$

and

$$w^\star = \begin{bmatrix} 3.8486 \\ 0.1409 \\ 2.5030 \end{bmatrix}, \quad \theta^\star = -1.0, \qquad (\textbf{pocket Perceptron}) \tag{60.49}$$

The resulting empirical error rates on the test data are 33.90% for Perceptron (20 misclassifications in 59 test samples) and 22.03% for Pocket Perceptron (13 misclassifications in 59 test samples). The empirical error rates over the training data are 20.59% and 13.45%, respectively.

Table 60.1 Empirical error rates over test and training data for both cases of 13− and 3−dimensional feature vectors.

algorithm	M	N	N_{train}	N_{test}	training error	testing error
Perceptron	13	297	238	59	17.23%	27.12%
pocket Perceptron	13	297	238	59	11.34%	17.23%
Perceptron	3	297	238	59	20.59%	33.90%
pocket Perceptron	3	297	238	59	13.45%	22.03%

We repeat the same procedure and apply Perceptron and pocket Perceptron to the heart disease dataset without reducing the dimension of the feature space. Recall that originally each feature consists of $M = 13$ attributes. We center the feature vectors around their mean and scale their variance to one, as was described earlier in the preprocessing steps for PCA in (57.6). We subsequently apply Perceptron and pocket Perceptron to 238 training samples from this set and test the performance on 59 other samples. We also test the performance on the training samples. Table 60.1 summarizes the empirical error rates obtained for both the reduced and full feature vectors. The symbols N_{train} and N_{test} refer to the number of samples used for training and testing.

60.5 COMMENTARIES AND DISCUSSION

The Perceptron. The word "Perceptron" appears to be a shorthand for the combination "perception automaton" and is nowadays commonly used to refer to the Perceptron structure (60.25). This algorithm corresponds to a linear classification rule that is guaranteed to converge in a finite number of iterations for linearly separable data, as explained in Sec. 60.3. The first works establishing bounds similar to (60.31) are by Block (1961,1962) and Novikoff (1962). More discussion on linear separability is included in Appendices 60.A and 60.B. The Perceptron rule was introduced and implemented into a hardware unit in 1957 by the American psychologist **Frank Rosenblatt (1928–1971)**. Rosenblatt (1957,1958) was interested in pattern classification problems while working at the Cornell Aeronautical Laboratory. He was motivated by the work performed about a decade earlier in 1949 by the Canadian neuroscientist **Donald Hebb (1904–1985)** on a model for the neural activity in the human brain. For additional information on the Perceptron and its history, the reader may refer to Rosenblatt (1962), Minsky and Papert (1969), Duda and Hart (1973), Widrow and Lehr (1990), Peretto (1992), Haykin (1999), Siu, Roychowdhury, and Kailath (1995), and Theodoridis (2015).

Hebbian model. In his influential text, Hebb (1949) postulated on how neurons in the brain adjust their connection strength. He argued that when a neural cell A is repeatedly involved in firing another neural cell B, then the strength of the synaptic weight linking A to B should increase so that the role of A in firing B is enhanced. This postulate motivated the following algorithmic construction — see the diagram on the left-hand side of Fig. 60.8.

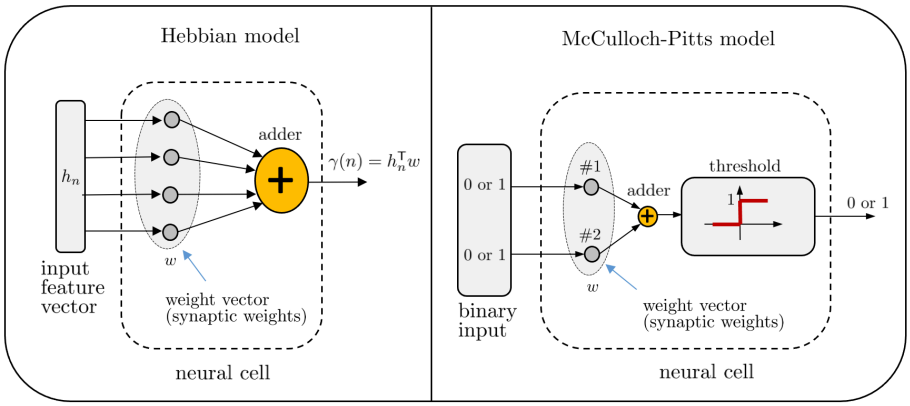


Figure 60.8 A diagram representation of the Hebbian neural model (*left*) and McCulloch-Pitts neural model (*right*) with binary input signals and a binary output.

Assume there are several neurons connected to B. We assign one scaling weight to the link between B and each of these neurons. This results in a linear combination output at B of the form $\boldsymbol{\gamma}(n) = \boldsymbol{h}_n^{\mathsf{T}} w$. Here, the vector w contains the synaptic weights and the vector $\boldsymbol{h}_n \in \mathbb{R}^M$ contains the incoming signals at each of the feeding neurons into B. The variable $\boldsymbol{\gamma}(n)$ denotes the output signal by neuron B at instant n. The Hebbian learning rule for adjusting the synaptic weights takes the following form:

$$\left\{ \begin{array}{l} \widehat{\boldsymbol{\gamma}}(n) = \boldsymbol{h}_n^{\mathsf{T}} \boldsymbol{w}_{n-1} \\ \boldsymbol{w}_n = \boldsymbol{w}_{n-1} + \mu \widehat{\boldsymbol{\gamma}}(n) \boldsymbol{h}_n \end{array} \right. \tag{60.50}$$

with a plus sign in the second equation! In this expression, each entry of $\boldsymbol{w}_{n-1}$ is adjusted in proportion to the corresponding entry in $\boldsymbol{h}_n$. Observe that the Hebbian rule (60.50) is an *unsupervised* learning rule; it only relies on knowledge of the feature data, $\{\boldsymbol{h}_n\}$. It is instructive to compare this form with the Perceptron update (60.25), namely,

$$\boldsymbol{w}_n = \boldsymbol{w}_{n-1} \; + \; \mu \boldsymbol{\gamma}(n) \boldsymbol{h}_n, \quad \text{if } \boldsymbol{\gamma}(n) \widehat{\boldsymbol{\gamma}}(n) \leq 0 \tag{60.51}$$

The Perceptron update is a *supervised rule*; its structure is similar to the Hebbian rule except that $\widehat{\boldsymbol{\gamma}}(n)$ is replaced by the true class variable, $\boldsymbol{\gamma}(n)$, and the update is only performed when misclassifications occur. The fact that the Hebbian rule relies on $\widehat{\boldsymbol{\gamma}}(n)$ makes it an *unstable algorithm* since its weights grow unbounded. The Hebbian rule was motivated heuristically by Hebb (1949), using intuition from biological data. The history of this rule serves as a good example of how closer bridges between the biological and mathematical sciences can help avoid unreasonable models. The instability pitfall in the Hebbian update can be seen from several perspectives. First, note that we can rewrite the Hebbian rule (60.50) in the equivalent form:

$$\boldsymbol{w}_n = \left(I_M + \mu \boldsymbol{h}_n \boldsymbol{h}_n^{\mathsf{T}} \right) \boldsymbol{w}_{n-1} \tag{60.52}$$

This is a first-order recursion. Assuming independent and identically-distributed feature vectors $\{\boldsymbol{h}_n\}$ and letting $R_h = \mathbb{E}\, \boldsymbol{h}_n \boldsymbol{h}_n^{\mathsf{T}} \geq 0$, it follows under expectation that

$$\mathbb{E}\, \boldsymbol{w}_n = (I_M + \mu R_h) \mathbb{E}\, \boldsymbol{w}_{n-1} \tag{60.53}$$

This is an *unstable* recursion since the spectral radius of $I_M + \mu R_h$ is larger than one. A second way to explain the instability problem is to observe that the Hebbian rule (60.50) can be interpreted as a stochastic-gradient iteration for *maximizing* (rather than

minimizing) the variance $P(w) = \mathbb{E}\,(\boldsymbol{h}^{\mathsf{T}}w)^2$, which is *convex* over w. Yet another way to highlight the instability problem is to note that the maximization of $P(w)$ amounts to determining a vector w that solves (assuming $\boldsymbol{h}$ has zero mean):

$$w^o \;\triangleq\; \underset{w\in\mathbb{R}^M}{\operatorname{argmax}} \left\{ w^{\mathsf{T}} R_h w \right\} \tag{60.54}$$

This is an ill-posed problem since we know from the Rayleigh-Ritz characterization (1.16) for the largest eigenvalue of R_h that

$$w^{\mathsf{T}} R_h w \;\leq\; \lambda_{\max}(R_h)\|w\|^2 \tag{60.55}$$

and that equality is achieved when w is an eigenvector for R_h corresponding to $\lambda_{\max}$. However, there are infinitely many such eigenvectors since any eigenvector can be scaled up or down and it continues to be an eigenvector. Therefore, without any constraint on the norm of w, the bound on the right-hand side of (60.55) can be made arbitrarily large.

Oja rule. It took over three decades until a viable stable variant to the Hebbian rule was proposed by Oja (1982,1983). The resulting recursion is nowadays known as Oja rule and it takes the following form:

$$\boldsymbol{w}_n = \boldsymbol{w}_{n-1} + \mu\widehat{\boldsymbol{\gamma}}(n)(\boldsymbol{h}_n - \widehat{\boldsymbol{\gamma}}(n)\boldsymbol{w}_{n-1}) \tag{60.56}$$

For comparison purposes with the Hebbian rule (60.52), we can rewrite (60.56) in the equivalent form:

$$\boldsymbol{w}_n = (I + \mu\boldsymbol{h}_n\boldsymbol{h}_n^{\mathsf{T}})\boldsymbol{w}_{n-1} \;-\; \mu\,(\widehat{\boldsymbol{\gamma}}(n))^2\,\boldsymbol{w}_{n-1} \tag{60.57}$$

which shows that we now have an additional decay term that is proportional to the square $\widehat{\boldsymbol{\gamma}}^2(n)$. $(\widehat{\boldsymbol{\gamma}}(n))^2$. For a fixed $\widehat{\boldsymbol{\gamma}}$, the update (60.56) can be "motivated" as a stochastic-gradient iteration for *minimizing* $P(w) = \mathbb{E}\,\|\boldsymbol{h} - \widehat{\boldsymbol{\gamma}}w\|^2$, which is convex over w. This is not how Oja (1982) motivated the rule. However, by considering this risk function, the rule can be explained as follows. As $n \to \infty$, we expect the product $\widehat{\boldsymbol{\gamma}}w$ to approach $\boldsymbol{h}$ in order to minimize the mean-square-error:

$$\boldsymbol{h}_n \approx \widehat{\boldsymbol{\gamma}}(n)\boldsymbol{w}_{n-1} \;\;\Longrightarrow\;\; \underbrace{\boldsymbol{w}_{n-1}^{\mathsf{T}}\boldsymbol{h}_n}_{=\,\widehat{\boldsymbol{\gamma}}(n)} \approx \widehat{\boldsymbol{\gamma}}(n)\|\boldsymbol{w}_{n-1}\|^2 \;\Longrightarrow\; \|\boldsymbol{w}_{n-1}\|^2 \approx 1 \tag{60.58}$$

In this way, the norm of the weight vector will approach the value one (and remain bounded). Actually, it can be verified that Oja rule seeks a solution to (60.54) subject to the constraint $\|w\|^2 = 1$ — see Prob. 60.10 and also Oja (1992). In this way, Oja rule converges towards an estimate for the unit-norm eigenvector of R_h corresponding to its largest eigenvalue. As a result, there is a strong connection between Oja rule and the principal component analysis (PCA) method studied in Chapter 57. In particular, Oja rule is in effect approximating the first column of U in (57.20) by solving a problem similar to (57.19) — recall Prob. 57.5.

McCulloch-Pitts model. Hebb's (1949) rule was a generalization of an earlier model for neural activity introduced by McCulloch and Pitts (1943) in a famous article. As indicated in the diagram on the right-hand side of Fig. 60.8, they considered a simpler neural model consisting of two neurons feeding into a threshold unit. They limited the input signals to binary values 0 and 1. If the threshold value is set to 1, then the input-output mapping of this neural model emulates the behavior of the OR logical function as illustrated in Table 60.2. If, on the other hand, the threshold value is set to 2, then the input-output mapping emulates the behavior of the AND logical function. The McCulloch-Pitts model is limited in its capability since it restricts the input signals to binary values and does not associate weights with the links. Hebb's (1949) model, as well as the subsequent work by Rosenblatt (1957,1958) on the Perceptron, allowed for extensions in both of these domains as well as for the critical insight of adjusting

the weights over time.

Table 60.2 Input-output mapping of the McCulloch-Pitts neural model with binary input signals and a threshold value set at 1.

neuron #1	neuron #2	output
0	0	0
0	1	1
1	0	1
1	1	1

Pocket Perceptron. We indicated in Sec. 60.3 that the Perceptron recursion terminates in a finite number of steps for linearly separable data — see, e.g., Block (1961,1962) and Novikoff (1962). When the data is not linearly separable, the Perceptron recursion will not terminate and the algorithm may move from a good solution to a bad one as it updates. The pocket Perceptron algorithm improves performance under these circumstances; it was proposed by Gallant (1986,1990).

Separation theorem. There are important results in geometry due to Minkowski (1911) that ensure the existence of hyperplanes that separate disjoint convex sets in $\mathbb{R}^M$. These results are relevant to the concept of linear separability. Consider first a nonempty convex set $\mathcal{C} \subset \mathbb{R}^M$ and an arbitrary point $z_o \notin \mathcal{C}$. The so-called *supporting hyperplane theorem* affirms the existence of a hyperplane passing through z_o with the set $\mathcal{C}$ belonging to one of its halfspaces, i.e., there exists $w \in \mathbb{R}^M$ and $\theta \in \mathbb{R}$ such that

$$z_o^{\mathsf{T}} w - \theta = 0, \quad \text{and} \quad \sup_{c \in \mathcal{C}} \Big\{ c^{\mathsf{T}} w - \theta \Big\} \leq 0 \tag{60.59}$$

This is also equivalent to stating that there exists a vector w such that

$$\boxed{\sup_{c \in \mathcal{C}} c^{\mathsf{T}} w \leq z_o^{\mathsf{T}} w} \tag{60.60}$$

In the case when $\mathcal{C}$ is closed and z_o is a point on its boundary, then the hyperplane would correspond to the tangent at z_o — see the illustration in Fig. 60.9.

Next consider two disjoint nonempty convex sets $\mathcal{X}$ and $\mathcal{Y}$ in $\mathbb{R}^M$. Then, the *separating hyperplane theorem* states that there exists a vector $w \in \mathbb{R}^M$ and a scalar θ such that:

$$x^{\mathsf{T}} w - \theta \leq 0, \quad \forall x \in \mathcal{X} \tag{60.61a}$$

$$x^{\mathsf{T}} w - \theta \geq 0, \quad \forall y \in \mathcal{Y} \tag{60.61b}$$

This is also equivalent to stating that there exists a vector w such that

$$\boxed{\sup_{x \in \mathcal{X}} x^{\mathsf{T}} w \leq \inf_{y \in \mathcal{Y}} y^{\mathsf{T}} w} \tag{60.62}$$

The inequalities in the above expressions cannot be made strict. Figure 60.10 illustrates a situation where two disjoint convex sets cannot be strictly separated. However, when at least one of the sets happens to be closed *and* bounded (also called compact), then there exist (w, θ) such that

$$x^{\mathsf{T}} w - \theta > 0, \quad \forall x \in \mathcal{X} \tag{60.63a}$$

$$x^{\mathsf{T}} w - \theta < 0, \quad \forall y \in \mathcal{Y} \tag{60.63b}$$

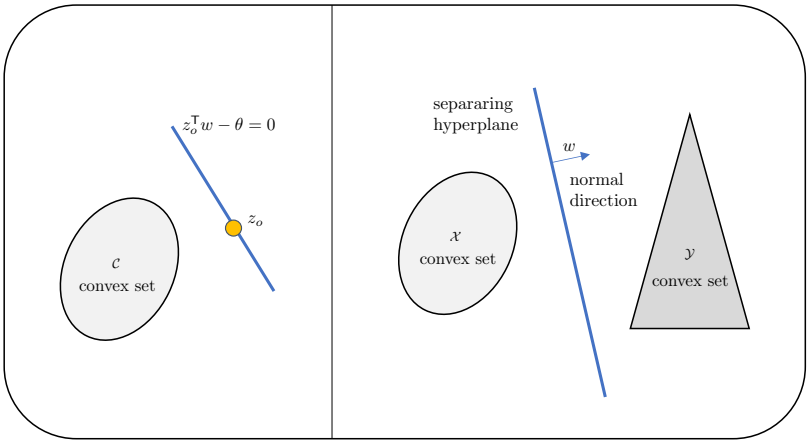


Figure 60.9 A supporting hyperplane on the left where the convex set appears on one side of it, and a separating hyperplane on the right where the convex sets are separated by it.

Proof of (60.62). Assuming the validity of the supporting hyperplane theorem, we can establish the separating hyperplane theorem as follows. Introduce the set $\mathcal{D} = \mathcal{X} - \mathcal{Y}$ where $d \in \mathcal{D}$ if, and only if, $d = x - y$ for some $x \in \mathcal{X}$ and $y \in \mathcal{Y}$. The origin $z = 0$ does not belong to $\mathcal{D}$ because otherwise it will require $x = y$ and we know that the sets $\mathcal{X}$ and $\mathcal{Y}$ do not share elements. We conclude from (60.60) that a vector $w \in \mathbb{R}^M$ should exist such that

$$\sup_{d \in \mathcal{D}} d^{\mathsf{T}} w \leq 0$$

which means that $x^{\mathsf{T}} w \leq y^{\mathsf{T}} w$ for any $(x, y) \in \mathcal{X} \times \mathcal{Y}$. It follows that (60.62) holds. ■

For further discussion and proofs, the reader is referred to Pettis (1956), Luenberger (1969), and Boyd and Vandenberghe (2004).

PROBLEMS

60.1 Refer to the Perceptron recursion (60.25). Is the performance of the algorithm affected if we set $\mu = 1$?

60.2 Can the Perceptron algorithm learn to implement the AND function? And what about the XOR function? We define these functions over four feature vectors in $\mathbb{R}^2$ as follows:

$$\text{AND} \rightarrow \begin{cases} h = [-1, -1]^{\mathsf{T}} \in \text{class -1} \\ h = [-1, +1]^{\mathsf{T}} \in \text{class -1} \\ h = [+1, -1]^{\mathsf{T}} \in \text{class -1} \\ h = [+1, +1]^{\mathsf{T}} \in \text{class } +1 \end{cases} \qquad \text{XOR} \rightarrow \begin{cases} h = [-1, -1]^{\mathsf{T}} \in \text{class -1} \\ h = [-1, +1]^{\mathsf{T}} \in \text{class } +1 \\ h = [+1, -1]^{\mathsf{T}} \in \text{class } +1 \\ h = [+1, +1]^{\mathsf{T}} \in \text{class -1} \end{cases}$$

60.3 Show that the Perceptron algorithm can learn to implement a NAND function? Consider then the other logical operations represented by NOT, AND, OR, NOR, XOR (exclusive OR), and XNOR (exclusive NOR). Show how each of these logical operations can be implemented by using solely NAND gates.

