Chapter 2

Classification

In this chapter we study one of the most basic topics in machine learning called **classification**. By classification we meant a **supervised learning** procedure where we try to classify a sample based on a model learned from the data in a training dataset. The goal of this chapter is to understand the general principle of classification, to explore various methodologies so that we can do classification, and to make connections between different methods.

2.1 Discriminant Function

Basic Terminologies

A classification method always starts with a pair of variables $(\boldsymbol{x}, y)$. The first variable $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^d$ is an **input vector**. The set $\mathcal{X}$ is called the **input space**. Depending on the application, an input could the intensity of a pixel, a wavelet coefficient, the phase angle of wave, or anything we want to model. We assume that each input vector has a dimensionality d and d is finite. The second variable $y \in \{1, \ldots, k\} = \mathcal{Y}$ denotes the **class label** of the classes $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$. The choice of the class labels is arbitrary. They do not need to be positive integers. For example, in binary classification, one can choose $y \in \{0, 1\}$ or $y \in \{-1, +1\}$, depending which one is more convenient mathematically.

In supervised learning, we assume that we have a **training set** $\mathcal{D}$. A training set is a collection of paired variables $(\boldsymbol{x}_j, y_j)$, for $j = 1, \ldots, n$. The vector $\boldsymbol{x}_j$ denotes the j-th sample input in $\mathcal{D}$, and y_j denotes the corresponding class label. The relationship between $\boldsymbol{x}_j$ and y_j is specified by the **target function** $f : \mathcal{X} \to \mathcal{Y}$ such that $y_j = f(\boldsymbol{x}_j)$. The target function is *unknown*. By unknown we really mean that it is unknown.

The training set $\mathcal{D}$ can be generated deterministically or probabilistically. If $\mathcal{D}$ is deterministic, then $\{\boldsymbol{x}_j\}_{j=1}^n$ is a sequence of fixed vectors. If $\mathcal{D}$ is probabilistic, then there is an underlying generative model that *generates* $\{(\boldsymbol{x}_j)\}_{j=1}^n$. The generative model is specified by the distribution $p_{\boldsymbol{X}}(\boldsymbol{x})$. The distinction between a deterministic and a probabilistic model is

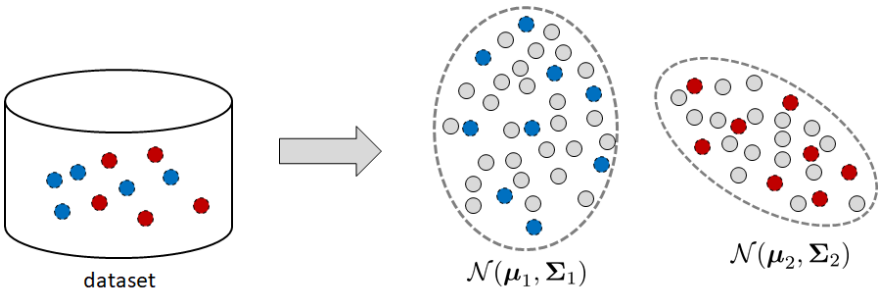


Figure 2.1: Visualizing a training data $\mathcal{D}$ consisting of two classes $\mathcal{C}_1$ and $\mathcal{C}_2$, with each class being a multi-dimensional Gaussian. The gray dots represent the samples **outside** the training set, whereas the color dots represent the samples **inside** the training set.

subtle but important. We will go into the details when we study learning theory in Chapter 4. For now, let us assume that $\boldsymbol{x}_j$'s are generated probabilistically.

When there is no noise in creating the labels, the class label y_j is defined as $y_j = f(\boldsymbol{x}_j)$. However, in the presence of noise, instead of using a fixed but unknown target function f to generate $y_j = f(\boldsymbol{x}_j)$, we assume that y_j is drawn from a posterior distribution $y \sim p_{Y|\boldsymbol{X}}(y|\boldsymbol{x})$. As a result, the training pair $(\boldsymbol{x}_j, y_j)$ is now drawn according to the **joint distribution** $p_{\boldsymbol{X},Y}(\boldsymbol{x}, y)$ of random variables $\boldsymbol{X}$ and Y. By Bayes Theorem, it holds that $p_{\boldsymbol{X},Y}(\boldsymbol{x}, y) = p_{Y|\boldsymbol{X}}(y|\boldsymbol{x})p_{\boldsymbol{X}}(\boldsymbol{x})$, which means the joint distribution $p_{\boldsymbol{X},Y}(\boldsymbol{x}, y)$ can be completely determined by the posterior distribution $p_{Y|\boldsymbol{X}}(y|\boldsymbol{x})$ and the data generator $p_{\boldsymbol{X}}(\boldsymbol{x})$.