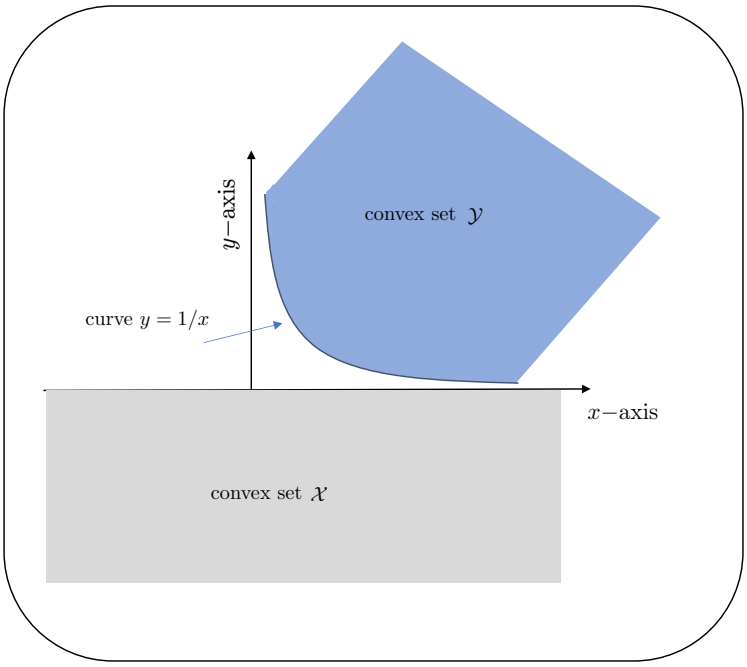


Figure 60.10 An example of two disjoint convex sets that cannot be strictly separated.

60.4 Consider a collection of $N-$linearly separable data pairs $\{\gamma(n), h_n\}$. Assume the offset parameter is zero. Show that linear separability is equivalent to the existence of a vector $w \in \mathbb{R}^M$ that satisfies $Hw \succeq \mathbb{1}_N$, where H is the $N \times M$ data matrix whose rows are $\gamma(n)h_n^{\mathsf{T}}$ and $\succeq$ denotes elementwise comparisons.

60.5 Consider two collections of vectors in $\mathbb{R}^M$ denoted by $\mathcal{H} = \{h_1, h_2, \ldots, h_N\}$ and $\mathcal{X} = \{x_1, x_2, \ldots, x_L\}$. We say that these sets are linearly separable if there exist $w^\star \in \mathbb{R}^M$ and $\theta^\star \in \mathbb{R}$ such that $h_n^{\mathsf{T}} w^\star > \theta^\star$ for all vectors in set $\mathcal{H}$ and $x_n^{\mathsf{T}} w^\star < \theta^\star$ for all vectors in set $\mathcal{X}$. Show more strongly that the two sets $\mathcal{H}$ and $\mathcal{X}$ are linearly separable if, and only if, there exist $z^\star \in \mathbb{R}^M$ and $\alpha^\star \in \mathbb{R}$ such that $h_n^{\mathsf{T}} z^\star - \alpha^\star \geq 1$ for all vectors in $\mathcal{H}$ and $x_n^{\mathsf{T}} z^\star - \alpha^\star \leq -1$ for all vectors in $\mathcal{X}$.

60.6 Continuing with Prob. 60.5, we show that checking linear separability of two sets can be reduced to solving a *linear program*. Show that two sets $\mathcal{H}$ and $\mathcal{X}$ are linearly separable if, and only if, the optimal value for the following linear program is zero:

$$\min_{\{z,\alpha,a,b\}} \ \frac{1}{N}\mathbb{1}_N^{\mathsf{T}} a + \frac{1}{L}\mathbb{1}_L^{\mathsf{T}} b, \ \text{where } z \in \mathbb{R}^M, \alpha \in \mathbb{R}, \ a \in \mathbb{R}^N, \ b \in \mathbb{R}^L$$

$$\text{subject to } \begin{cases} a(n) \geq -h_n^{\mathsf{T}} z + \alpha + 1, & n = 1, 2, \ldots, N \\ b(\ell) \ \geq \ \ x_\ell^{\mathsf{T}} z - \alpha + 1, & \ell \ = 1, 2, \ldots, L \\ a(n) \geq 0, & n = 1, 2, \ldots, N \\ b(\ell) \ \geq 0, & \ell \ = 1, 2, \ldots, L \end{cases}$$

Show further that if $\{z^\star, \alpha^\star, a^\star, b^\star\}$ is an optimal solution, then $g(f) = f^{\mathsf{T}} z^\star - \alpha^\star$ is a separating hyperplane for the two sets, where f denotes a generic feature vector. *Remark.* The reader may refer to Smith (1968) and Bennett and Mangasarian (1992) for a related discussion.

60.7 Consider a collection of N linearly separable data pairs $\{\gamma(n), h_n\}$ where $\gamma(n) \in$

$\{-1,+1\}$ denotes the label and $h_n \in \mathbb{R}^M$ is the corresponding feature vector. We assume feature vectors have already been extended according to (60.20). We wish to determine a separating hyperplane w such that $\gamma(n)h_n^{\mathsf{T}}w > 0$. We motivated the Perceptron recursion in the body of the chapter as one solution method. Here, we motivate a second *relaxation* method based on using the alternating projection algorithm from Sec. 12.6. Introduce the N halfspaces $\mathcal{H}_n = \{w \,|\, -\gamma(n)h_n^{\mathsf{T}}w < 0\}$, one for each data pair $(\gamma(n), h_n)$. We are then faced with the problem of solving N linear inequalities and finding a point $w^\star$ in the intersection of these halfspaces. Use the result of Prob. 9.5 to show that the alternating projecting method motivates the following recursion:

$$w_n = w_{n-1} + \frac{\gamma(n)h_n}{\|h_n\|^2}\,\max\Big\{0, -\gamma(n)h_n^{\mathsf{T}}w_{n-1}\Big\}$$

How is this method different from the unregularized Perceptron recursion? *Remark.* The above recursion is known as a relaxation method for solving a set of linear inequalities; it was introduced by Agmon (1954) and Motzkin and Schoenberg (1954) in back-to-back papers in the same journal issue — see also Eremin (1965). A footnote on the first page of Agmon (1954) acknowledges that the idea of the algorithm was communicated to the author by the first author of Motzkin and Schoenberg (1954).

60.8 Consider a collection of N data points $\{\gamma(n), h_n\}$ and refer to the Perceptron recursion (60.25). Let $w_1^\star$ denote the separating hyperplane that is obtained by running the recursion over this data. Now assume we replace each $\gamma(n)$ by $-\gamma(n)$; that is, we switch the labeling of the classes: class $+1$ becomes -1 and vice-versa. We run Perceptron again on this modified data and obtain $w_2^\star$. How are $w_1^\star$ and $w_2^\star$ related to each other? Is Perceptron sensitive to how we label the classes?

60.9 Refer to the Perceptron recursion (60.25). Introduce the variable $\boldsymbol{d}(n)$ defined as follows: $\boldsymbol{d}(n) = +1$ if $\boldsymbol{\gamma}(n) = +1$ and $\boldsymbol{d}(n) = 0$ if $\boldsymbol{\gamma}(n) = -1$. Introduce also the hard-threshold function:

$$g(x) \;\triangleq\; \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

Show that recursion (60.25) can be re-worked into the following form:

$$\begin{aligned} \boldsymbol{e}(n) &= \boldsymbol{d}(n) - g(\boldsymbol{h}_n^{\mathsf{T}}\boldsymbol{w}_{n-1}) \\ \boldsymbol{w}_n &= \boldsymbol{w}_{n-1} \;+\; \mu\boldsymbol{h}_n\boldsymbol{e}(n) \end{aligned}$$

60.10 Refer to Oja rule (60.56). Explain that this rule is maximizing $\mathbb{E}\,(\boldsymbol{h}_n^{\mathsf{T}}w)^2$ subject to $\|w\|^2 = 1$.

60.11 Let $\boldsymbol{\gamma}$ denote a generic binary random variable that assumes the values ± 1, and let $\boldsymbol{h}$ denote an $M \times 1$ random (feature) vector. Consider the following *regularized* exponential risk function:

$$P(w) \;\triangleq\; \rho\|w\|^2 \;+\; \mathbb{E}\,\Big\{e^{-\boldsymbol{\gamma}\boldsymbol{h}^{\mathsf{T}}w}\Big\}$$

where $\rho > 0$ is a regularization parameter. Derive a stochastic-gradient algorithm for the minimization of $P(w)$. How does the algorithm compare to Perceptron learning?

60.12 Establish the equality to 2^N in (60.77).

60.13 Conclude from (60.65) that when $N = 2(M+1)$, then the number of linearly separable dichotomies of the feature vectors $\{h_n \in \mathbb{R}^M\}$ is given by $\mathcal{S}(M,N) = 2^{2M+1}$. In other words, show that in this case only half of the 2^N possible dichotomies are linearly separable.

60.14 Assume the number of feature vectors is fixed at N while their dimension is allowed to increase from $m = 1$ up to $m = M-1$. Conclude from (60.65) that the number of linearly separable dichotomies, $\mathcal{S}(m,N)$, increases monotonically with m from the value $\mathcal{S}(1,N) = 2N$ up to the value $\mathcal{S}(M-1,N) = 2^N$.

60.15 Refer to result (60.65).

(a) Use it to bound the number of linearly separable Boolean functions in $M-$dimensions as follows:

$$\mathcal{S}(M, 2^M) \;\leq\; 2\sum_{m=0}^{M}\binom{2^M-1}{m}$$

Is this bound consistent with future Sauer lemma (64.86)?

(b) Use an argument similar to (64.102) to establish that for any $M \geq L$:

$$\sum_{n=0}^{L}\binom{M}{m} < \left(\frac{Me}{L}\right)^L$$

(c) Combine the results of parts (a) and (b) to establish (60.85).

60.16 Refer to the probability expression (60.78), which evaluates the likelihood that a randomly selected dichotomy of N feature vectors in $\mathbb{R}^M$ is linearly separable.

(a) Plot $\mathbb{P}(M, N)$ against the ratio $N/(M+1)$.

(b) What are the values of $\mathbb{P}(M, N)$ when $N = 2(M+1)$ and $N \leq M+1$.

(c) Establish the limits (60.79)–(60.80).

(d) Establish (60.81).

60.17 Consider a unit-edge hypercube in $M-$dimensions, with one vertex lying at the origin. The hypercube has 2^M edges. Let the feature vectors $\{h_m\}$ correspond to the locations of these vertices. Show that no $2M$ vertices in general position exist.

60.A COUNTING THEOREM

The concept of linearly separable data is paramount in the study of binary classification problems. We indicated in (60.2) that a collection of feature vectors $\{h_n \in \mathbb{R}^M\}$ is linearly separable if at least one hyperplane, $w^\star \in \mathbb{R}^M$, can be determined that separates the data into two classes with one class lying on one side of the plane and the other class lying on the other side of the plane, namely,

$$\begin{cases} h_n^{\mathsf{T}} w^\star < 0, \text{ whenever } h_n \in \text{class } -1 \\ h_n^{\mathsf{T}} w^\star > 0, \text{ whenever } h_n \in \text{class } +1 \end{cases} \tag{60.64}$$

In our discussion in this appendix we will assume that the feature data and the weight vector for the separating hyperplane have been extended according to (60.20) so that there is no need to account for the offset parameter separately. Now, given an arbitrary collection of N feature vectors $h_n \in \mathbb{R}^M$, there are 2^N possibilities for assigning each one of them to a class $\gamma(n) \in \{\pm 1\}$. Each of these possibilities is referred to as a *dichotomy*. In general, not all dichotomies will be linearly separable. We refer to Fig. 60.11 to illustrate this concept.

The figure shows $N = 3$ feature vectors in $\mathbb{R}^2$ and all eight possible dichotomies (i.e., label assignments). For example, in the leftmost box in the top row, we show two feature vectors assigned to $+1$ (represented by the plus sign) and one feature vector assigned to -1 (represented by the minus sign). In this same box, we show a line that can be used to separate both classes. Similarly for the other remaining 7 boxes in the figure. It is seen in this example that all eight dichotomies are linearly separable. In the rightmost box, we consider another situation involving $N = 4$ feature vectors in $\mathbb{R}^2$ and show one particular dichotomy with two features assigned to $+1$ and two other features assigned to -1. In this case, the dichotomy is not linearly separable. We expand on this situation in Fig. 60.12, which lists all 16 possible dichotomies for $N = 4$ feature vectors. Each circle represents an assignment to class $+1$ and each square represents an assignment to class -1. The two boxes marked with background color correspond

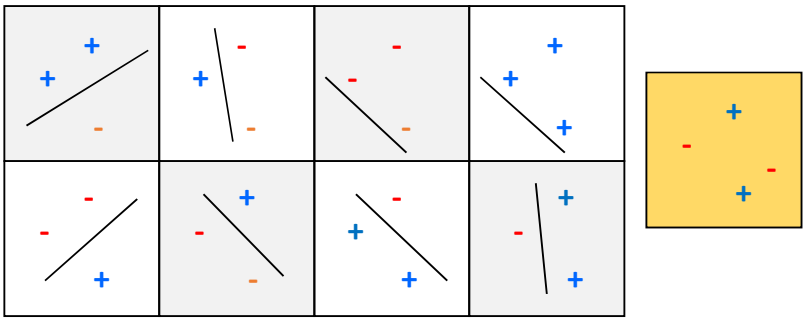


Figure 60.11 The eight squares on the left show all possible assignments of the *same* three feature vectors in $\mathbb{R}^2$. In each case, a line exists that separates the classes ± 1 from each other. We therefore say that the three feature vectors in this example are separable by linear classifiers. In contrast, the figure on the right shows four feature vectors in the same space $\mathbb{R}^2$ and an assignment of classes that cannot be separated by a linear classifier.

to dichotomies that are not linearly separable. It is seen from the figure that there are 14 out of 16 dichotomies that are linearly separable.

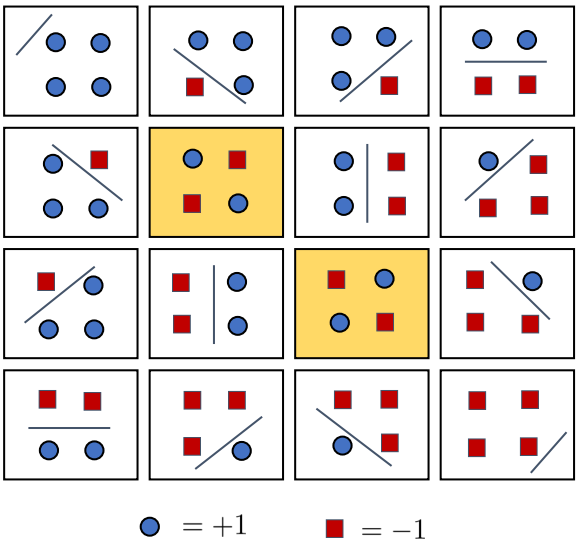


Figure 60.12 Given four ($N = 4$) feature vectors, there are $2^4 = 16$ possible dichotomies shown in the figure. Each circle represents assignment to class $+1$, while each square represents assignments to class -1. The marked boxes with background color correspond to the two dichotomies that are not linearly separable.

One useful question in the study of binary classification problems is the following. Given N feature vectors $\{h_n\}$ in $M-$dimensional space, how many of the 2^N dichotomies can be expected to be linearly separable? This is a classical problem in

combinatorial geometry and has been answered elegantly by Cover (1965); a couple of other works from the early 1950s and 1960s with similar conclusions are mentioned in Cover (1965) including an earlier proof technique by Schlafli (1950, pp. 209–212). The counting theorem that we describe below can be viewed as an early precursor to a famous inequality known as Sauer lemma, and which we establish later in Appendix 64.B — see (64.86). In the terminology of that appendix, the number of linearly separable dichotomies is also called the *shatter coefficient*. We denote this number by the notation $\mathcal{S}(M, N)$ where N is the number of feature vectors and M is the dimension of the feature space (and also the size of the parameter space that defines the classifier). Although we are focusing here on *linear* classifiers, we hasten to add that the shatter coefficient can be defined for other classes of classifiers as well; for this reason, in the future Appendix 64.B we will use instead the more general notation $\mathcal{S}(\mathcal{C}, N)$ to refer to the shatter coefficient, where the symbol $\mathcal{C}$ refers to the class of classifiers under consideration (linear or otherwise). The counting theorem stated further ahead is specific to linear classifiers, in which case it is justifiable to replace $\mathcal{C}$ by the dimension M of the parameter space, w. The statement of the counting theorem requires the notion of points in *general position*.

(Definition of points in general position). *Consider N column vectors $\{h_n\}$ in $M-$dimensional space, $h_n \in \mathbb{R}^M$. The N points are said to be in general position if no subset of $M+1$ vectors lies in an $(M-1)-$dimensional hyperplane. We also say that the points are in general position if every subset of M or fewer vectors is linearly independent.*

This situation is illustrated in Fig. 60.13. The plot on the left shows $N = 5$ feature vectors in $\mathbb{R}^2$ (for which $M = 2$). These vectors are not in general position because 3 vectors happen to lie on the same line. This example shows that four points in $\mathbb{R}^3$ are in general position if no three of them lie on the same line.

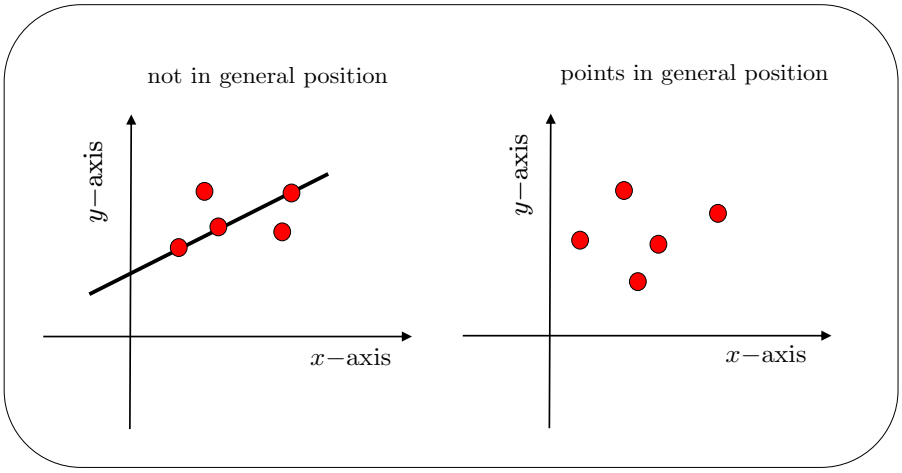


Figure 60.13 The plot on the left shows $N = 5$ feature vectors in $\mathbb{R}^2$ (for which $M = 2$). These vectors are not in general position because 3 vectors happen to lie on the same line.

We are now ready to state the counting theorem and prove it following Cover (1965). Observe that the theorem provides an exact count for the number of linearly separable dichotomies. It is not generally possible to provide such an exact count for other classes

of classifiers. For these more general cases, the theorem will be replaced by future Sauer lemma (64.86), which provides an upper bound (rather than an equality) for the number of separable dichotomies — see future Appendix 64.C on the Vapnik-Chervonenkis bound.

Counting theorem (Cover (1965)). *Consider the class of linear classifiers defined by* $\mathcal{C} = \{\text{sign}(h^{\mathsf{T}}w)\}$, *where* $h \in \mathbb{R}^M$ *denotes feature vectors and the free parameter* $w \in \mathbb{R}^M$ *defines the hyperplane. Feature vectors are assigned to classes* $+1$ *or* -1 *depending on the sign of the inner product* $h^{\mathsf{T}}w$. *Consider a collection of* N *feature vectors,* $\{h_n\}$, *in general position in* $\mathbb{R}^M$. *It holds that the number of linearly separable dichotomies, from among the* 2^N *possible dichotomies, is given by*

$$\mathcal{S}(M,N) = \begin{cases} 2\displaystyle\sum_{m=0}^{M}\binom{N-1}{m}, & \text{when } N > M+1 \\ 2^N, & \text{when } N \le M+1 \end{cases} \tag{60.65}$$

Proof: Starting with the N feature vectors $\{h_n\}$ in $\mathbb{R}^M$, we let $\mathcal{S}(M,N)$ denote the number of linearly separable dichotomies for this set of generally-positioned points. Next, we enlarge the set to $N+1$ points by adding a new feature vector, h_{N+1}, such that the new expanded feature set continues to have general position. We similarly let $\mathcal{S}(M,N+1)$ denote the number of linearly separable dichotomies for this new set. The argument that follows determines a relation between $\mathcal{S}(M,N)$ and $\mathcal{S}(M,N+1)$.

Let w be one of the linear classifiers that generates one of the dichotomies for the initial feature set $\{h_n\}$ of size N. Under this classifier, a feature vector h_n would be mapped to the label:

$$\gamma(n) = \text{sign}(h_n^{\mathsf{T}}w) \tag{60.66}$$

The value of $\gamma(n)$ is either $+1$ or -1. Thus, the hyperplane w generates the following dichotomy for the N feature vectors:

$$[\,\gamma(1),\gamma(2),\ldots,\gamma(N)\,], \quad \gamma(n) \in \{+1,-1\} \tag{60.67}$$

When this same hyperplane is applied to the additional feature h_{N+1}, its will generate some label denoted by

$$\gamma(N+1) = \text{sign}(h_{N+1}^{\mathsf{T}}w) \tag{60.68}$$

The value of this label is again either $+1$ or -1. In this way, the hyperplane w leads to the following dichotomy over the expanded set:

$$\Big[\gamma(1),\gamma(2),\ldots,\gamma(N),\,\underline{\gamma(N+1)}\Big] \tag{60.69}$$

We therefore find that for every linear dichotomy defined over the original N feature vectors $\{h_n\}$, we can associate at least one dichotomy over the expanded feature set of size $N+1$. The analysis so far shows that $\mathcal{S}(M,N+1)$ is at least as large as $\mathcal{S}(M,N)$:

$$\mathcal{S}(M,N+1) \;\ge\; \mathcal{S}(M,N) \tag{60.70}$$

Let us verify next that it is actually possible to generate more dichotomies over the $N+1$ feature vectors than the $\mathcal{S}(M,N)$ dichotomies generated over the smaller set. The argument depends on whether we can find a separating hyperplane w from the original set that passes through h_{N+1} or not:

(a) Assume first that there exists a hyperplane w from the set that generates the dichotomies for the original N feature vectors with the following property: the hyperplane passes through the added point h_{N+1}. In this case, we can perturb

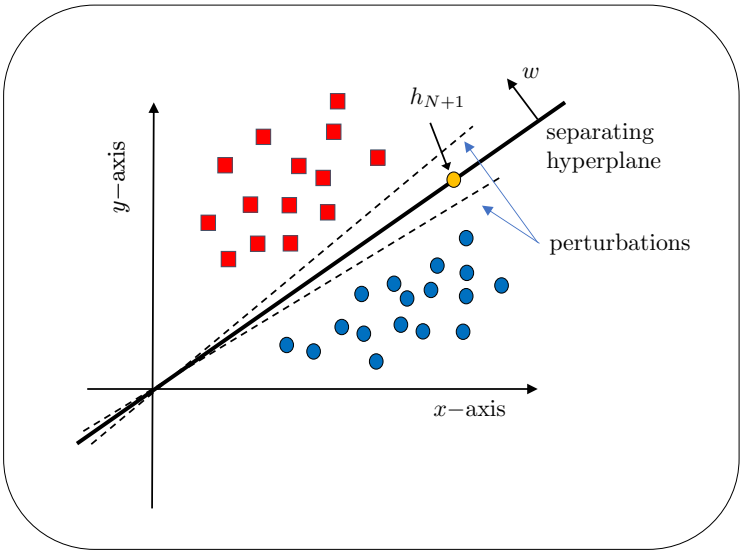


Figure 60.14 The plot shows one dichotomy for the N features vectors $\{h_n\}$ with a separating hyperplane w that passes through the new feature h_{N+1}. By perturbing this hyperplane slightly to one side or the other, the feature h_{N+1} can end up with label $+1$ or -1.

this hyperplane by an infinitesimal amount and have h_{N+1} appear on one side or the other of the plane, with the plane still separating the original N feature vectors — see Fig. 60.14. It follows in this case that, for each separating w for the original feature vectors, we are able to generate two dichotomies for the expanded $(N+1)-$long set (and not just one as above), with the label for h_{N+1} being either $+1$ or -1:

$$[\gamma(1),\gamma(2),\ldots,\gamma(N),\underline{+1}], \quad [\gamma(1),\gamma(2),\ldots,\gamma(N),\underline{-1}] \tag{60.71}$$

This argument indicates that $\mathcal{S}(M,N+1)$ will be larger than $\mathcal{S}(M,N)$ and we write

$$\mathcal{S}(M,N+1) \;=\; \mathcal{S}(M,N) \;+\; \Delta \tag{60.72}$$

for some positive number Δ to be determined.

(b) Assume, on the other hand, that there is no hyperplane from the $\mathcal{S}(M,N)$ dichotomies for the original N feature vectors that passes through h_{N+1}. Then, in this case, the point h_{N+1} would always lie on one side of all the hyperplanes for the original dichotomies. As a result, only one dichotomy over the $N+1$ features is possible, as explained earlier, and not two dichotomies as in part (a).

We therefore need to determine Δ. By definition, its value is equal to the number of dichotomies of the original N feature vectors with the *constraint* that the separating hyperplanes should pass through h_{N+1}. By restricting the separating hyperplanes to pass through a particular point, we are in effect reducing the dimension (or degrees of freedom) of the problem from M down to $M-1$. Therefore, it holds that $\Delta = \mathcal{S}(M-1,N)$ and we arrive at the relation

$$\mathcal{S}(M,N+1) \;=\; \mathcal{S}(M,N) \;+\; \mathcal{S}(M-1,N) \tag{60.73}$$

We can now use this relation to establish (60.65) by induction. We assume result (60.65)

holds for (M, N) and establish a similar form for $(M, N+1)$. To begin with, note that the relation holds for $N = 1$ since it gives $\mathcal{S}(M, 1) = 2$ — see (60.77), and we know that for a single feature vector in $M-$dimensional space there are only two possible dichotomies. Note also that relation (60.65) holds for $M = 1$ since it gives $\mathcal{S}(1, N) = 2N$, and we know that there are $2N$ dichotomies for N generally-positioned points on a line.

Next, using (60.73) and the assumed induction form (60.65) we have

$$\begin{aligned}
\mathcal{S}(M, N+1) &= 2\sum_{m=0}^{M}\binom{N-1}{m} + 2\sum_{m=0}^{M-1}\binom{N-1}{m} \\
&= 2\sum_{m=0}^{M}\binom{N-1}{m} + 2\sum_{m'=1}^{M}\binom{N-1}{m'-1}, \quad m \leftarrow m'-1 \\
&= 2\sum_{m=0}^{M}\binom{N-1}{m} + 2\sum_{m=1}^{M}\binom{N-1}{m-1}, \quad m' \leftarrow m \\
&\overset{(a)}{=} 2\sum_{m=0}^{M}\binom{N-1}{m} + 2\sum_{m=0}^{M}\binom{N-1}{m-1} \\
&= 2\sum_{m=0}^{M}\left\{\binom{N-1}{m} + \binom{N-1}{m-1}\right\} \\
&\overset{(b)}{=} 2\sum_{m=0}^{M}\binom{N}{m}
\end{aligned} \tag{60.74}$$

as expected, where step (a) uses the property

$$\binom{N-1}{k} = 0, \quad \text{when } k < 0 \tag{60.75}$$

and step (b) uses the equality

$$\binom{N}{m} = \binom{N-1}{m} + \binom{N-1}{m-1} \tag{60.76}$$

Relation (60.65) is valid as long as the value of m within the combinatorial expression does not exceed $N-1$. This is satisfied whenever $M < N-1$ or, equivalently, $N > M+1$. On the other hand, when $N \leq M+1$, we can replace the upper limit M in the summation by $N-1$ and write instead

$$\mathcal{S}(M, N) = 2\sum_{m=0}^{N-1}\binom{N-1}{m} = 2^N, \quad \text{when } N \leq M+1 \tag{60.77}$$

This concludes the proof.

■

Now, given a collection of N feature vectors $h_n \in \mathbb{R}^M$ and assuming each of the 2^N possible dichotomies are equally likely to occur, we readily conclude from result (60.65) that the probability that a randomly selected dichotomy is linearly separable is captured by the expression:

$$\mathbb{P}(M, N) = \mathcal{S}(M, N)/2^N \tag{60.78}$$

This is a revealing expression and brings forth some useful properties. Problems 60.12–60.16 explore these properties and are motivated by the exposition and results from Cover (1965). In particular, the following useful conclusions are established in these problems:

(a) When $N \leq M+1$, each one of the 2^N possible dichotomies of the feature vectors $h_n \in \mathbb{R}^M$ are linearly separable.

(b) When $N = 2(M+1)$, only half of the 2^N possible dichotomies of the feature vectors $h_n \in \mathbb{R}^M$ is linearly separable.

(c) The value $N = 2(M+1)$ corresponds to a critical turning point for large dimensional problems. In particular, it holds for any small $\epsilon > 0$ that

$$\lim_{M\to\infty} \mathbb{P}\Big(M, (1+\epsilon)2(M+1)\Big) = 0 \tag{60.79}$$

$$\lim_{M\to\infty} \mathbb{P}\Big(M, (1-\epsilon)2(M+1)\Big) = 1 \tag{60.80}$$

Observe how at the cut-off point $N = 2(M+1)$ (i.e., for this many feature vectors), the probability of linear separation transitions sharply from one down to zero.

(d) These limiting results motivate introducing the notion of the *capacity* of the class of linear classifiers in $M-$dimensional space. The capacity is defined as the *largest* number C such that for any $N < (1-\epsilon)C$, a random dichotomy of size N in $\mathbb{R}^M$ is linearly separable with probability larger than $1-\delta$, for some small $\delta > 0$. It can be shown that, for M large enough,

$$C = 2(M+1) \tag{60.81}$$

That is, the capacity corresponds roughly to two random feature vectors per weight dimension.

60.B BOOLEAN FUNCTIONS

It is useful to comment on how the results from the previous appendix on linear separability relate to the (more complex) problem of counting the number of linearly separable dichotomies generated by *Boolean* functions. One key difference in relation to what we have discussed so far is that the entries of each h_n will now be restricted to assuming only the *binary* values 0 or 1. In this case, the feature vectors $\{h_n\}$ will generally violate the general position requirement, as illustrated by Prob. 60.17. Consequently, result (60.65) will not be applicable anymore. However, building on arguments from Furedi (1986), the work by Budinich (1991) shows that the probability expression (60.78) will continue to hold for $M \to \infty$. The expression would then provide the probability that a collection of N vertices in a large $M-$dimensional hypercube are linearly separable, as we proceed to clarify.

Let $f(a_1, a_2, \ldots, a_M) : \{0,1\}^M \to \{0,1\}$ denote a Boolean function defined over M binary arguments denoted by $\{a_m\}$. Each a_m can assume one of only two possible values, 0 or 1, and the function itself can only assume the values 0 or 1. We can interpret each realization of the $M-$dimensional vector $(a_1, a_2, \ldots, a_M)$ as representing the coordinates of some vertex of a hypercube in $M-$dimensions. This coordinate vector plays the role of a feature vector h_n in our previous notation. The class that this feature vector belongs to will be the value of the function $f(a_1, \ldots, a_M)$, written more compactly as $f(h_n)$:

$$f(h_n) : \ \{0,1\}^M \ \longrightarrow \ \{0,1\} \tag{60.82}$$

Note that we are denoting the classes by $\{0,1\}$. Now, an $M-$dimensional hypercube will have 2^M vertices. Each of these vertices can be assigned to class 0 or 1. There are a total of 2^{2^M} possible binary assignments for all vertices of the hypercube, i.e., there are a total of 2^{2^M} Boolean functions over M arguments. For any particular choice of the Boolean function, we let $\mathcal{V}_0$ denote the collection of vertices it assigns to class 0 and $\mathcal{V}_1$ the collection of vertices it assigns to class 1. The Boolean function will then be said to

be *linearly separable* if there exists at least one hyperplane in $\mathbb{R}^M$ that separates the vertices $\mathcal{V}_0$ and $\mathcal{V}_1$ from each other: one set of vertices would appear on one side of the hyperplane and the other set would appear on the other side, i.e., if there exists some $w^\star \in \mathbb{R}^M$ such that

$$f(h_n) = \begin{cases} 1, & \text{if } h_n^{\mathsf{T}} w^\star > 0, \text{ i.e., } h_n \in \mathcal{V}_1 \\ 0, & \text{if } h_n^{\mathsf{T}} w^\star < 0, \text{ i.e., } h_n \in \mathcal{V}_0 \end{cases} \tag{60.83}$$

This situation is illustrated in Fig. 60.15. The figure shows one realization of a Boolean function; vertices marked in blue are assigned the binary value one and vertices marked in yellow are assigned the binary value zero. It is seen in this example that the sets $\mathcal{V}_0$ and $\mathcal{V}_1$ are linearly separable. One example of a Boolean function that is not linearly separable is the XOR function defined as follows (where $M = 2$):

$$f(h_n) = a_1 \,\text{XOR}\, a_2 \;\rightarrow \begin{cases} (a_1, a_2) = (0,0) \;\longrightarrow\; f(a_1, a_2) = 0 \\ (a_1, a_2) = (0,1) \;\longrightarrow\; f(a_1, a_2) = 1 \\ (a_1, a_2) = (1,0) \;\longrightarrow\; f(a_1, a_2) = 1 \\ (a_1, a_2) = (1,1) \;\longrightarrow\; f(a_1, a_2) = 0 \end{cases} \tag{60.84}$$

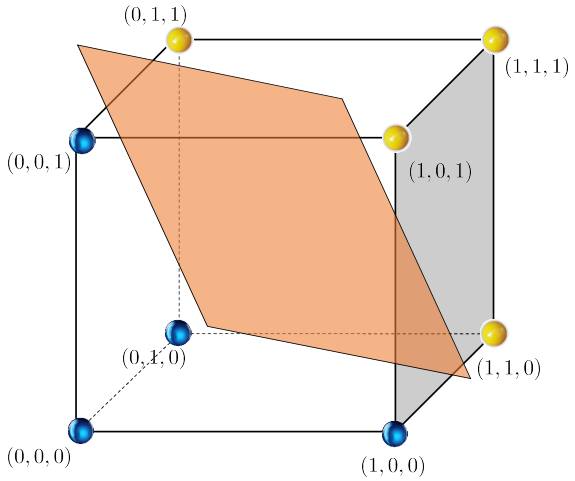


Figure 60.15 The figure shows one realization of a Boolean function; vertices marked in blue are assigned the binary value one and vertices marked in yellow are assigned the binary value zero. It is seen in this example that the sets $\mathcal{V}_0$ and $\mathcal{V}_1$ are linearly separable.

The question we would like to examine is to determine how many of the 2^{2^M} possible Boolean functions are linearly separable. We already know from the XOR example and from Fig. 60.12 that not all Boolean functions are linearly separable. For instance, consider the situation corresponding to $M = 2$ (Boolean functions with two arguments). Hypercubes in this space are squares with $2^2 = 4$ vertices. There are a total of $2^4 = 16$ possible assignments for these vertices. We know from the representation in Fig. 60.12 that there are only 14 linearly separable Boolean functions in this case.

More generally, there is no closed-from expression for the number of linearly separable Boolean functions for arbitrary values of M; this is in contrast to result (60.65). In the Boolean context, we have $N = 2^M$ (the number of feature vectors is the number

of vertices). Therefore, we will denote the number of linearly separable Boolean functions by $\mathcal{S}(M, 2^M)$. Although a closed-form expression for $\mathcal{S}(M, 2^M)$ does not exist, the work by Muroga (1971) provides a useful upper bound — see also the text by Peretto (1992), the volume edited by Smolensky, Mozer, and Rumelhart (1996), and the proof in Anthony (2001, pp. 37–38), as well as Prob. 60.15:

$$\mathcal{S}(M, 2^M) \leq 2^{M^2} \tag{60.85}$$

The following table provides some known values for the number of linearly separable Boolean functions up to $M = 8$ — see Muroga (1971).

Table 60.3 Number of linearly separable Boolean functions in $M-$dimensional space (up to $M = 8$) and the probability that the vertices of the unit-edge hypercube are linearly separable.

M	# Boolean functions $\left(2^{2^M}\right)$	# linearly separable Boolean functions	probability of separation
1	4	4	1
2	16	14	0.875
3	256	104	0.40625
4	65,356	1,882	0.02880
5	4,294,967,296	94,572	$\approx 2.2 \times 10^{-5}$
6	18,446,744,073,709,551,616	15,028,134	$\approx 8.1 \times 10^{-13}$
7	$\approx 3.4028 \times 10^{38}$	8,378,070,864	$\approx 2.5 \times 10^{-29}$
8	$\approx 1.1579 \times 10^{77}$	17,561,539,552,946	$\approx 1.5 \times 10^{-64}$

REFERENCES

Agmon, S. (1954), "The relaxation method for linear inequalities," *Canadian Journal of Mathematics*, vol. 6, no. 3, pp. 382–392.