Example. [Gaussian]. Consider a training set $\mathcal{D}$ consisting of training samples $\{(\boldsymbol{x}_j, y_j)\}$. The input vector $\boldsymbol{x}_j$ is a d-dimensional vector in $\mathbb{R}^2$, and the label y_j is either $\{1, 2\}$. The target function f is a mapping which assigns each $\boldsymbol{x}_j$ to a correct y_j. The left hand side of Figure 2.1 shows a labeled dataset, marking the data points in red and blue.

We assume that for each class, the input vectors are distributed according to a d-dimensional Gaussian:

$$p_{\boldsymbol{X}\,|\,Y}(\boldsymbol{x}\,|\,i) = \frac{1}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}_i|}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right\}, \tag{2.1}$$

where $\boldsymbol{\mu}_i \in \mathbb{R}^d$ is the mean vector, and $\boldsymbol{\Sigma}_i \in \mathbb{R}^{d\times d}$ is the covariance matrix. If we assume that $p_Y(i) = \pi_i$ for $i = 1, 2$, then the joint distribution of $\boldsymbol{X}$ and Y is defined through the Bayes Theorem

$$\begin{aligned} p_{\boldsymbol{X},Y}(\boldsymbol{x}, i) &= p_{X|Y}(x|i)p_Y(i) \\ &= \pi_i \cdot \frac{1}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}_i|}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right\}. \end{aligned}$$

In the above example, we can use a simple MATLAB / Python code to visualize the joint

distribution. Here, the proportion of class 1 is $\pi = 0.25$, and the proportion of class 2 is $1-\pi = 0.75$. When constructing the data, we define two random variables $Y \sim \text{Bernoulli}(\pi)$, and $\boldsymbol{X}$ with conditional distribution $\boldsymbol{X}|Y \sim \mathcal{N}(\boldsymbol{\mu}_Y, \boldsymbol{\Sigma}_Y)$. Therefore, the actual random number is drawn by a two-step procedure: First draw Y from a Bernoulli. If $Y = 1$, then draw $\boldsymbol{X}$ from the first Gaussian; If $Y = 0$, then draw $\boldsymbol{X}$ from the other Gaussian.

```
mu1    = [0 0];
mu0    = [2 0];
Sigma1 = [.25 .3; .3 1];
Sigma0 = [0.5 0.1; 0.1 0.3];
n      = 1000;           % number of samples
p      = 0.25;           % pi
y      = (rand(n,1)<=p); % class label
idx1   = (y==1);         % indices of class 1
idx0   = (y==0);         % indices of class 0
x      = mvnrnd(mu1, Sigma1, n).*repmat(y,[1,2]) + ...
         + mvnrnd(mu0, Sigma0, n).*repmat((1-y),[1,2]);
```

The plotting of the data points can be done by a scatter plot. In the code below, we use the indices `idx1` and `idx0` to identify the class 1 and class 0 data points. This is an aftermath label. When `x` was first generated, it was unlabeled. Therefore, in an unsupervised learning case, there will not be any coloring of the dots.

```
figure(1);
scatter(x(idx1,1),x(idx1,2),'rx', 'LineWidth', 1.5); hold on;
scatter(x(idx0,1),x(idx0,2),'bo', 'LineWidth', 1.5); hold off;
xlabel('x'); ylabel('y');
set(gcf, 'Position', [100, 100, 600, 300]);
```

The result of this example is shown in Figure 2.2. If we count of the number of red and blue markers, we see that there are approximately 25% red and 75% blue. This is a result of the prior distribution. Shown in this plot is a linearly not separable example, meaning that some data points will never be classified correctly by a linear classifier even if we use the classifier is theoretically optimal.

One thing we need to be careful in the above example is that the training samples $\{(\boldsymbol{x}_j, y_j)\}_{j=1}^n$ represent a **finite set of n samples** drawn from the distribution $p_{\boldsymbol{X},Y}(\boldsymbol{x}, y)$. The distribution $p_{\boldsymbol{X},Y}(\boldsymbol{x}, y)$ is "bigger" in the sense that many samples in $p_{\boldsymbol{X},Y}(\boldsymbol{x}, y)$ are not necessarily part of $\mathcal{D}$. Think about doing a survey in the United States about a person's height: The distribution $p_{\boldsymbol{X},Y}(\boldsymbol{x}, y)$ covers the entire population, whereas $\mathcal{D}$ is only a sub-collection of the data resulting from the survey. Samples that are **inside** the training set are called the **in-samples**, and samples that are **outside** the training set are called the **out-samples**. See Figure 2.1 for an illustration.

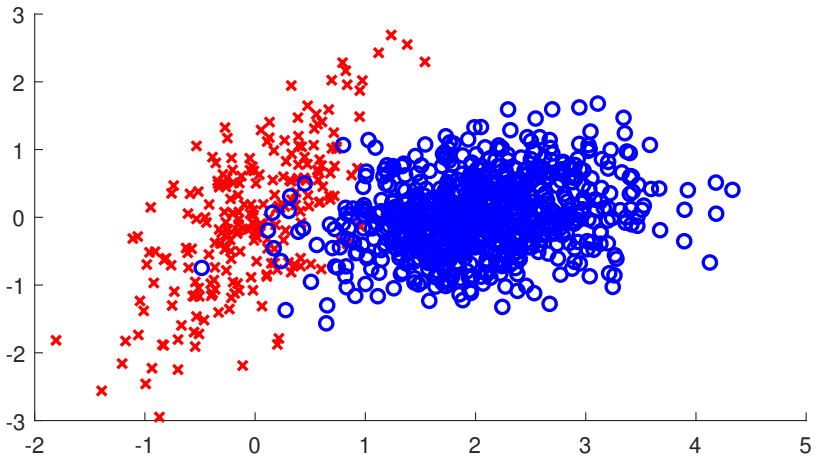


Figure 2.2: Generating random samples from two Gaussians where the class label is generated according to a Bernoulli distribution.

The difference between in-sample and out-sample is important. Straightly speaking, the Gaussian parameters $(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ in the previous example are called the population parameter, and they are *unknown* no matter how many people we have surveyed unless we have asked everyone. Given the training set $\mathcal{D}$, any model parameter estimated from $\mathcal{D}$ is the sample parameter $(\widehat{\boldsymbol{\mu}}_1, \widehat{\boldsymbol{\Sigma}}_1)$.

Given the training dataset $\mathcal{D}$, the goal of learning (in particular classification) is to pick a mapping $h : \mathcal{X} \to \mathcal{Y}$ that can minimize the error of misclassification. The mapping h is called a **hypothesis function**. A hypothesis function could be linear, nonlinear, convex, non-convex, expressible as equations, or an algorithm like a deep neural network. The set containing all candidate hypothesis functions is called the **hypothesis set** $\mathscr{H}$. See Figure 2.3 for an illustration.

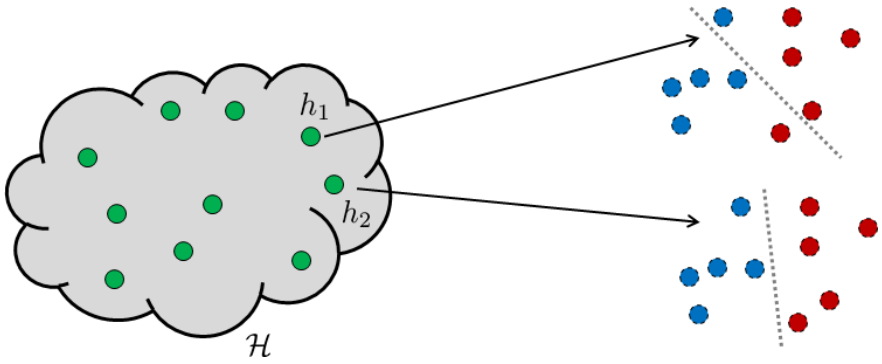


Figure 2.3: Hypothesis set $\mathscr{H}$ and hypothesis function h. The goal of learning is to use an algorithm to find a hypothesis function or approximate a hypothesis function such that the error of classification is minimized. In this example, h_2 is a better hypothesis because it has a lower misclassification rate.