Anthony, M. (2001), *Discrete Mathematics of Neural Networks*, SIAM, PA.

Bennett, K. P. and O. L. Mangasarian (1992), "Robust linear programming discrimination of two linearly inseparable sets," *Optimization Methods and Software*, vol. 1, pp. 23–34.

Block, H. D. (1961), "Analysis of Perceptrons," *Proc. West Joint. Computer Conf.*, vol. 19, pp. 281–289.

Block, H. D. (1962), "The Perceptron: A model for brain functioning I," *Reviews of Modern Physics*, vol. 34, no. 1, pp. 123–135.

Boyd, S. and L. Vandenberghe (2004), *Convex Optimization*, Cambridge University Press.

Budinich, M. (1991), "On linear separability of random subsets of hypercube vertices," *J. Phys. A: Math. Gen.*, vol. 24, pp. L211–L213.

Cover, T. M. (1965), "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition," *IEEE Trans. Electronic Computers*, vol. 14, pp. 326–334.

Duda, R. O. and P. E. Hart (1973), *Pattern Classification and Scene Analysis*, Wiley, NY.

Eremin, I. (1965), "The relaxation method of solving systems of inequalities with convex functions on the left sides," *Soviet Math. Dokl.*, vol. 6, pp. 219–222.

Furedi, Z. (1986), "Random polytopes in the d-dimensional cube," *Discrete Comput. Geometry*, vol. 1, pp. 315–319.

Gallant, S. I. (1986), "Optimal linear discriminants," *Proc. Eighth Int. Conf. Pattern Recognition*, pp. 849–852, Pans, France,
Gallant, S. I. (1990), "Perceptron-based learning algorithms," *IEEE Trans. Neural Networks*, vol. 1, no. 2, pp. 179–191.
Haykin, S. (1999), *Neural Networks: A Comprehensive Foundation*, Prentice Hall, NY.
Hebb, D. O. (1949), *The Organization of Behavior*, Wiley, NY.
Luenberger, D. G. (1969), *Optimization by Vector Space Methods*, Wiley, NY.
McCulloch, W. and W. Pitts (1943), "A logical calculus of ideas immanent in nervous activity," *Bull. Math. Biophysics*, vol. 5, no. 4, pp. 115–133.
Minkowski, H. (1911), *Gesammelte Abhandlungen*, Leipzig and Berlin.
Minsky, M. and S. Papert (1969), *Perceptrons*, MIT Press, Cambridge, MA. Expanded edition published in 1987.
Motzkin, T. and I. J. Schoenberg (1954), "The relaxation method for linear inequalities," *Canadian Journal of Mathematics*, vol. 6, no. 3, pp. 393–404.
Muroga, S. (1971), *Threshold Logic and Its Applications*, Wiley, NY.
Novikoff, A. (1962), "'On convergence proofs on perceptrons," *Proc. Symp. Mathematical Theory Automata*, pp. 615–622, Brooklyn, NY.
Oja, E. (1982), "Simplified neuron model as a principal component analyzer," *J. Math. Biology*, vol. 15, no. 3, pp. 267–273.
Oja, E. (1983), *Subspace Methods of Pattern Recognition*, Research Studies Press, NY.
Oja, E. (1992), "Principal components, minor components, and linear neural networks," *Neural Networks*, vol. 5, pp. 927–935.
Peretto, P. (1992), *An Introduction to the Modeling of Neural Networks*, Cambridge University Press.
Pettis, B. J. (1956), "Separation theorems for convex sets," *Mathematics Magazine*, vol. 29, no. 5, pp. 233–247.
Rosenblatt, F. (1957), *The Perceptron: A Perceiving and Recognizing Automaton*, Technical Report 85-460-1, Project PARA, Cornell Aeronautical Lab.
Rosenblatt, F. (1958), "The Perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408.
Rosenblatt, F. (1962), *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Spartan Books, Washington DC.
Schlafli, L. (1950), *Gesammelte Mathematische Abhandlungen I.*, Basel, Switzerland.
Siu, K.-Y., V. P. Roychowdhury, and T. Kailath (1995), *Discrete Neural Computation: A Theoretical Foundation*, Prentice Hall, NJ.
Smith, F. W. (1968), "Pattern classifier design by linear programming," *IEEE Trans. on Comput.*, vol. 17, no. 4, pp. 367–372.
Smolensky, P., M. C. Mozer, and D. E. Rumelhart, *Eds.* (1996), *Mathematical Perspectives on Neural Networks*, Lawrence Erlbaum Publishers.
Theodoridis, S. (2015), *Machine Learning: A Bayesian and Optimization Perspective*, Academic Press.
Widrow, B. and M. A. Lehr (1990),"30 years of adaptive neural networks: Perceptron, Madaline, and backpropagation," *Proc. IEEE*, vol. 78, no. 9, pp. 1415–1442.

61 SUPPORT VECTOR MACHINES

When the training data $\{\gamma(n), h_n\}$ is linearly separable, there will exist many separating hyperplanes that can discriminate the data into two classes. Some of the techniques we described in the previous chapters, such as logistic regression and Perceptron , are able to find such separating hyperplanes. However, in general, there are many others. For example, if we refer to the earlier Fig. 60.1, we observe that the slopes of the separating lines in the figure can be adjusted, with the lines tilted further in one direction or the other, and we would still obtain correct classification for the same training data. For each valid choice of a separating hyperplane, $w^\star$, there will exist some feature vector h in the training set that is closest to the hyperplane. We indicated in the previous chapter that the distance of this closest point to the hyperplane is called the *margin* and was denoted by $m(w^\star)$. We illustrate this situation again in Fig. 61.1. In this chapter, we describe the support vector machine (SVM) technique, whose purpose is to find the hyperplane $w^\star$ with the *largest* possible margin, so that the training data will be the farthest away from it compared to other separating hyperplanes. Doing so adds a degree of robustness, as well as a desirable safety margin, to the operation of the classifier. We will consider two formulations of SVM: one is referred to as hard-margin SVM and the other is soft-margin SVM. Both techniques are again examples of *deterministic* methods, which operate directly on data realizations $\{\gamma(n), h_n\}$ without assuming explicitly any form for the underlying conditional or joint pdfs of the random variables $(\boldsymbol{\gamma}, \boldsymbol{h})$, as was the case, for example, with logistic regression and linear discriminant analysis (LDA).

61.1 SVM EMPIRICAL RISK

The SVM formulation can be motivated by following geometric arguments. Let $(w^\star, \theta^\star)$ denote the parameters (weight vector and scalar offset) of some generic separating hyperplane for a collection of N *linearly separable* training points $\{\gamma(n), h_n\}$, where $\gamma(n) \in \{\pm 1\}$ is the label associated with feature vector $h_n \in \mathbb{R}^M$. Some feature vectors in this set will be closer to the hyperplane $(w^\star, \theta^\star)$ than other feature vectors. Let $(\gamma(n^\star), h_{n^\star})$, with index $n^\star$, denote one of the data points in the set that is closest to $(w^\star, \theta^\star)$. This situation is illustrated in

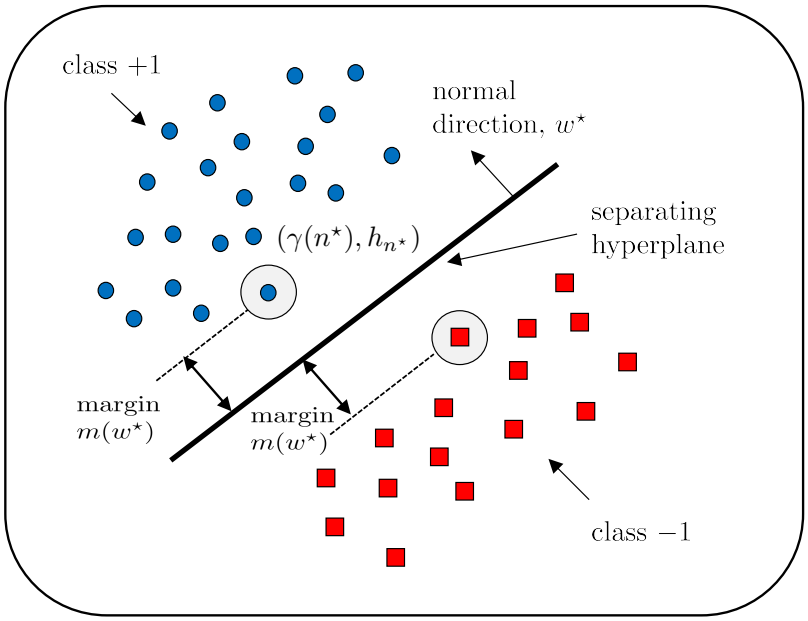


Figure 61.1 The figure shows one separating hyperplane and the two closest points from the training data to it; the points are highlighted inside a circle. The distance from these points to the hyperplane is called the *margin*. Other separating hyperplanes will have their own margins.

Fig. 61.1; the figure further illustrates the possibility that there can also exist points in the other class at the same closest distance from the hyperplane. Since all points are correctly classified by $(w^\star, \theta^\star)$, then using expression (60.10) we conclude that the margin is given by:

$$m(w^\star) = \gamma(n^\star)\left(h_{n^\star}^{\mathsf{T}} w^\star - \theta^\star\right)\frac{1}{\|w^\star\|} \tag{61.1}$$

We are free to scale $(w^\star, \theta^\star)$ without altering the hyperplane $h^{\mathsf{T}} w^\star - \theta^\star = 0$. Thus, assume the parameters $(w^\star, \theta^\star)$ are scaled by the same value to attain the normalization:

$$\gamma(n^\star)(h_{n^\star}^{\mathsf{T}} w^\star - \theta^\star) \;=\; 1 \tag{61.2}$$

In this case, the margin associated with the scaled $(w^\star, \theta^\star)$ becomes

$$m(w^\star) = \frac{1}{\|w^\star\|} \tag{61.3}$$

which is inversely proportional to $\|w^\star\|$. It follows that maximizing $m(w^\star)$ is equivalent to minimizing $\frac{1}{2}\|w^\star\|^2$ (the scaling by $1/2$ is added for convenience).

Hard-margin version

Motivated by these considerations, we formulate the design problem:

$$(w^\star, \theta^\star) = \underset{w \in \mathbb{R}^M, \theta \in \mathbb{R}}{\operatorname{argmin}} \ \frac{1}{2}\|w\|^2 \tag{61.4a}$$

$$\text{subject to } \ \gamma(n)(h_n^{\mathsf{T}} w - \theta) \geq 1, \ \ n = 0, 1, \ldots, N-1 \tag{61.4b}$$

This formulation helps enforce three properties:

(a) (**Correct classifications**) First, it enforces that all training data points are correctly classified by the resulting classifier $(w^\star, \theta^\star)$. This is because the predictor $\widehat{\gamma}(n) = h_n^{\mathsf{T}} w^\star - \theta^\star$ and the true label $\gamma(n)$ will have the same sign by (61.4b).

(b) (**Sufficient distance away from hyperplane**) Second, all training points will be sufficiently away from the separating hyperplane $(w^\star, \theta^\star)$, at a distance that is at least equal to $1/\|w^\star\|$. This is because, using expression (60.10), the distance from any training feature vector h_n to the separating hyperplane will satisfy

$$\text{distance} = \gamma(n)\left(h_n^{\mathsf{T}} w^\star - \theta^\star\right) \frac{1}{\|w^\star\|} \overset{(61.4b)}{\geq} \frac{1}{\|w^\star\|} \tag{61.5}$$

(c) (**Margin attained**) Third, there should exist an index $n^\star$ that satisfies (61.4b) with *equality*. This conclusion can be verified by contradiction. Assume the solution $(w^\star, \theta^\star)$ leads to a strict inequality for all training points, namely, $\gamma(n)(h_n^{\mathsf{T}} w^\star - \theta^\star) > 1$ for all $0 \leq n \leq N-1$. Let $n^\star$ denote the index with smallest value for the product $\gamma(n)(h_n^{\mathsf{T}} w^\star - \theta^\star)$, i.e.,

$$n^\star = \underset{0 \leq n \leq N-1}{\operatorname{argmin}} \ \left\{\gamma(n)(h_n^{\mathsf{T}} w^\star - \theta^\star)\right\} \tag{61.6}$$

and denote the corresponding value by

$$\delta^\star \ \triangleq \ \gamma(n^\star)\left(h_{n^\star}^{\mathsf{T}} w^\star - \theta^\star\right) \tag{61.7}$$

By assumption, we have $\delta^\star > 1$. We scale $(w^\star, \theta^\star)$ down by $\delta^\star$ and replace them by

$$w^\star \longleftarrow w^\star/\delta^\star, \quad \theta^\star \longleftarrow \theta^\star/\delta^\star \tag{61.8}$$

The scaled $(w^\star, \theta^\star)$ continues to be a separating hyperplane that satisfies the constraint (61.4b) for all n. However, the scaled $w^\star$ has a smaller norm than the original $w^\star$ since $\delta^\star > 1$, which contradicts (61.4a). We conclude that there must exist an index $n^\star$ that satisfies $\gamma(n^\star)(h_{n^\star}^{\mathsf{T}} w^\star - \theta^\star) = 1$. In view of expression (61.1), the feature vector $h_{n^\star}$ attains the margin $m(w^\star) = 1/\|w^\star\|$.

Once a separating hyperplane $(w^\star, \theta^\star)$ is determined by solving problem (61.4a)–(61.4b), we may encounter three situations depending on how a training point $(\gamma(n), h_n)$ is positioned relative to the hyperplane:

$$\begin{cases} \gamma(n)\left(h_n^{\mathsf{T}} w^\star - \theta^\star\right) > 1 \;\rightarrow\; \text{point } (\gamma(n), h_n) \text{ exceeds the margin} \\ \gamma(n)\left(h_n^{\mathsf{T}} w^\star - \theta^\star\right) = 1 \;\rightarrow\; \text{point } (\gamma(n), h_n) \text{ meets the margin} \\ \gamma(n)\left(h_n^{\mathsf{T}} w^\star - \theta^\star\right) < 1 \;\rightarrow\; \text{point } (\gamma(n), h_n) \text{ violates the margin} \end{cases} \tag{61.9}$$

In the first case, the distance from h_n to the separating hyperplane will be larger than $1/\|w^\star\|$ and, therefore, the point $(\gamma(n), h_n)$ will be farther away from the separating hyperplane than the margin. In the second case, we say that the training point $(\gamma(n), h_n)$ meets the margin since the distance from h_n to the separating hyperplane $(w^\star, \theta^\star)$ will be $1/\|w^\star\|$, which is the value of the margin. In the third case, the point h_n will be closer to the hyperplane than the margin. Obviously, as was just proven under items (a)–(c), when problem (61.4a)–(61.4b) admits a solution $(w^\star, \theta^\star)$, then all points $\{\gamma(n), h_n\}$ will either meet the margin or exceed it and the violation in the third case will not occur; this scenario will only arise when we study the *soft-margin* SVM further ahead. The solution $(w^\star, \theta^\star)$ to (61.4a)–(61.4b) is called the *hard-margin* SVM solution because we are requiring the training data to be linearly separable *and* to have a distance of at least $1/\|w^\star\|$ away from the separating hyperplane (i.e., to exceed the margin). We will refer to all points $(\gamma(n), h_n)$ that meet or violate the margin as *support vectors*:

$$\boxed{(\gamma(n), h_n) \text{ is a support vector } \iff \gamma(n)(h_n^{\mathsf{T}} w^\star - \theta^\star) \leq 1} \tag{61.10}$$

The presence of these vectors is the reason for the name "support vector machine." We will explain in a later section, using duality arguments, that the solution to the SVM problem is exclusively defined by these support vectors — see future expression (61.45). In the hard-margin SVM formulation under discussion, support vectors will only consist of points $(\gamma(n), h_n)$ that meet the margin with *equality* sign in (61.10). However, as we will see in the sequel, support vectors $(\gamma(n), h_n)$ will exist under soft-margin SVM for which strict inequality holds in (61.10).

Soft-margin version

We formulate next a more relaxed version of problem (61.4a)–(61.4b), leading to *soft-margin* SVM, in order to accommodate situations where the data points are not fully linearly separable or when outliers may be present. Outliers can perturb the choice of the separating hyperplane in a significant manner and push it closer to one class or the other if one insists on a hard-margin design. This scenario is illustrated in Fig. 61.2. An outlier feature vector is highlighted by the surrounding circle; its presence results in a separating hyperplane (the solid line) with a smaller margin compared to the original dashed hyperplane (in dahsed line) obtained in the absence of the outlier.

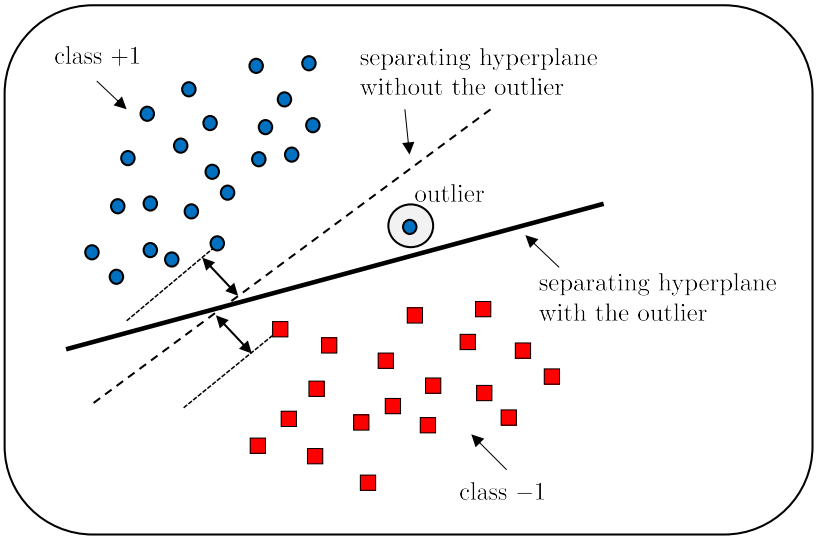


Figure 61.2 An outlier is indicated by the surrounding circle; its presence results in a separating hyperplane with a smaller margin compared to the original (dashed) hyperplane in the absence of the outlier. Soft-margin SVM reduces the influence of outliers on the selection of the separating hyperplane and leads to solutions that approach the dashed line.