So what is "error" in classification? To be clear about this, we are fundamentally asking two questions (i) How well does the hypothesis function h do for the **training** dataset, i.e.,

the **in-samples**? (ii) How well does it generalize to the **testing** dataset, i.e., the **out-samples**? A good hypothesis function should minimize the error for both. If a hypothesis function that only performs well for the training set but generalizes poorly to the testing dataset, then it is more or less useless as it can only *memorize* the pattern but not *predict* the pattern.

We will come back to the details of these fundamental questions later. For now, we will present a set of computational tools for seeking (or approximating) a hypothesis function. The methods we are presenting here are by no means exhaustive, but they all fall into the same general principle of linear classification.

Linear Discriminant Analysis

The goal of classification is to construct a good hypothesis function h. In this course we will focus on **linear classifiers** under a framework called linear discriminant analysis. Why are we interested in linear classifiers? First, linear classifiers are easy to understand. They have simple geometry, and they often allow analytic results. Second, linear classifiers contain most of the essential insight we need to understand a classification method, and the principle is generalizable to nonlinear classifiers.

So what is a linear classifier? Let us consider a simple binary $\{0, 1\}$ classification problem. The hypothesis function h of a binary classification problem takes the form of

$$h(\boldsymbol{x}) = \begin{cases} 1, & g(\boldsymbol{x}) > 0, \\ 0, & g(\boldsymbol{x}) < 0. \end{cases} \tag{2.2}$$

Here, the function $g : \mathcal{X} \to \mathbb{R}$ is called a **discriminant function**. The job of the discriminant function g is to map the vector $\boldsymbol{x}$ to a value $g(\boldsymbol{x})$ such that the class can be determined by checking the sign of $g(\boldsymbol{x})$. How does g look like? For linear classifiers, g take a linear form and is called a linear discriminant function.

Definition 1 (Linear Discriminant Function)**.** *A **linear discriminant function** is a mapping $g : \mathcal{X} \to \mathbb{R}$ with*

$$g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0, \tag{2.3}$$

for some vectors $\boldsymbol{w} \in \mathbb{R}^d$ and scalar $w_0 \in \mathbb{R}$.

In this definition, the vector $\boldsymbol{w}$ is called the **weight** of the linear classifier, and w_0 is called the **bias** of the classifier. To simplify the notation, we sometimes concatenate $\boldsymbol{w}$ and w_0 by defining $\boldsymbol{\theta} = \{\boldsymbol{w}, w_0\}$. We call $\boldsymbol{\theta} \in \mathbb{R}^{d+1}$ the **model parameter** of the linear discriminant function g. For different dimensionality d, the geometry of $g(\boldsymbol{x})$ changes. Figure 2.4 shows the examples in 1D, 2D and 3D.

A discriminant function does not need to be linear. For example, we can construct a quadratic discriminant function:

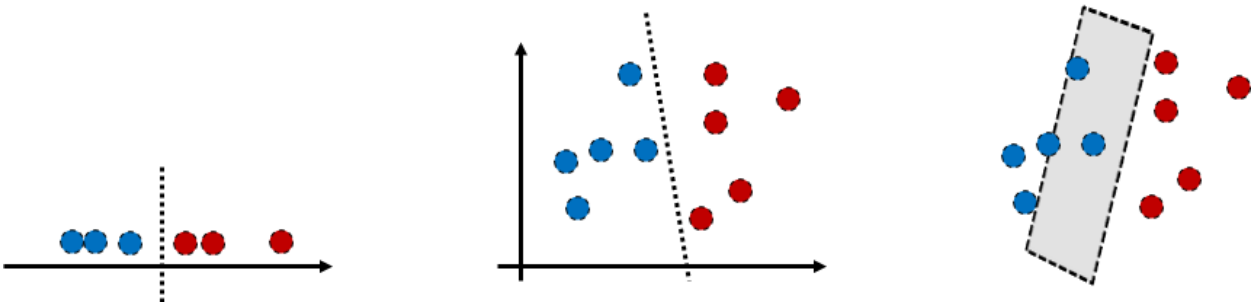


Figure 2.4: Discriminant function $g(\boldsymbol{x})$ at different dimensions. [Left] In 1D, the discriminant function is simply a cutoff value. [Middle] In 2D, the discriminant function is a line. [Right] In 3D, the discriminant function is a plane.