Soft-margin SVM helps reduce the influence of outliers on the selection of the separating hyperplane. It continues to seek a hyperplane with the largest possible margin but will allow a small number of the data points to *violate* the margin (i.e., to be either closer to the separating hyperplane than the margin or misclassified altogether). This relaxation is achieved by replacing the original formulation (61.4a)–(61.4b) by the following optimization problem:

$$\left(w^\star, \theta^\star, \{s^\star(n)\}\right) = \underset{w,\theta,s(n)}{\operatorname{argmin}} \left\{ \frac{1}{2}\|w\|^2 \;+\; \eta\left(\frac{1}{N}\sum_{n=0}^{N-1} s(n)\right)\right\} \tag{61.11a}$$

$$\text{subject to } \gamma(n)(h_n^{\mathsf{T}} w - \theta) \geq 1 - s(n) \tag{61.11b}$$

$$s(n) \geq 0, \quad n = 0, 1, 2, \ldots, N-1 \tag{61.11c}$$

where $w \in \mathbb{R}^M$, $\theta \in \mathbb{R}$, $\eta > 0$ is a scaling parameter, and the $\{s(n) \geq 0\}$ are newly introduced nonnegative variables, called the *slack* variables. There is one slack variable for each data point in the training set. From expression (61.11b), we see that each slack variable $s(n)$ introduces some tolerance and allows the quantity $\gamma(n)(h_n^{\mathsf{T}} w - \theta)$ to be smaller than one. That is, it allows the point $(\gamma(n), h_n)$ to violate the margin since $1 - s(n)$ can be smaller than one, in which case h_n ends up being closer to the hyperplane than desired, or perhaps even on the wrong side of it. Two types of violations are possible:

(a) (**Margin violation**) Values of $s(n)$ in the range $0 \leq s(n) \leq 1$ will correspond to points $(\gamma(n), h_n)$ that fall on the correct side of the separating hyperplane but are closer to the hyperplane than the margin.

(b) (**Misclassification**) Values $s(n) > 1$ will correspond to points $(\gamma(n), h_n)$ that fall on the *wrong* side of the separating hyperplane and are therefore misclassified.

Compared with (61.4a), the cost function in (61.11a) incorporates an additional term that penalizes the contribution from the slack variables; the size of this penalty is controlled by the parameter η. By minimizing the augmented cost, we are in effect attempting to reduce the contribution from the slack deviations. Note that large values for η favor solutions $(w^\star, \theta^\star)$ with a small slack contribution and, hence, with a smaller number of misclassification errors. In particular, as $\eta \to \infty$, problem (61.11a)–(61.11c) reduces to the hard-margin SVM formulation (61.4a)–(61.4b since this situation will force all $s(n) \to 0$. On the other hand, smaller values for η accommodate some violations of the margin including more misclassifications.

Empirical risk

By examining the structure of problem (61.11a)–(61.11c) we can readily deduce the values of the slack variables $s(n)$ for all data points:

(a) (**Zero slack variables**) To begin with, whenever some data point $(\gamma(n_o), h_{n_o})$ satisfies $\gamma(n_o)(h_{n_o}^{\mathsf{T}} w - \theta) \geq 1$, then the corresponding slack variable, $s(n_o)$, should be zero. That is, data points than are on the correct side of the hyperplane and are farther away from it than its margin, will necessarily have zero slack variables. This is because the objective is to reduce the cost (61.11a) and, therefore, we can set $s(n_o)$ to zero to reduce the sum of the slack variables without violating (61.11b)–(61.11c).

(b) (**Positive slack variables**) On the other hand, whenever $\gamma(n_1)(h_{n_1}^{\mathsf{T}} w - \theta) < 1$ for some data point $(\gamma(n_1), h_{n_1})$, then the smallest value that can be chosen for the corresponding slack variable is

$$s(n_1) = 1 - \gamma(n_1)(h_{n_1}^{\mathsf{T}} w - \theta) > 0 \tag{61.12}$$

in order to satisfy the nonnegativity constraint (61.11c). We select the smallest value for $s(n_1)$ because the cost (61.11a) penalizes the sum of the slack variables.

Based on these observations, we are motivated to consider the following alternative formulation of the optimization problem (61.11a)–(61.11c):

$$(w^\star, \theta^\star) = \underset{w \in \mathbb{R}^M, \theta \in \mathbb{R}}{\operatorname{argmin}} \left\{ P(w) \triangleq \rho \|w\|^2 + \frac{1}{N} \sum_{n=0}^{N-1} \max\Big\{0,\, 1 - \gamma(n)\left(h_n^{\mathsf{T}} w - \theta\right)\Big\} \right\} \tag{61.13}$$

where $\rho = 1/2\eta$. Note that large values for η correspond to small values for ρ.

Accordingly, small ρ will favor solutions with a small number of margin violations or misclassifications (i.e., solutions with mostly small slack variables). This means that small values for ρ are recommended for data that are more or less separable, with $\rho \to 0$ corresponding to the hard-margin solution. On the other hand, larger values for ρ tolerate a higher level of margin violations and/or misclassifications. This case is better suited for data that are more challenging to separate.

If we invoke ergodicity on the data $\{\gamma(n), h_n\}$, we find that $P(w)$ motivates the following stochastic risk function

$$\frac{1}{N}\sum_{n=0}^{N-1}\max\Big\{0,\,1-\gamma(n)(h_n^{\mathsf{T}}w-\theta)\Big\} \;\;\overset{N\to\infty}{\longrightarrow}\;\; \mathbb{E}\,\max\Big\{0,\,1-\boldsymbol{\gamma}(\boldsymbol{h}^{\mathsf{T}}w-\theta)\Big\} \tag{61.14}$$

so that the soft-margin SVM construction can also be interpreted as solving the following Bayesian inference problem

$$(w^o,\theta^o) = \underset{w\in\mathbb{R}^M,\theta\in\mathbb{R}}{\operatorname{argmin}}\left\{\rho\|w\|^2 + \mathbb{E}\,\max\Big\{0,\,1-\boldsymbol{\gamma}(\boldsymbol{h}^{\mathsf{T}}w-\theta)\Big\}\right\} \tag{61.15}$$

where the expectation is over the joint distribution of $(\boldsymbol{\gamma},\boldsymbol{h})$.

Online recursion

Problem (61.13) can be solved by a variety of stochastic optimization methods, already discussed in previous chapters, such as using stochastic subgradient algorithms and variations thereof. It is sufficient to illustrate the construction by considering one solution method. We will therefore focus on stochastic subgradient implementations, with or without regularization, that rely on instantaneous subgradient approximations. The sampling of the data in the stochastic implementation can also be done with or without replacement. Using the result of Example 16.8, we list the SVM algorithm for solving (61.13) in (61.22), where the notation $\mathbb{I}[x]$ refers to the indicator function that is equal to one when condition x is true and zero otherwise. Comparing (61.22) with the Perceptron listing (60.19), we find that the condition $\mathbb{I}[\boldsymbol{\gamma}(n)\widehat{\boldsymbol{\gamma}}(n)\le 0]$ is now replaced by $\mathbb{I}[\boldsymbol{\gamma}(n)\widehat{\boldsymbol{\gamma}}(n)\le 1]$.

We can simplify the notation in listing (61.22) by extending the feature and weight vectors as follows:

$$h \leftarrow \begin{bmatrix}1\\ h\end{bmatrix}, \quad w \leftarrow \begin{bmatrix}-\theta\\ w\end{bmatrix} \tag{61.16}$$

so that the recursions can be rewritten more compactly in the following manner where the offset parameter is now implicit:

$$\begin{cases}\widehat{\boldsymbol{\gamma}}(n) = \boldsymbol{h}_n^{\mathsf{T}}\boldsymbol{w}_{n-1}\\ \boldsymbol{w}_n = A\boldsymbol{w}_{n-1} + \Big(\mu\boldsymbol{\gamma}(n)\,\mathbb{I}\,[\boldsymbol{\gamma}(n)\widehat{\boldsymbol{\gamma}}(n)\le 1]\Big)\boldsymbol{h}_n, \;\; n\ge 0\end{cases} \tag{61.17}$$

and the diagonal matrix A depends on the regularization parameter:

$$A \triangleq \begin{bmatrix} 1 & \\ & (1-2\mu\rho)I_M \end{bmatrix} \tag{61.18}$$

When a mini-batch of size B is used, the SVM recursion is replaced by

$$\begin{cases} \text{select } B \text{ data samples } \{\boldsymbol{\gamma}(b), \boldsymbol{h}_b\} \text{ at random} \\ \boldsymbol{\gamma}(b) = \boldsymbol{h}_b^{\mathsf{T}} \boldsymbol{w}_{n-1}, \;\; b = 0, 1, \ldots, B-1 \\ \boldsymbol{w}_n = A\boldsymbol{w}_{n-1} + \displaystyle\sum_{b=0}^{B-1} \Big(\mu\boldsymbol{\gamma}(b)\, \mathbb{I}\,[\boldsymbol{\gamma}(b)\widehat{\boldsymbol{\gamma}}(b) \le 1]\Big)\boldsymbol{h}_b, \;\; n \ge 0 \end{cases} \tag{61.19}$$

On the other hand, in the absence of regularization ($\rho = 0$), we obtain:

$$\boldsymbol{w}_n = \boldsymbol{w}_{n-1} + \mu\boldsymbol{\gamma}(n)\boldsymbol{h}_n\, \mathbb{I}\,[\boldsymbol{\gamma}(n)\widehat{\boldsymbol{\gamma}}(n) \le 1]\,, \;\; n \ge 0 \tag{61.20}$$

which can be rewritten in the equivalent form

$$\boxed{\boldsymbol{w}_n = \boldsymbol{w}_{n-1} \;+\; \mu\boldsymbol{\gamma}(n)\boldsymbol{h}_n, \;\; \text{if } \; \boldsymbol{\gamma}(n)\widehat{\boldsymbol{\gamma}}(n) \le 1} \tag{61.21}$$

Support vector machine (SVM) algorithm for minimizing (61.13).

$$\begin{array}{l} \text{given dataset } \{\gamma(m), h_m\}_{m=0}^{N-1} \text{ or streaming data } (\gamma(n), h_n); \\ \text{start from an arbitrary initial condition, } \boldsymbol{w}_{-1}. \\ \textbf{repeat until convergence over } n \ge 0: \\ \quad \text{select at random or receive a sample } (\boldsymbol{\gamma}(n), \boldsymbol{h}_n) \text{ at iteration } n; \\ \quad \widehat{\boldsymbol{\gamma}}(n) = \boldsymbol{h}_n^{\mathsf{T}}\boldsymbol{w}_{n-1} - \boldsymbol{\theta}(n-1) \\ \quad \boldsymbol{\theta}(n) = \boldsymbol{\theta}(n-1) - \mu\boldsymbol{\gamma}(n)\mathbb{I}\,[\boldsymbol{\gamma}(n)\widehat{\boldsymbol{\gamma}}(n) \le 1] \\ \quad \boldsymbol{w}_n = (1-2\mu\rho)\boldsymbol{w}_{n-1} + \mu\boldsymbol{\gamma}(n)\boldsymbol{h}_n\, \mathbb{I}\,[\boldsymbol{\gamma}(n)\widehat{\boldsymbol{\gamma}}(n) \le 1] \\ \textbf{end} \\ \text{return } w^\star \leftarrow \boldsymbol{w}_n, \;\; \theta^\star \leftarrow \boldsymbol{\theta}(n); \\ \text{classify a feature } h \text{ by using the sign of } \widehat{\gamma} = h^{\mathsf{T}}w^\star - \theta^\star \end{array} \tag{61.22}$$

Example 61.1 (**Binary classification using soft-SVM**) We show in Fig. 61.3 a collection of 150 feature samples $\boldsymbol{h}_n \in \mathbb{R}^2$ whose classes ± 1 are known beforehand: 120 samples are selected for training and 30 samples are selected for testing. The data arises from the dimensionally reduced iris dataset from Example 57.3; we denoted the two-dimensional reduced feature vectors by the notation h'_n in that example. We denote them by h_n here. We employ the two classes shown in the bottom plot of Fig. 57.5 and denote them by $\gamma(n) \in \{\pm 1\}$. We will use the data to compare the performance of the Perceptron and SVM algorithms.

We first use the data to train the Perceptron classifier (60.26), under extensions (60.20), by running 5 passes over the training data:

$$\widehat{\gamma}(n) = h_n^{\mathsf{T}} w_{n-1} \tag{61.23a}$$

$$w_n = w_{n-1} \;+\; \gamma(n)h_n, \qquad \text{if } \; \gamma(n)\widehat{\gamma}(n) \le 0 \tag{61.23b}$$

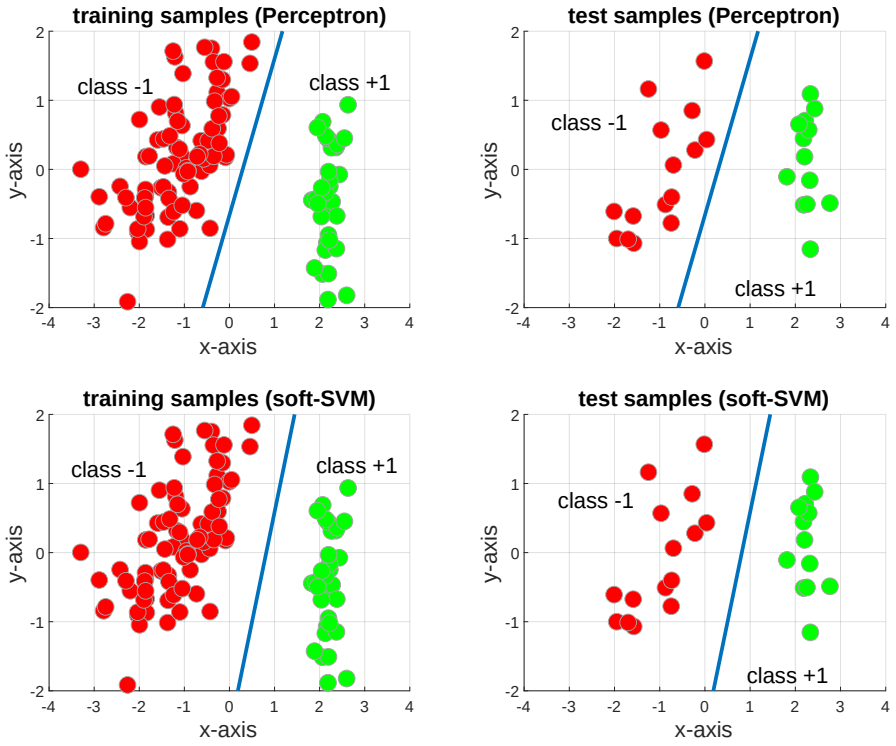


Figure 61.3 The first row shows the training and test data for the Perceptron algorithm without regularization and $\mu = 1$, while the second row shows the same data for the soft-margin SVM algorithm under ℓ_2-regularization with $\rho = 0.01$ and $\mu = 0.1$. The lines show the resulting classifiers.

During each pass, the data $\{\gamma(n), h_n\}$ is randomly reshuffled and the algorithm is re-run over the data starting from the weight iterate obtained at the end of the previous pass. The line in the figure shows the separating curve obtained in this manner with parameters (where we now undo the extension (60.20)):

$$w^\star = \begin{bmatrix} 3.4184 \\ -1.5104 \end{bmatrix}, \quad \theta^\star = 1.0, \quad (\textbf{Perceptron}) \tag{61.24}$$

It is seen that the separation curve is able to classify all test vectors and leads to 0% empirical error rate.

We also use the same data to run 5 passes of the soft-SVM classifier (61.22) by using $\rho = 0.01$ and $\mu = 0.1$. The data is randomly reshuffled at the start of each pass. The line in the figure shows the separating curve obtained in this manner with parameters

$$w^\star = \begin{bmatrix} 1.2253 \\ -0.3855 \end{bmatrix}, \quad \theta^\star = 1.0, \quad (\textbf{soft-SVM}) \tag{61.25}$$

It is also seen that the separation curve is able to classify all test vectors and leads to 0% empirical error rate.

Example 61.2 (**Application to breast cancer dataset**) We apply the soft-SVM classifier (61.22) to the breast cancer dataset encountered earlier in Example 53.3. The data consists of $N = 569$ samples, with each sample corresponding to a benign or malignant cancer classification. We use $\gamma(n) = -1$ for benign samples and $\gamma(n) = +1$ for malignant samples. Each feature vector in the data contains $M = 30$ attributes corresponding to measurements extracted from a digitized image of a fine needle aspirate (FNA) of a breast mass. The attributes describe characteristics of the cell nuclei present in the image; examples of these attributes were listed earlier in Table 53.1.

All feature vectors are centered around the sample mean and their variances scaled to unity according to the preprocessing step described earlier under PCA in (57.6). We select 456 samples (80%) randomly from these processed vectors for training and keep the remaining 113 samples (20%) for testing. We use $\rho = 0.01$ and $\mu = 0.01$. We run the algorithm 20 passes over the training data using random reshuffling. The resulting empirical error rate on the test data is 12.39%, resulting from 14 misclassified samples out of 113 test samples.

For comparison purposes, we use the PCA procedure (57.34) to reduce the dimension of the feature space down to $M = 2$ and run again the same soft-SVM procedure over this reduced data. Figure 61.2 shows the 456 training samples and 113 test samples, along with the resulting classifier whose parameters are determined to be

$$w^\star = \begin{bmatrix} -1.1022 \\ 0.6507 \end{bmatrix}, \quad \theta^\star = -0.07, \quad (\textbf{soft-SVM}) \tag{61.26}$$

The resulting empirical error on the test data is found to be 5.31%, which amounts to 6 misclassified decisions out of 113 test samples.

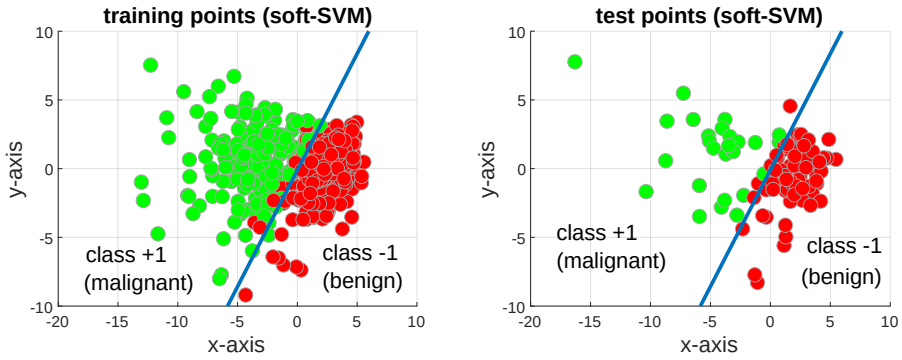


Figure 61.4 The plots show the training and test samples for 2−dimensional reduced feature vectors from a breast cancer dataset, along with the separating line that arises from training a soft-SVM classifier.