Definition 2 (Quadratic Discriminant Function)**.** *A* ***quadratic discriminant function*** *is a mapping* $g : \mathcal{X} \to \mathbb{R}$ *with*

$$g(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{W}\boldsymbol{x} + \boldsymbol{w}^T\boldsymbol{x} + w_0, \tag{2.4}$$

for some matrix $\boldsymbol{W} \in \mathbb{R}^{d\times d}$*, some vector* $\boldsymbol{w} \in \mathbb{R}^d$ *and some scalar* $w_0 \in \mathbb{R}$*.*

In quadratic discriminant function, the model parameter is $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{w}, w_0\}$. Depending on $\boldsymbol{W}$, the geometry of g could be convex, concave, or neither. Figure 2.5 below shows a quadratic discriminant function separating an inner and an outer cluster of data points.

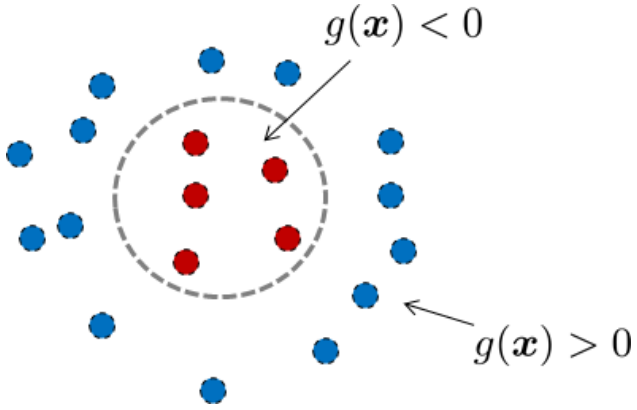


Figure 2.5: A quadratic discriminant function is able to classify data using quadratic surfaces. This example shows an ellipsoid surface for separating an inner and outer cluster of data points.

Beyond quadratic discriminant function, another common approach of constructing discriminant functions is the method of **transformation**. We shall come back to this idea later in this chapter. For now, let us focus on linear discriminant functions.

Geometry of Linear Discriminant Function

The equation $g(\boldsymbol{x}) = 0$ defines a **hyperplane** where on one side of the plane we have $\mathcal{C}_1$ and on the other side of the plane we have $\mathcal{C}_2$. The geometry is summarized below.

Theorem 1 (Geometry of a linear discriminant function)**.** *Let g be a linear discriminant function, and let $\mathcal{H} = \{\boldsymbol{x} \mid g(\boldsymbol{x}) = 0\}$ be the separating hyperplane. Then,*

(i) $\boldsymbol{w}/\|\boldsymbol{w}\|_2$ is the normal vector of $\mathcal{H}$, i.e., for any $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{H}$, we have

$$\boldsymbol{w}^T(\boldsymbol{x}_1 - \boldsymbol{x}_2) = 0. \tag{2.5}$$

(ii) For any $\boldsymbol{x}_0$, the distance between $\boldsymbol{x}_0$ and $\mathcal{H}$ is

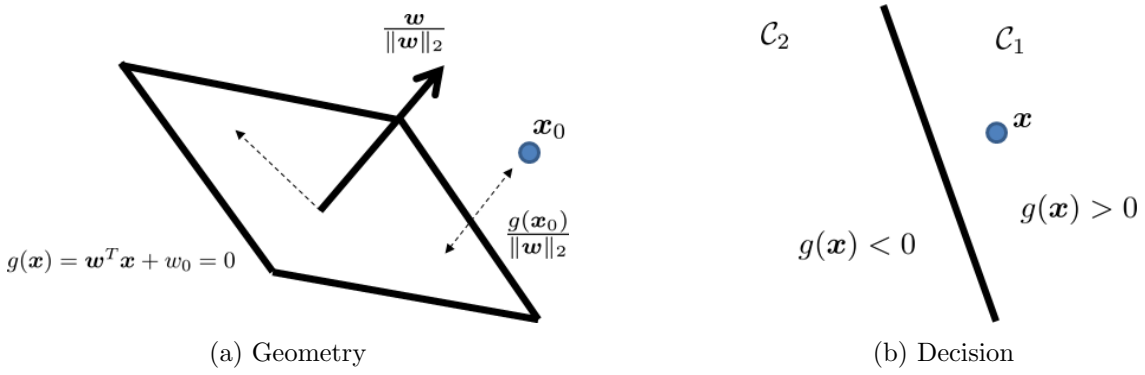
$$d(\boldsymbol{x}_0, \mathcal{H}) = g(\boldsymbol{x}_0)/\|\boldsymbol{w}\|_2. \tag{2.6}$$