Example 61.3 (**Support vectors and misclassification errors**) The number of support vectors in an SVM implementation conveys useful information about the learning ability of the SVM solution. Specifically, consider repeated experiments involving training data $\{\gamma(n), h_n\}$ of size N each. Then, it holds that the average number of support vectors over these experiments provides an indication of the expected empirical error rate over

the training data for the SVM classifier, denoted generically by $c^\star$, namely,

$$\boxed{\mathbb{E}\,\boldsymbol{R}_{\rm emp}(c^\star) \;\leq\; \frac{1}{N}\mathbb{E}\,[\#\text{ support vectors}]} \tag{61.27}$$

where the expectation is over experiments (or over the distribution of the data $(\boldsymbol{\gamma}, \boldsymbol{h})$). The empirical error rate is denoted in boldface because it is treated as a random variable whose value varies from one experiment to another; recall from definition (52.11) that $R_{\rm emp}(c^\star)$ counts the fraction of errors over the training data. Observe that the bound on the right-hand side is independent of the dimension M of the feature space $h \in \mathbb{R}^M$, which is a useful property of SVM solutions. Observe also that an SVM solution is expected to yield very few support vectors; otherwise, the SVM classifier would not be effective.

Proof of (61.27) Note that, for any set of training data of size N, the number of support vectors satisfies:

$$\begin{aligned}[\#\text{ support vectors}] &= [\#\text{ training data that meet or violate the margin}] \\ &\geq [\#\text{ misclassified training data}]\end{aligned} \tag{61.28}$$

Therefore, the empirical error rate over the training data in each experiment satisfies:

$$R_{\rm emp}(c^\star) \;\triangleq\; \frac{1}{N}[\#\text{ misclassified data}] \leq \frac{1}{N}[\#\text{ support vectors}] \tag{61.29}$$

Taking expectations of both sides, we arrive at (61.27).

■

Example 61.4 (**SVM for regression problems**) We refer to the empirical risk (61.13) used by SVM for binary classification, namely,

$$\widehat{\gamma}(n) = h_n^{\mathsf{T}} w - \theta \tag{61.30a}$$

$$(w^\star, \theta^\star) \;\triangleq\; \underset{w\in\mathbb{R}^M,\theta\in\mathbb{R}}{\operatorname{argmin}} \left\{ \rho\|w\|^2 + \frac{1}{N}\sum_{n=0}^{N-1} \max\Big\{0, 1-\gamma(n)\widehat{\gamma}(n)\Big\}\right\} \tag{61.30b}$$

This formulation relies on the non-differentiable hinge function $g(x) = \max\{0, 1-x\}$, which ignores all values $x > 1$. We can motivate a similar construction for the solution of *regression* (as opposed to classification) problems, where the purpose is to estimate the *target* variables $\gamma(n)$ (rather than their signs). For this purpose, we consider the following regularized formulation:

$$\widehat{\gamma}(n) = h_n^{\mathsf{T}} w - \theta \tag{61.31a}$$

$$(w^\star, \theta^\star) \;\triangleq\; \underset{w\in\mathbb{R}^M,\theta\in\mathbb{R}}{\operatorname{argmin}} \left\{ \rho\|w\|^2 + \frac{1}{N}\sum_{n=0}^{N-1} \max\Big\{0, |\gamma(n)-\widehat{\gamma}(n)| - \epsilon\Big\}\right\} \tag{61.31b}$$

for some small $\epsilon > 0$. This description continues to rely on a non-differentiable function albeit one of the form $g(x) = \max\{0, |x|-\epsilon\}$ so that only values $x \in (-\epsilon, \epsilon)$ are ignored. This is illustrated schematically in the diagram of Fig. 61.5, where the vertical axis is denoted by y. The function has two points of discontinuity at $x = \pm\epsilon$. The slope of the function is $+1$ for $x > \epsilon$, -1 for $x < -\epsilon$, and zero for $x \in (-\epsilon, \epsilon)$. At $x = \epsilon$ we select the subgradient as $+1$ and at $x = -\epsilon$ as -1. We therefore construct a subgradient for $g(x)$ as follows:

$$\partial\, g(x) = \mathbb{I}[x \geq \epsilon] - \mathbb{I}[x \leq -\epsilon] \tag{61.32}$$

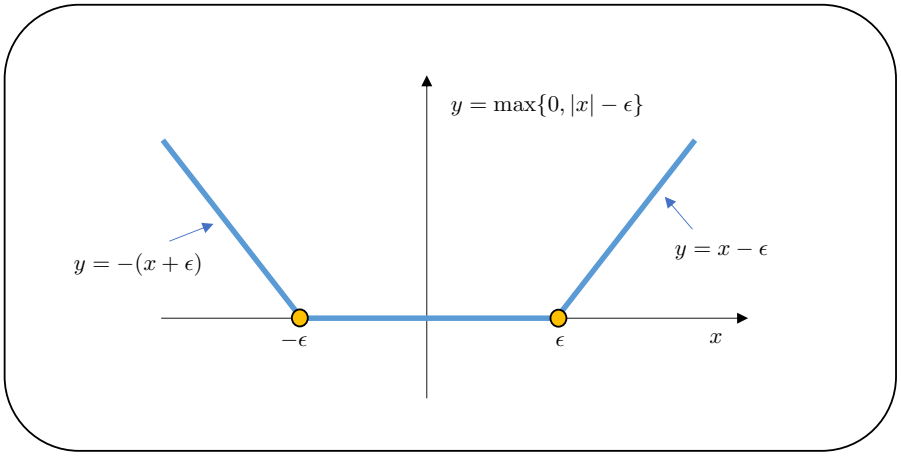


Figure 61.5 Plot of the function $y = \max\{0, |x| - \epsilon\}$.

Applying this construction to (61.31b) we can write down the following stochastic subgradient implementation:

$$\widehat{\boldsymbol{\gamma}}(n) = \boldsymbol{h}_n^{\mathsf{T}}\boldsymbol{w}_{n-1} - \boldsymbol{\theta}(n-1) \tag{61.33a}$$

$$\boldsymbol{\alpha}(n) = \mathbb{I}\Big[\widehat{\boldsymbol{\gamma}}(n) \leq \boldsymbol{\gamma}(n) - \epsilon\Big] - \mathbb{I}\Big[\widehat{\boldsymbol{\gamma}}(n) \geq \boldsymbol{\gamma}(n) + \epsilon\Big] \tag{61.33b}$$

$$\boldsymbol{\theta}(n) = \boldsymbol{\theta}(n-1) - \mu\boldsymbol{\alpha}(n) \tag{61.33c}$$

$$\boldsymbol{w}_n = (1 - 2\mu\rho)\boldsymbol{w}_{n-1} + \mu\,\boldsymbol{\alpha}(n)\boldsymbol{h}_n \tag{61.33d}$$

61.2 CONVEX QUADRATIC PROGRAM

There are several ways by which the hard and soft-margin SVM formulations can be solved. In listing (61.22) we pursued an online solution based on a stochastic subgradient implementation, which is one of the simplest and most commonly used methods for solving SVM problems. In this section, we describe another solution method that is based on transforming the SVM problem into a convex quadratic program (i.e., into an optimization problem with a quadratic cost function subject to a convex constraint — see future Eqs. (61.42a)–(61.42b)). Such quadratic programs can be solved efficiently by means of convex optimization packages. The main motivation for the derivation that follows is to highlight the role played by *support vectors*; the derivation will also be useful later when we develop a kernel-based SVM version for classifying data that are not necessarily linearly separable. The details of the convex program formulation are as follows.

Optimization by duality

We focus initially on the hard-margin SVM problem (61.4a)–(61.4b). We call upon the KKT conditions (9.28a)–(9.28e) to transform the constrained problem into an unconstrained version. Specifically, we start by introducing the Lagrangian function:

$$\mathcal{L}(w,\theta,\lambda(n)) \;\triangleq\; \frac{1}{2}\|w\|^2 \;-\; \sum_{n=0}^{N-1}\lambda(n)\left(\gamma(n)(h_n^{\mathsf{T}}w-\theta)-1\right) \tag{61.34}$$

where the $\{\lambda(n)\geq 0\}$ denote Lagrange multipliers; they are non-negative because of the direction of the inequalities in the constraints (61.4b). To determine the solution $(w^\star,\theta^\star)$, we need to perform two tasks. First, we minimize $\mathcal{L}(w,\theta,\lambda(n))$ over (w,θ) and determine the minimum value, which we denote by the dual function $\mathcal{D}(\lambda(n))$; it is a function of the multipliers $\{\lambda(n)\}$ alone. Second, we maximize the dual function over the $\{\lambda(n)\}$. From the solutions to these two steps, and in view of the KKT conditions, we will be able to recover the desired $(w^\star,\theta^\star)$, as we proceed to explain.

Computing the gradients of $\mathcal{L}(w,\theta,\lambda(n))$ relative to w and θ we get

$$\nabla_{w^{\mathsf{T}}}\mathcal{L}(w,\theta,\lambda(n)) = w - \sum_{n=0}^{N-1}\lambda(n)\gamma(n)h_n \tag{61.35a}$$

$$\partial\mathcal{L}(w,\theta,\lambda(n))\Big/\partial\theta = \sum_{n=0}^{N-1}\lambda(n)\gamma(n) \tag{61.35b}$$

Setting these gradients to zero at $(w^\star,\theta^\star)$, we find that the variables $\{w^\star,\lambda(n)\}$ must satisfy:

$$w^\star = \sum_{n=0}^{N-1}\lambda(n)\gamma(n)h_n, \qquad \sum_{n=0}^{N-1}\lambda(n)\gamma(n)=0 \tag{61.36}$$

Using these conditions, we substitute into the Lagrangian function and determine the dual function as follows:

$$\begin{aligned}
&D(\lambda(n))\\
&=\mathcal{L}(w^\star,\theta^\star,\lambda(n))\\
&=\frac{1}{2}\|w^\star\|^2 \;-\; \sum_{n=0}^{N-1}\lambda(n)\left(\gamma(n)(h_n^{\mathsf{T}}w^\star-\theta^\star)-1\right)\\
&=\frac{1}{2}\|w^\star\|^2+\sum_{n=0}^{N-1}\lambda(n)-\left(\sum_{n=0}^{N-1}\lambda(n)\gamma(n)h_n^{\mathsf{T}}\right)w^\star+\left(\sum_{n=0}^{N-1}\lambda(n)\gamma(n)\right)\theta^\star\\
&\overset{(61.36)}{=}\frac{1}{2}\|w^\star\|^2 \;+\; \sum_{n=0}^{N-1}\lambda(n) \;-\; \|w^\star\|^2\\
&\overset{(61.36)}{=}\sum_{n=0}^{N-1}\lambda(n) \;-\; \frac{1}{2}\sum_{n=0}^{N-1}\sum_{m=0}^{N-1}\gamma(n)\gamma(m)\lambda(n)\lambda(m)h_n^{\mathsf{T}}h_m
\end{aligned} \tag{61.37}$$

The resulting dual function is dependent on the $\{\lambda(n)\}$ alone and is given by:

$$\mathcal{D}(\lambda(n)) \triangleq \sum_{n=0}^{N-1} \lambda(n) - \frac{1}{2} \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} \gamma(n)\gamma(m)\lambda(n)\lambda(m) h_n^{\mathsf{T}} h_m \tag{61.38}$$

which we now need to maximize subject to the constraints

$$\lambda(n) \geq 0, \quad \sum_{n=0}^{N-1} \lambda(n)\gamma(n) = 0 \tag{61.39}$$

We can express the dual function in vector form by introducing the vector and matrix quantities:

$$\lambda \triangleq \begin{bmatrix} \lambda(0) \\ \lambda(1) \\ \vdots \\ \lambda(N-1) \end{bmatrix}, \quad \gamma = \begin{bmatrix} \gamma(0) \\ \gamma(1) \\ \vdots \\ \gamma(N-1) \end{bmatrix}, \quad [A]_{n,m} = \gamma(n)\gamma(m) h_n^{\mathsf{T}} h_m \tag{61.40}$$

The vector λ is $N \times 1$ and the matrix A (also called the *Gramian* matrix) is $N \times N$. Then, we can rewrite (61.38) as:

$$\mathcal{D}(\lambda) = \mathbb{1}^{\mathsf{T}}\lambda - \frac{1}{2}\lambda^{\mathsf{T}} A \lambda \tag{61.41}$$

We wish to maximize $\mathcal{D}(\lambda)$, which can be achieved by minimizing $-\mathcal{D}(\lambda)$. Hence, the problem of determining the $\{\lambda(n)\}$ is formulated as follows:

$$\lambda^\star = \underset{\lambda \in \mathbb{R}^N}{\operatorname{argmin}} \left\{ \frac{1}{2}\lambda^{\mathsf{T}} A \lambda - \mathbb{1}^{\mathsf{T}}\lambda \right\} \tag{61.42a}$$

$$\text{subject to } \lambda \succeq 0, \quad \lambda^{\mathsf{T}}\gamma = 0 \tag{61.42b}$$

where the notation $a \succeq b$ means element-wise comparison. The above problem is a convex quadratic programming problem: it involves a cost (61.42a) that is quadratic in λ, with coefficients $\{\frac{1}{2}A, -\mathbb{1}\}$. It also involves the linear constraint $\lambda^{\mathsf{T}}\gamma = 0$ and the condition $\lambda \succeq 0$. A quadratic program solver can be used to return a vector $\lambda^\star$.

Support vectors

The solution $\lambda^\star$ will exhibit a useful property, namely, most of its entries will be zero. This is because of the KKT *complementary condition* (9.28d), which needs to hold. That condition translates into the requirement:

$$\lambda^\star(n)\Big(\gamma(n)(h_n^{\mathsf{T}} w^\star - \theta^\star) - 1\Big) = 0, \quad n = 0, 1, 2, \ldots, N-1 \tag{61.43}$$

Now, consider any data point $(\gamma(n), h_n)$ that exceeds the margin, i.e., for which $\gamma(n)(h_n^{\mathsf{T}} w^\star - \theta^\star) > 1$; these points are correctly classified. Then, from (61.43), it must hold for these points that $\lambda^\star(n) = 0$. On the other hand, if $\lambda^\star(n) \neq 0$, then it must hold that $\gamma(n)(h_n^{\mathsf{T}} w^\star - \theta^\star) = 1$ so that nonzero values for $\lambda^\star(n)$ will only

occur for data points that meet the margin. There are generally only a few of these points and they are examples of *support vectors*. More generally, support vectors were defined in (61.10) as any points $(\gamma(n), h_n)$ that meet or violate the margin. In the hard-margin SVM formulation under discussion, support vectors will only consist of points $(\gamma(n), h_n)$ that meet the margin with *equality* sign in (61.10).

Observe further from condition (61.39) that there should exist at least one support vector from each class $\{\pm 1\}$. This is because if $\lambda^\star(n_1)$ is some nonzero entry of the vector $\lambda^\star$ corresponding to label $\gamma(n_1)$, then there should exist another entry of similar value $\lambda(n_2)$ in the vector $\lambda^\star$ albeit with label $\gamma(n_2) = -\gamma(n_1)$. When this happens, the two terms $\lambda^\star(n_1)\gamma(n_1)$ and $\lambda^\star(n_2)\gamma(n_2)$ cancel each other and it becomes possible for the sum in (61.39) to evaluate to zero, as required.

Using the solution $\lambda^\star$ we can determine $w^\star$ by using relation (61.36):

$$w^\star = \sum_{n=0}^{N-1} \lambda^\star(n)\gamma(n)h_n \tag{61.44}$$

But since most of the $\{\lambda^\star(n)\}$ will be zero, this expression actually provides a *sparse* representation for $w^\star$ in terms of the support vectors; it shows that $w^\star$ is a *linear* combination of the support vectors. We can therefore write

$$\boxed{w^\star = \sum_{s\in\mathcal{S}} \lambda^\star(s)\gamma(s)h_s} \tag{61.45}$$

where the sum is limited to the set $\mathcal{S}$ of support vectors.

We still need to determine $\theta^\star$. For that purpose, we pick any point $(\gamma(n), h_n)$ that meets the margin (i.e., any support vector in the hard-margin SVM implementation) and use it to solve for $\theta^\star$:

$$\gamma(n)(h_n^{\mathsf{T}} w^\star - \theta^\star) = 1 \implies \theta^\star = h_n^{\mathsf{T}} w^\star - \frac{1}{\gamma(n)} \tag{61.46}$$

We can enhance the accuracy of this construction for $\theta^\star$ by averaging estimates over all support vectors that meet the margin (or several of them), say, as:

$$\theta^\star = \frac{1}{|\mathcal{S}|}\sum_{s\in\mathcal{S}} \left(h_s^{\mathsf{T}} w^\star - \frac{1}{\gamma(s)} \right) \tag{61.47}$$

where $|\mathcal{S}|$ denotes the cardinality of $\mathcal{S}$. Obviously, under hard-margin SVM it holds that $\mathcal{S}_1 = \mathcal{S}$ since all support vectors meet the margin. Combining (61.45) and (61.47) we estimate the label of a test vector h by using the following expression (which is written in terms of the support vectors)

$$\begin{aligned} \widehat{\gamma} &= h^{\mathsf{T}} w^\star - \theta^\star \\ &= \sum_{s\in\mathcal{S}} \lambda^\star(s)\gamma(s) h^{\mathsf{T}} h_s \;-\; \frac{1}{|\mathcal{S}|}\sum_{s\in\mathcal{S}} \left(\sum_{s'\in\mathcal{S}} \lambda^\star(s')\gamma(s') h_s^{\mathsf{T}} h_{s'} - \frac{1}{\gamma(s)} \right) \end{aligned} \tag{61.48}$$

and making the classification decision:

$$\begin{cases} \text{if } \widehat{\gamma} \geq 0, \text{ assign } h \text{ to class } +1 \\ \text{if } \widehat{\gamma} < 0, \text{ assign } h \text{ to class } -1 \end{cases} \tag{61.49}$$

We summarize the solution method of this section in the following listing.