Proof. To prove the first statement, we note consider two points $\boldsymbol{x}_1 \in \mathcal{H}$ and $\boldsymbol{x}_2 \in \mathcal{H}$. Since both points are on the hyperplane, we have $g(\boldsymbol{x}_1) = 0$ and $g(\boldsymbol{x}_2) = 0$, and hence $\boldsymbol{w}^T\boldsymbol{x}_1 + w_0 = \boldsymbol{w}^T\boldsymbol{x}_2 + w_0 = 0$. Rearranging the equations yields $\boldsymbol{w}^T(\boldsymbol{x}_1 - \boldsymbol{x}_2) = 0$. Therefore $\boldsymbol{w}$ is a normal vector to the surface $\mathcal{H}$, and $\boldsymbol{w}/\|\boldsymbol{w}\|_2$ is a scaled version of $\boldsymbol{w}$ so that the norm of $\boldsymbol{w}/\|\boldsymbol{w}\|_2$ is 1.

To prove the second statement, we can decompose $\boldsymbol{x}_0$ as $\boldsymbol{x}_0 = \boldsymbol{x}_p + \eta\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}$, where $\boldsymbol{x}_p$ is the orthogonal projection of $\boldsymbol{x}_0$ onto $\mathcal{H}$ such that $g(\boldsymbol{x}_p) = 0$. In other words, we decompose $\boldsymbol{x}_0$ into its two orthogonal components: one along the normal vector $\boldsymbol{w}$, and the other along the tangent. Since

$$\begin{aligned} g(\boldsymbol{x}_0) = \boldsymbol{w}^T\boldsymbol{x}_0 + w_0 &= \boldsymbol{w}^T\left(\boldsymbol{x}_p + \eta\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}\right) + w_0 \\ &= g(\boldsymbol{x}_p) + \eta\|\boldsymbol{w}\|_2 = \eta\|\boldsymbol{w}\|_2, \end{aligned}$$

we have that $\eta = g(\boldsymbol{x}_0)/\|\boldsymbol{w}\|_2$. □



(a) Geometry (b) Decision

Figure 2.6: The geometry of linear discriminant function.

Exercise 1.1. An alternative proof of the above result is to solve an optimization. Show that the solution to the problem

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \|\boldsymbol{x}-\boldsymbol{x}_0\|^2 \quad \text{subject to} \quad \boldsymbol{w}^T\boldsymbol{x}+w_0=0, \tag{2.7}$$

is given by

$$\boldsymbol{x}=\boldsymbol{x}_0-\frac{g(\boldsymbol{x}_0)}{\|\boldsymbol{w}\|_2}\cdot\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}. \tag{2.8}$$

Hint: Use Lagrange multiplier.

The geometry of the above theorem is illustrated in Figure 2.6. The theorem tells us two things about the hyperplane: (i) **Orientation of $\mathcal{H}$**. The orientation of $\mathcal{H}$ is specified by $\boldsymbol{w}/\|\boldsymbol{w}\|_2$, as it is the normal vector of $\mathcal{H}$. (ii) **Distance to $\mathcal{H}$**. The distance of a point $\boldsymbol{x}_0$ to $\mathcal{H}$ is $g(\boldsymbol{x}_0)/\|\boldsymbol{w}\|_2$. Therefore, the bigger the value $g(\boldsymbol{x}_0)$ is, the farther away the point $\boldsymbol{x}_0$ is from the decision boundary, and hence the more "confident" $\boldsymbol{x}_0$ is belonging to one class but not the other. To classify a data point $\boldsymbol{x}$, we check whether $\boldsymbol{x}$ belongs to $\mathcal{C}_1$ or $\mathcal{C}_2$ by checking if $g(\boldsymbol{x})>0$.