Convex program solution of hard-margin SVM (61.4a)–(61.4b)

(training)

compute :

given N data points $\{\gamma(n), h_n\}, n = 0, 1, \ldots, N-1$;
form the vector γ and matrix A defined by (61.40);
solve (61.42a)–(61.42b) and determine $\lambda^\star$;
$\mathcal{S} \triangleq$ set of support vectors defined by (61.10):
these are the points $(\gamma(s), h_s)$ with $\lambda^\star(s) \neq 0$.

$$w^\star = \sum_{s \in \mathcal{S}} \lambda^\star(s)\gamma(s)h_s$$

$$\theta^\star = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \left(h_s^{\mathsf{T}} w^\star - \frac{1}{\gamma(s)} \right)$$

end

return $(w^\star, \theta^\star)$

(classification)

classify feature vector h using (61.49) where $\widehat{\gamma} = h^{\mathsf{T}} w^\star - \theta^\star$. (61.50)

Soft-margin adjustment

Following similar arguments, we can verify that the soft-margin SVM problem (61.11a)–(61.11c) reduces to a convex quadratic programming problem of the following form, where the main modification is the upper bound on the entries of λ — see Prob. 61.9:

$$\lambda^\star = \underset{\lambda \in \mathbb{R}^N}{\text{argmin}} \quad \left\{ \frac{1}{2}\lambda^{\mathsf{T}} A \lambda \; - \; \mathbb{1}^{\mathsf{T}} \lambda \right\} \tag{61.51a}$$

$$\text{subject to } 0 \preceq \lambda \preceq \frac{\eta}{N}\mathbb{1}, \quad \lambda^{\mathsf{T}} \gamma = 0 \tag{61.51b}$$

In this case, it turns out that the solution vector $\lambda^\star$ will have nonzero entries at data points that meet the margin *and* also at data points that violate the margin. As explained earlier in (61.10), these points constitute the *support vectors*. The same listing (61.50) will continue to hold with one adjustment to the expression for $\theta^\star$. Let $\mathcal{S}_1 \subset \mathcal{S}$ denote the subset of support vectors that *meet* the margin. Then, we estimate $\theta^\star$ by averaging over these vectors (or a subset of them), say, as:

$$\theta^\star \; = \; \frac{1}{|\mathcal{S}_1|} \sum_{s \in \mathcal{S}_1} \left(h_s^{\mathsf{T}} w^\star - \frac{1}{\gamma(s)} \right) \tag{61.52}$$

and, therefore, the expression for $\widehat{\gamma}$ becomes:

$$\begin{aligned}\widehat{\gamma} &= h^{\mathsf{T}} w^{\star} - \theta^{\star} \qquad (61.53)\\ &= \sum_{s\in\mathcal{S}} \lambda^{\star}(s)\gamma(s)h^{\mathsf{T}}h_s \;-\; \frac{1}{|\mathcal{S}_1|}\sum_{s\in\mathcal{S}_1}\left(\sum_{s'\in\mathcal{S}}\lambda^{\star}(s')\gamma(s')h_s^{\mathsf{T}}h_{s'} - \frac{1}{\gamma(s)}\right)\end{aligned}$$

61.3 CROSS VALIDATION

The material in this section is not specific to support vector machines but is applicable more broadly. We present it here because at this stage of our development, we are in a good position to motivate the useful technique of cross validation for selecting hyperparameters for learning algorithms. We have encountered several such algorithms so far, such as the nearest-neighbor rule, the $K-$means algorithm, logistic regression, Perceptron, support vector machines, recursive least-squares, and various other stochastic optimization methods with and without regularization. We will encounter additional algorithms in subsequent chapters such as AdaBoost, kernel methods, neural networks, and so forth. In most of these implementations, certain parameters, also called *hyperparameters*, need to be set by the designer such as regularization parameters, forgetting factors, step-sizes, number of clusters, etc. Two important questions arise:

(a) How do we pick a good learning algorithm for an application from among multiple possibilities? And how do we set the hyperparameters for the algorithm in a guided manner?

(b) How do we assess the performance of the algorithm, such as its empirical error rate in order to estimate its generalization ability?

One useful technique to answer these questions is *cross validation*. While there are several variations of cross validation, we describe one construction that is common in practice.

We denote the learning algorithm that is under study generically by the notation $\mathcal{A}_p$, where the letter $\mathcal{A}$ refers to the algorithm and the letter p refers to some hyperparameter that influences its performance. For example, $\mathcal{A}$ could be the logistic regression algorithm and p could be the regularization parameter, ρ.

We start with a total of $N_{\text{TOTAL}} = N+T$ data points, $\{\gamma(n), h_n\}$, where $\gamma(n)$ is the label corresponding to feature vector $h_n \in \mathbb{R}^M$. The set is split into two disjoint groups: a training group consisting of N data points and a test group consisting of T data points:

$$\{\gamma(n), h_n\}, \quad n = 0, 1, 2, \ldots, N-1, \qquad (\textbf{training data}) \qquad (61.54a)$$

$$\{\gamma(t), h_t\}, \quad t = 0, 1, 2, \ldots, T-1, \qquad (\textbf{test data}) \qquad (61.54b)$$

$$N + T = N_{\text{TOTAL}} \qquad (61.54c)$$

Usually, the split is about $70-80\%$ of N_{TOTAL} used for training and $20-30\%$ of

N_{TOTAL} used for testing. If $N_{\text{TOTAL}} = 1000$, then we will have $N = 800$ training data points and $T = 200$ test data points. The test data should be separated completely from the training data and only used for testing purposes later after the classifier has been trained.

Training is performed as follows. We start from the N training data points and split them $K-$fold, where K is some integer normally between 5 and 10, though the value $K = 10$ is common. Let us select $K = 5$ for illustration purposes. Then, the N training points are split into K segments, with N/K data points in each segment. For the example with $N = 800$ and $K = 5$, we end up with 5 segments with $N_s = 160$ samples per segment. We index these segments by $s = 1, 2, 3, \ldots, K$ — see Fig. 61.6. During each iteration of the cross-validation procedure described below, one of the segments (also called a validation set) is left-out and used for cross validation purposes while the remaining $K - 1$ segments are used for training. This procedure, known as *$K-$fold cross-validation*, operates as follows:

repeat for $s = 1, 2, \ldots, K$:

(1) Exclude the data from the segment indexed by s, and use all data from the remaining $K-1$ segments to train the learning algorithm. For example, when $s = 1$ and $K = 5$, we use the data from segments $2, 3, 4$, and 5 for training. In the $N = 800$ example, this would amount to using a total of $4 \times 160 = 640$ data points for training. Note that we started from $N_{\text{TOTAL}} = 1000$ data points but are only using 640 for training algorithm $\mathcal{A}_p$. We can run multiple passes of the algorithm over the training data. Once training is completed, we test the performance of the resulting classifier using the data from the cross-validation segment, s, that was left out to measure its empirical error rate. If we let the set $\mathcal{N}_s$ denote the indexes of the data points within the cross validation segment, then this error is given by

$$R_{\text{emp}}(s) \;=\; \frac{1}{N_s} \sum_{n \in \mathcal{N}_s} \mathbb{I}\left[\mathcal{A}_p(h_n) \neq \gamma(n)\right] \tag{61.55}$$

This calculation counts the average number of erroneous classifications over the cross validation segment.

(2) We repeat the construction in step (1) for each of the segments: use one segment for cross validation and the remaining segments for training. In each run, we compute the resulting empirical error rate. By the time we have scanned over all K segments, we would have available K error values, $R_{\text{emp}}(s)$, one for each segment $s = 1, 2, \ldots, K$. We average these values to obtain an estimate for the error rate of the algorithm:

$$R_{\text{emp}}(\mathcal{A}_p) \;=\; \frac{1}{K} \sum_{s=1}^{K} R_{\text{emp}}(s) \tag{61.56}$$

(3) The important fact to recognize is that $R_{\text{emp}}(\mathcal{A}_p)$ estimates the performance

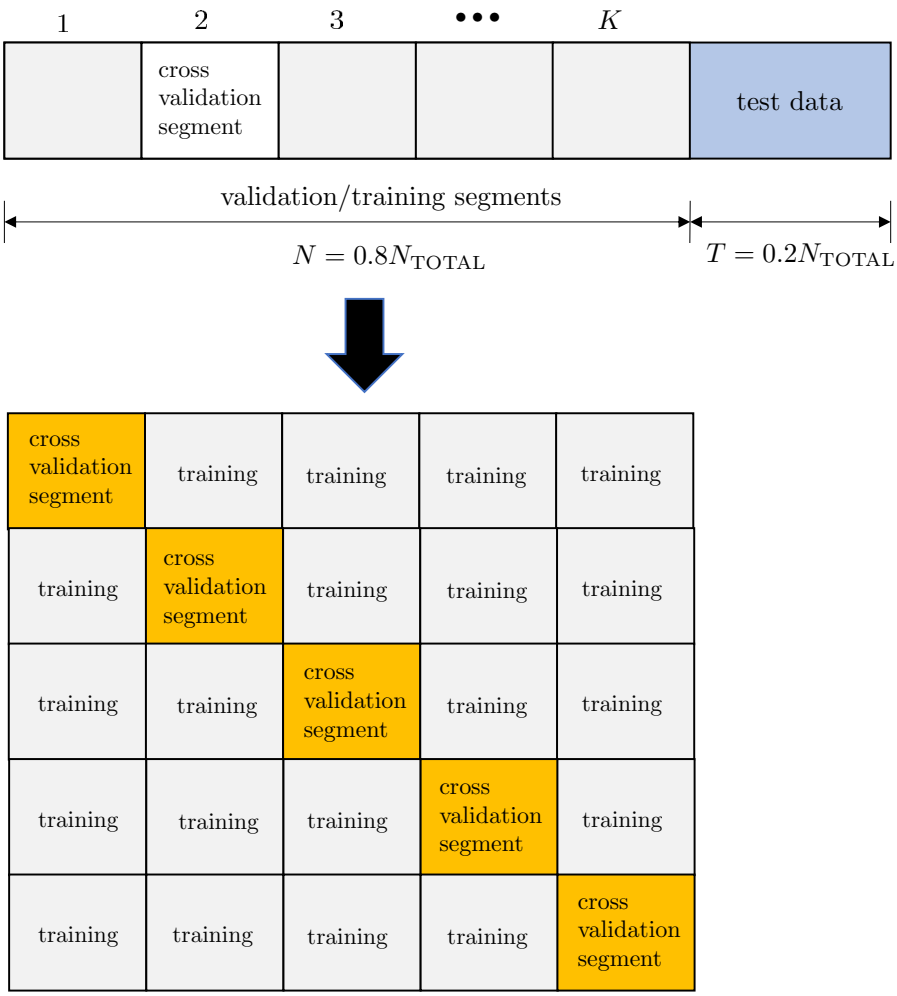


Figure 61.6 The data is divided into two parts, N and T with about 80% of the data points in the first set for training and 20% in the second set for testing. This first set is subsequently divided into K segments of width N_s each. During each iteration of the cross validation procedure, one of the segments is used for cross validation while the remaining segments are used for training.

of algorithm $\mathcal{A}$ for a particular parameter value p. We repeat steps (1)–(2) for different values of p, which would then allow us to arrive at a curve that shows how the error, $R_{\text{emp}}(\mathcal{A}_p)$, varies with p and subsequently pick the value of p that leads to the smallest error value.

(4) In another scenario, we may be interested in repeating steps (1)–(2) for different algorithms, while keeping the parameters fixed, in order to select the algorithm that results in the smallest value for $R_{\text{emp}}(\mathcal{A}_p)$.

end

Sometimes, the size of N may not be large enough for meaningful training. An alternative implementation is to employ a variation known as *leave-one-out* cross validation. In this case, we set $K = N$ so that each segment consists of a single data point. During cross validation, training will be performed by using $N - 1$ points and the empirical error will be evaluated on the single point that is left out.

At the end of the cross validation phase, we arrive at an answer to our first question about how to select the "best" algorithm or how to set "hyperparameters" in a guided manner. Once the algorithm and/or its hyperparameter(s) have been selected, we return to the full collection of N training data points, without excluding any segment for cross validation, and retrain the selected algorithm on this entire data set of N points, i.e., on the 800 points in our example. The resulting classifier is denoted by $\mathcal{A}^\star$.

We still need to answer the second question about how to test the performance of the "optimized" algorithm, $\mathcal{A}^\star$. To do so, we resort to the testing data (the T points) that we set aside and did not use during the cross validation procedure or training. We measure the empirical error rate on this test data:

$$R_{\text{emp}}(\mathcal{A}^\star) \;=\; \frac{1}{T}\sum_{t=0}^{T-1} \mathbb{I}\left[\mathcal{A}^\star(h_t) \neq \gamma(t)\right] \tag{61.57}$$

This calculation computes the average number of erroneous classifications over the test data for the learning algorithm and serves as its performance measure.

Example 61.5 (**Selecting the regularization parameter**) We apply the cross validation procedure to the selection of the regularization parameter ρ and the step-size parameter μ in an ℓ_2-regularized logistic regression implementation. We consider the same data from Example 59.2 except that we now examine the problem of separating class $r = 1$ from class $r = 2$. There are a total of $N_{\text{TOTAL}} = 100$ samples, with 50 samples from each class. We separate $T = 20$ samples for testing (that is 20% of the total number of samples) and use the remaining $N = 80$ samples for training. We extend the feature vectors according to (59.16) and apply 100 passes of the ℓ_2-regularized logistic regression algorithm (59.15).

We generate two plots for the empirical error rate of the logistic learner. In one case, we fix the step-size parameter at $\mu = 0.01$ and vary the regularization parameter ρ in steps of one in the range $\rho \in [0, 20]$. In the second case, we fix the regularization parameter at $\rho = 5$ and vary the step-size μ in steps of 0.005 in the range $\mu \in [0.001, 0.1]$. We implement a $10-$fold cross validation scheme. That is, we set $K = 10$ and divide the training data into 10 segments of 8 samples each. We fix ρ at one particular value, and run the logistic regression on 9 segments while keeping the tenth segment for testing; this tenth segment generates an empirical error value. While running the algorithm on the nine segments we run it multiple times over the data using 100 passes. We repeat this procedure 10 times, using nine segments for training and one segment for testing, and subsequently average the empirical errors to determine the error rate that corresponds to the fixed value of ρ. We repeat the construction for other values of ρ and arrive at the curve shown on the left in Fig. 61.7. From this figure, it is evident that

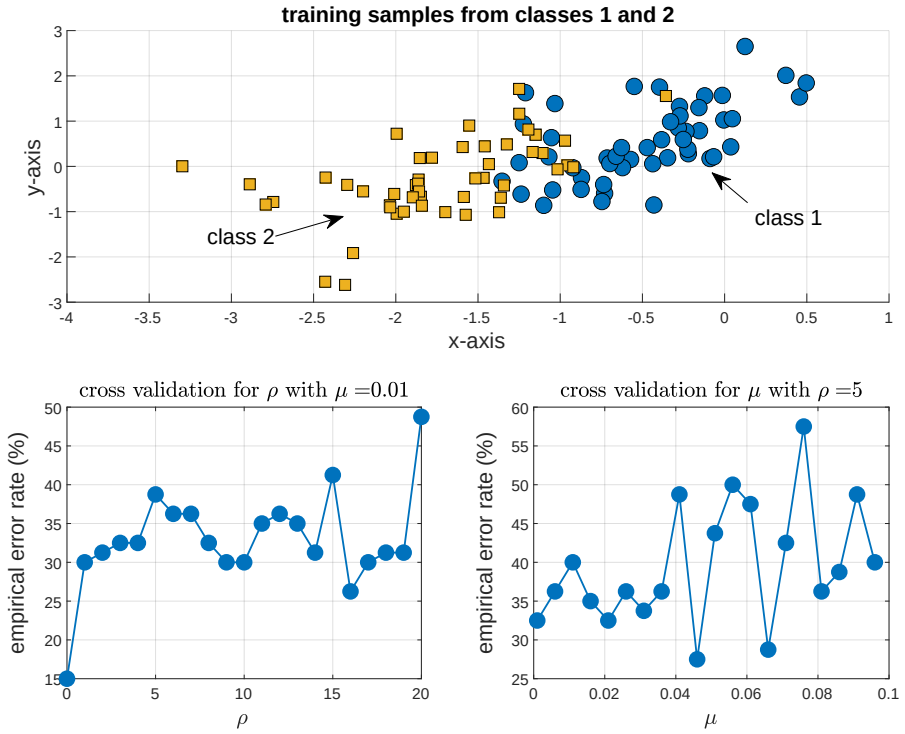


Figure 61.7 The plot on the left shows how the empirical error rate for the ℓ_2-regularized logistic regression algorithm varies with the selection of ρ. A 10−fold cross-validation implementation is used to generate this curve. The plot on the right shows the same curve as a function of the step-size parameter.

smaller values of ρ are preferred.

We repeat the same construction for the step-size parameter. We fix μ at one particular value, and run (59.15) on 9 of the segments while keeping the tenth segment for testing; this tenth segment generates an empirical error value. While running the algorithm on the nine segments we run it multiple times over the data using 100 passes. We repeat the procedure 10 times, using nine segments for training and one segment for testing, and subsequently average the empirical errors to determine the error rate that corresponds to the fixed value of μ. We repeat the construction for other values of μ and arrive at the curve shown on the right in the same Fig. 61.7.

Example 61.6 (**Structural risk minimization**) The cross-validation approach of this section can be seen as a form of *structural risk minimization*. Consider, for instance, the ℓ_2-regularized empirical risk formulation:

$$w^\star \triangleq \operatorname*{argmin}_{w\in\mathbb{R}^M} \Big\{ P(w) \;=\; q(w) + P_{\text{unreg}}(w) \Big\} \tag{61.58}$$

where we are expressing the risk $P(w)$ as the sum of two components: $q(w)$ denotes the convex regularization factor, and $P_{\text{unreg}}(w)$ denotes the remaining unregularized

component. For example, for the logistic regression problem:

$$w^\star \;=\; \operatorname*{argmin}_{w\in\mathbb{R}^M} \left\{ \rho\|w\|^2 \;+\; \frac{1}{N}\sum_{n=0}^{N-1} \ln\left(1+e^{-\gamma(n)h_n^{\mathsf{T}}w}\right)\right\} \tag{61.59}$$

we would have

$$q(w)=\rho\|w\|^2, \quad P_{\text{unreg}}(w)=\frac{1}{N}\sum_{n=0}^{N-1}\ln\left(1+e^{-\gamma(n)h_n^{\mathsf{T}}w}\right) \tag{61.60}$$

Now, we know from the earlier result (51.94) that, under some reasonable technical conditions that are usually satisfied for our problems of interest, solving a regularized problem of the form (61.58) is equivalent to solving

$$w^\star \;\triangleq\; \operatorname*{argmin}_{w\in\mathbb{R}^M} \; P_{\text{unreg}}(w), \quad \text{subject to } q(w)\le\tau \tag{61.61}$$

for some $\tau\ge 0$ dependent on ρ, written as $\tau(\rho)$. In other words, problem (61.61) is effectively searching for the classifier $w^\star$ within the set:

$$\mathcal{W}_\rho \;\triangleq\; \{w\in\mathbb{R}^M \mid q(w)\le\tau(\rho)\} \tag{61.62}$$

which is parameterized by ρ. By solving (61.58) for different values of ρ, as happens during a cross-validation procedure to select an optimal ρ, we are then searching for the solution $w^\star$ over successive sets $\{\mathcal{W}_{\rho_1},\mathcal{W}_{\rho_2},\ldots\}$ defined by successive values for the hyperparameter ρ. This sequence of nested optimization problems to determine an optimal classifier (i.e., the $w^\star$ corresponding to the optimal choice of ρ) is an example of "structural risk minimization."

61.4 COMMENTARIES AND DISCUSSION

Support vector machines. It is mentioned in the text by Vapnik (1979), and also in the article by Cortes and Vapnik (1995, p. 275), that the original idea of the hard-margin SVM formulation (61.4a)–(61.4b) was developed by Vapnik and Chervonenkis back in 1965, although the modern form of SVM and its kernel version first appeared in the publication by Boser, Guyon, and Vapnik (1992). The soft-margin formulation (61.11a)–(61.11c) appeared in Cortes and Vapnik (1995). Hard-margin SVM can be viewed as a nonlinear extension of the *Generalized Portrait* algorithm introduced by Vapnik and Lerner (1963) and further developed by Vapnik and Chervonenkis (1964). All these algorithms are based on the idea of seeking separating surfaces that maximize the margin from the training data, and have found applications in a range of areas including bioinformatics, image recognition, face detection, text processing, and others — see the overview by Burges (1998). Mentions of classifier designs that use large-margin hyperplanes also appear in the works by Cover (1965) and Duda and Hart (1973). For more information on SVM classifiers, their history, properties, and variations, the reader may refer to the texts by Vapnik (1995,1998), Scholkopf (1997), Cristianini and Shawe-Taylor (2000), Scholkopf and Smola (2001), Herbrich (2002), and Steinwart and Christmann (2008), as well as the articles by Burges (1998), Lin (2002), Lin, Lee, and Wahba (2002), and Smola and Scholkopf (2004).

Quadratic program. We explained in Sec. 61.2 that SVM problems can be recast as convex quadratic programs whose solutions can be pursued by duality arguments. These quadratic programs are extensions of a body of work from the early and mid sixties by Minnick (1961), Singleton (1962), Charnes (1964), and more broadly by Mangasarian

(1965,1968), who posed the binary classification problem as the solution to *linear* (as opposed to quadratic) programming problems. In linear programs, the objective function and the constraint function are all linear (affine) in the unknown w.

Slack variables. The soft-margin framework relies on introducing slack variables to enhance the robustness of the SVM solution. The idea of using slack variables is due to Smith (1968), whose work was motivated by the linear programming approach of Mangasarian (1965). The application of slack variables to separating hyperplanes appears in the article by Bennett and Mangasarian (1992). Result (61.27) relating the average number of support vectors to the expected empirical error rate for SVM classifiers appears in Boser, Guyon, and Vapnik (1992) and Cortes and Vapnik (1995).

Cross validation. One of the advantages of the cross validation procedure is that, by alternating over training and validation segments, it becomes possible to investigate the generalization performance of a learning algorithm without the need to collect additional training data. The technique performs generally well in practice although some difficulties may arise. For example, we discussed two versions of cross validation: the leave-one-out model and the $K-$fold model. In the leave-one-out procedure, one sample is left aside while training is performed on the remaining $N-1$ samples. When this is repeated a second time, a second sample is set aside and training is performed on the other $N-1$ samples, and so on. Note that the training data used during the successive training steps share $N-2$ data points. This means that the models that result from these training steps are highly correlated, which affects the quality of the estimate for the empirical risk in (61.56) since it is obtained by averaging strongly correlated quantities. This is one reason why it is preferred to employ the $K-$fold construction to reduce correlation between successive runs of the procedure. Nevertheless, the leave-one-out procedure is simpler and computationally less demanding than $K-$fold implementations.

The idea of setting aside some random subset of the data for subsequent testing is widely used in statistical analysis and correlation studies. Some of the earlier works involve, for example, contributions by Larson (1931) and Quenouille (1949,1957). According to Stone (1974), the method of cross validation in the form described in this chapter appears to have been originally developed by Lachenbruch (1965) who was motivated by the work of Mosteller and Wallace (1963). Useful early accounts, including discussion of $K-$fold cross validation, appear in Lachenbruch and Mickey (1968), Mosteller and Tukey (1968), and Luntz and Brailovsky (1969). Other earlier works dealing with cross validation techniques and their properties appear in Hills (1966), Cochran (1968), Allen (1974), Stone (1974,1977,1978), and Cox (1975). Further treatment on the subject, including more modern accounts and analysis of bias and variance properties, can be found in Devijver and Kittler (1982), Picard and Cook (1984), Breiman *et al.* (1984), Geisser (1993), Breiman (1996c), Holden (1996), Efron and Tibshirani (1997), Anthony and Holden (1998), Dietterich (1999), Nadeau and Bengio (2003), McLachlan (2004), Bengio and Grandvalet (2005), and Hastie, Tibshirani, and Friedman (2009). The results by Holden (1996) and Anthony and Holden (1998), in particular, provide a useful characterization of the quality of the empirical error rate estimated according to (61.56) in a $K-$fold implementation. They derived a bound on the probability that this empirical estimate is close enough to the true error rate of the classifier c by showing that, for any $0 < \delta < 1$, $N \geq K \geq 3$, and $N\delta^2 > 2K$:

$$\mathbb{P}\left(\sup_{c\in\mathcal{C}}\left|R_{\text{emp}}(\mathcal{A}_p) - R(c)\right| > \delta\right) \;\leq\; 2K\left(\frac{N(1+\frac{1}{K})e}{\text{VC}}\right)^{2\text{VC}} 2^{-N\delta/2K} \tag{61.63}$$

where $R_{\text{emp}}(\mathcal{A}_p)$ refers to the estimated empirical error rate the algorithm under consideration, $R(c)$ is the *actual* probability of error of classifier c, $\mathcal{C}$ is the class of classifiers over which the design is performed (such as limiting c to affine classifiers), and VC is a constant that measures the complexity of the set $\mathcal{C}$; for example, it is $M-1$ for affine classifiers in $\mathbb{R}^M$. We will define the VC dimension in a future chapter. The bound

on the right-hand side depends on δ, the size of the training data, N, the number of segments, K, and the VC dimension. The result is similar in form to the Vapnik-Chervonenkis bound, which we will derive in future expression (64.111).

PROBLEMS

61.1 Consider two feature vectors $\{h_a, h_b\}$ where h_a belongs to class $+1$ and h_b belongs to class -1. Assume that these two vectors meet the margin in an SVM implementation, that is, they satisfy $h_a^{\mathsf{T}} w^\star - \theta^\star = +1$ and $h_b^{\mathsf{T}} w^\star - \theta^\star = -1$. The parameters $(w^\star, \theta^\star)$ describe the separating hyperplane with maximal margin. Project the vector difference $h_a - h_b$ along the unit-norm normal to the separating hyperplane and determine the size of the margin, $m(w^\star)$, associated with $w^\star$ from this calculation.

61.2 Is the solution to the hard-margin SVM problem (61.4a)–(61.4b) unique?

61.3 Is the solution to the soft-margin SVM problem (61.11a)–(61.11c) unique?

61.4 Justify recursions (61.33a)–(61.33d) for the solution of an ℓ_2-regularized SVM risk for regression purposes. *Remark.* For more discussion on the use of the $\epsilon-$*insensitive loss* function $\max\{0, |x| - \epsilon\}$ in (61.31b), the reader may refer to Vapnik (1995,1998).

61.5 How would recursions (61.33a)–(61.33d) be modified if the empirical risk is ℓ_1-regularized and changed to

$$(w^\star, \theta^\star) \;\triangleq\; \underset{w\in\mathbb{R}^M, \theta\in\mathbb{R}}{\operatorname{argmin}} \left\{ \alpha\|w\|_1 + \frac{1}{N}\sum_{n=0}^{N-1} \max\Big\{0, (\gamma(n) - \widehat{\gamma}(n))^2 - \epsilon\Big\} \right\}$$

where $\widehat{\gamma}(n) = h_n^{\mathsf{T}} w - \theta$?

61.6 Refer to the statement of Prob. 60.7, except that now we wish to determine a separating hyperplane w such that $\gamma(n) h_n^{\mathsf{T}} w > 1$. We motivated the SVM recursion in the body of the chapter as one solution method. Here, we motivate a second *relaxation* method based on using the alternating projection algorithm from Sec. 12.6. Introduce the N halfspaces $\mathcal{H}_n = \{w \,|\, 1 - \gamma(n) h_n^{\mathsf{T}} w < 0\}$, one for each data pair $(\gamma(n), h_n)$. We are then faced with the problem of solving N linear inequalities and finding a point $w^\star$ in the intersection of these halfspaces. Use the result of Prob. 9.5 to show that the alternating projecting method motivates the following recursion:

$$w_n = w_{n-1} + \frac{\gamma(n) h_n}{\|h_n\|^2} \max\Big\{0, 1 - \gamma(n) h_n^{\mathsf{T}} w_{n-1}\Big\}$$

How is this method different from the hard-margin SVM recursion?

61.7 Consider a collection of $N-$data points $\{\gamma(m), h_m\}$ where $\gamma(m) \in \{\pm 1\}$ and $h_m \in \mathbb{R}^M$. Assume the data is linearly separable with zero offset, meaning that there exists some vector w such that $h_m^{\mathsf{T}} w > 0$ for features in class $+1$ and $h_m^{\mathsf{T}} w < 0$ for features in class -1. We know that such separating hyperplanes are highly non-unique. Consider the logistic regression formulation

$$\min_{w\in\mathbb{R}^M} \left\{ P(w) \;\triangleq\; \frac{1}{N}\sum_{m=0}^{N-1} \ln\left(1 + e^{-\gamma(m) h_m^{\mathsf{T}} w}\right) \right\}$$

Assume we apply the gradient-descent recursion repeatedly to minimize $P(w)$, namely,

$$w_n = w_{n-1} - \mu \nabla_{w^{\mathsf{T}}} P(w_{n-1}), \quad n \geq 0$$

Show that, for small μ, the iterate w_n converges to a limit satisfying

$$\lim_{n\to\infty} w_n/\|w_n\| \;=\; w^{\mathrm{svm}}/\|w^{\mathrm{svm}}\|$$

where w^{svm} is the solution to the hard-margin SVM problem:

$$w^{\text{svm}} = \underset{w\in\mathbb{R}^M}{\text{argmin}} \ \frac{1}{2}\|w\|^2, \ \text{subject to } \gamma(m)h_m^{\mathsf{T}}w \geq 1, \ \ m = 0, 1, 2, \ldots, N-1$$

Remark. The result of this problem provides another example of the *implicit bias* problem discussed in the comments of Chapter 29. The data is linearly separable and there exist infinitely many choices for the separating hyperplane. The gradient-descent algorithm chooses one particular solution from among these; namely, the one with the largest margin. See Soudry *et al.* (2018) for more discussion.

61.8 Verify that the Gramian matrix A defined by (61.40) is non-negative definite and conclude that the cost function in the minimization problem (61.42a) is convex.

61.9 Repeat the derivation given in Sec. 61.2 to show that the soft-margin SVM problem (61.11a)–(61.11c) can be rewritten as in (61.51a)–(61.51b). Explain that the solution $\lambda(n)$ will be nonzero at data points that meet or violate the margin. In particular, verify that:

$$\begin{cases} \lambda^\star(n) = 0, & \text{when } \gamma(n)(h_n^{\mathsf{T}}w^\star - \theta^\star) > 1 \\ \lambda^\star(n) = \eta/N, & \text{when } \gamma(n)(h_n^{\mathsf{T}}w^\star - \theta^\star) < 1 \\ 0 \leq \lambda^\star(n) \leq \eta/N, & \text{when } \gamma(n)(h_n^{\mathsf{T}}w^\star - \theta^\star) = 1 \end{cases}$$

REFERENCES

Allen, D. M. (1974), "The relationship between variable selection and data augmentation and a method of prediction," *Technometrics*, vol. 16, pp. 125–127.

Anthony, M. and S. B. Holden (1998), "Cross-validation for binary classification by real-valued functions: Theoretical analysis," in *Proc. International Conference on Computational Learning Theory* (COLT), pp. 218–229, Madison, WI, USA.

Bengio, Y. and Y. Grandvalet (2005), "Bias in estimating the variance of $K-$fold cross-validation," in *Statistical Modeling and Analysis for Complex Data Problems*, P. Duchesne and B. Remillard, *Editors*, pp. 75–95, Springer, NY.

Bennett, K. P. and O. L. Mangasarian (1992), "Robust linear programming discrimination of two linearly inseparable sets," *Optimization Methods and Software*, vol. 1, pp. 23–34.

Boser, B. E., I. Guyon, and V. N. Vapnik (1992), "A training algorithm for optimal margin classifiers," in *Proc. Annual Workshop on Computational Learning Theory* (COLT), pp. 144–152, Pittsburgh, PA, USA.

Breiman, L. (1996c), "Bias, variance and arcing classifiers," *Technical Report 460*, Statistics Department, University of California at Berkeley, Berkeley, CA.

Breiman, L., J. H. Friedman, R. A. Olshen, and C. J. Stone (1984), *Classification and Regression Trees*, Wadsworth International Group, Belmont, CA.

Burges, C. (1998), "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167.

Charnes, A. (1964), "Some fundamental theorems of Perceptron theory and their geometry," in *Computer and Information Sciences*, J. T. Tou and R. H. Wilcox, *Eds.*, Spartan Books, Washington, DC.

Cochran, W. G. (1968), "Commentary on estimation of error rates in discriminant analysis," *Technornetrics*, vol. 10, pp. 204–205.

Cortes, C. and V. N. Vapnik (1995), "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297.

Cover, T. M. (1965), "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition," *IEEE Trans. Electronic Computers*, vol. 14, pp. 326–334.

Cox, D. R. (1975), "A note on data-splitting for the evaluation of significance levels," *Biometrika*, vol. 62, no. 2, pp. 441–445.
Cristianini, N. and J. Shawe-Taylor (2000), *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*, Cambridge University Press.
Devijver, P. A. and J. Kittler (1982), *Pattern Recognition: A Statistical Approach*, Prentice Hall, NJ.
Dietterich, T. G. (1999), "Approximate statistical tests for comparing supervised classification learning algorithms," *Neural Computation*, vol. 10, pp. 1895–1924.
Duda, R. O. and P. E. Hart (1973), *Pattern Classification and Scene Analysis*, Wiley, NY.
Efron, B. and R. Tibshirani (1997), "Improvements on cross-validation: The .632+bootstrap method," *J. Amer. Statist. Assoc.*, vol. 92, no. 438, pp. 548–560.
Geisser, S. (1993), *Predictive Inference*, Chapman and Hall, NY.
Hastie, T., R. Tibshirani, and J. Friedman (2009), *The Elements of Statistical Learning*, 2nd edition, Springer, NY.
Herbrich, R. (2002), *Learning Kernel Classifiers: Theory and Algorithms*, MIT Press, Cambridge, MA.
Hills, M. (1966), "Allocation rules and their error rates," *J. Roy. Statist. Soc. B,*, vol. 28, pp. 1–31.
Holden, S. B. (1996), "Cross-validation and the PAC learning model," *Research Note RN/96/64*, Department of Computer Science, University College London.
Lachenbruch, P. (1965), *Estimation of Error Rates in Discriminant Analysis*, Ph.D. dissertation, University of California, Los Angeles.
Lachenbruch, P. and M. Mickey (1968), "Estimation of error rates in discriminant analysis," *Technometrics*, vol. 10, pp. 1–11.
Larson, S. (1931), "The shrinkage of the coefficient of multiple correlation," *J. Educat. Psychol.*, vol. 22, pp. 45–55.
Lin, Y. (2002), "Support vector machines and the Bayes rule in classification," *Data Mining and Knowledge Discovery*, vol. 6, pp. 259–275.
Lin, Y., Y. Lee, and G. Wahba (2002), "Support vector machines for classification in nonstandard situations," *Machine Learning*, vol. 46, pp. 191–202.
Luntz, A. and V. Brailovsky (1969), "On estimation of characters obtained in statistical procedure of recognition" *Techicheskaya Kibernetica*, vol. 3, pp. 6–12 (in Russian).
Mangasarian, O. L. (1965), "Linear and nonlinear separation of patterns by linear programming," *Operations Research*, vol. 13, pp. 444–452.
Mangasarian, O. L. (1968), "Multi-surface method of pattern separation," *IEEE Trans. Information Theory*, vol. 14, no. 6, pp. 801–807.
McLachlan, G. J. (2004), *Discriminant Analysis and Statistical Pattern Recognition*, Wiley, NY.
Minnick, R. C. (1961), "Linear-input logic," *IRE Trans. Electronic Computers*, vol. 10, pp. 6–16.
Mosteller, F. and J. W. Tukey (1968), " Data analysis, including statistics," In *Handbook of Social Psychology*, G. Lindzey and E. Aronson, *Eds.*, Addison-Wesley.
Mosteller, F. and D. L. Wallace (1963), "Inference in an authorship problem," *J. Amer. Stat. Assoc.*, vol. 58, pp. 275–309.
Nadeau, C. and Y. Bengio (2003), "Inference for the generalization error," *Machine Learning*, vol. 52, pp. 239–281.
Picard, R. and D. Cook (1984), "Cross-validation of regression models," *J. Amer. Statist. Assoc.*, vol. 79, no. 387, pp. 575–583.
Quenouille, M. (1949), "Approximate tests of correlation in time series," *J. Roy. Statist. Soc. B*, vol. 11, pp. 18–84.
Quenouille, M. (1957), *The Analysis of Multiple Time Series,* Griffin, London.
Scholkopf, B. (1997), *Support Vector Learning*, Oldenbourg Verlag.
Scholkopf, B. and A. J. Smola (2001), *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond,* MIT Press, Cambridge, MA.
Singleton, R. C. (1962), "A test for linear separability as applied to self-organizing

machines," *Proc. Conference on Self-Organizing Systems*, M. C. Yovits, G. T. Jacobi, and G. D. Goldstein, *Eds.*, pp. 503–524, Spartan Books, Washington, DC.

Smith, F. W. (1968), "Pattern classifier design by linear programming," *IEEE Trans. on Comput.*, vol. 17, no. 4, pp. 367–372.

Smola, A. J. and B. Scholkopf (2004), "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, no. 3, pp. 199–222.

Soudry, D., E. Hoffer, M. S. Nacson, S. Gunasekar, and N. Srebro (2018), "The implicit bias of gradient descent on separable data," *J. Machine Learning Research*, vol. 19, pp. 1–57.

Steinwart, I. and A. Christmann (2008), *Support Vector Machines*, Springer, Berlin.

Stone, M. (1974), "Cross-validatory choice and assessment of statistical predictions," *J. Royal Statist. Soc., Ser. B*, vol. 36, pp. 111–147.

Stone, M. (1977), "Asymptotics for and against cross-validation," *Biometrika*, vol. 64, no. 1, pp. 29–35.

Stone, M. (1978), "Cross-validation: A review," *Math. Operationsforsch. Statist., Ser. Statistics*, vol. 9, no. 1, pp. 127–139.

Vapnik, V. N. (1979), *Estimation of Dependences Based on Empirical Data*, Nauka, Moscow (in Russian). English translation published in 1982 by Springer-Verlag. Reprinted in 2006.

Vapnik, V. N. (1995), *The Nature of Statistical Learning Theory*, Springer, NY.

Vapnik, V. N. (1998), *Statistical Learning Theory*, Wiley, NY.

Vapnik V. N. and A. Y. Chervonenkis (1964), "A note on one class of perceptrons," *Automation and Remote Control*, vol. 25, no. 1.

Vapnik V. N. and A. Lerner (1963), "Pattern recognition using generalized portrait method," *Automation and Remote Control*, vol. 24, no. 6, pp. 774–780.