Separating Hyperplane Theorem

Before we move on to talk about different methods to construct the discriminant function g, we should ask a "learnability" question: For what kind of datasets can we guarantee that a linear classifier is able to *perfectly* classify the data? This is a *universal* result because the certificate derived from answering this question has to hold for *any* linear classifier and *any* algorithms used to train the classifier.

Theorem 2 (Separating Hyperplane Theorem)**.** *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two closed convex sets such that $\mathcal{C}_1\cap\mathcal{C}_2=\emptyset$. Then, there exists a linear function*

$$g(\boldsymbol{x})=\boldsymbol{w}^T\boldsymbol{x}+w_0,$$

such that $g(\boldsymbol{x})>0$ for all $\boldsymbol{x}\in\mathcal{C}_1$ and $g(\boldsymbol{y})<0$ for all $\boldsymbol{y}\in\mathcal{C}_2$.

Pictorially, the separating hyperplane says that if $\mathcal{C}_1$ and $\mathcal{C}_2$ are convex and do not overlap, then there must exists two points $\boldsymbol{x}^*\in\mathcal{C}_1$ and $\boldsymbol{y}^*\in\mathcal{C}_2$ such that $\|\boldsymbol{x}^*-\boldsymbol{y}^*\|^2$ represents the shortest distance between the two sets. The vector $\boldsymbol{w}=\boldsymbol{x}^*-\boldsymbol{y}^*$ is therefore a potential candidate for the normal of the separating hyperplane. The potential candidate for w_0 is then $-\boldsymbol{w}^T\left(\frac{\boldsymbol{x}^*+\boldsymbol{y}^*}{2}\right)$, i.e., the mid point of $\boldsymbol{x}^*$ and $\boldsymbol{y}^*$. Once $\boldsymbol{w}$ and w_0 are identified, we can check the inner product of any $\boldsymbol{x}\in\mathcal{C}_1$ with $\boldsymbol{w}$, i.e., $\boldsymbol{w}^T\boldsymbol{x}+w_0$, and check its sign. This gives the proof of the theorem.

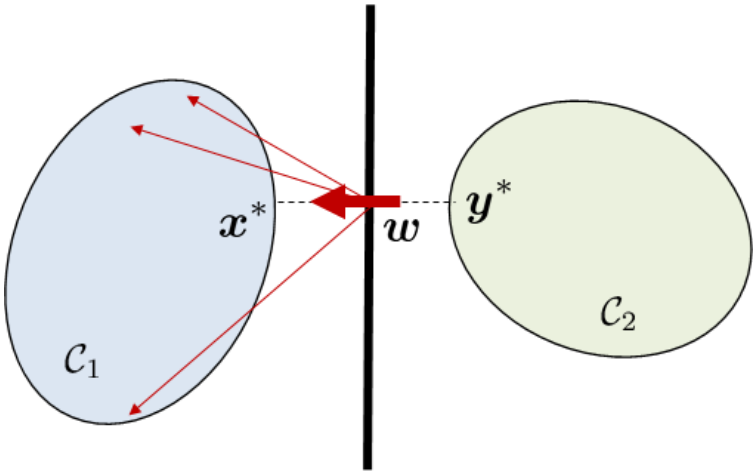


Figure 2.7: Pictorial illustration of the separating hyperplane theorem. Given two closed convex sets, we can find two extreme points $\boldsymbol{x}^*$ and $\boldsymbol{y}^*$, and hence define the normal vector $\boldsymbol{w}$. The proof of the theorem follows by showing any point in $\mathcal{C}_1$ will form an acute angle with $\boldsymbol{w}$, and hence $g(\boldsymbol{x}) > 0$ for all $\boldsymbol{x} \in \mathcal{C}_1$.

Proof. Since $\mathcal{C}_1$ and $\mathcal{C}_2$ are convex and closed, there exist $\boldsymbol{x}^* \in \mathcal{C}_1$ and $\boldsymbol{y}^* \in \mathcal{C}_2$ such that

$$\|\boldsymbol{x}^* - \boldsymbol{y}^*\|^2 = \min_{\boldsymbol{x}\in\mathcal{C}_1, \boldsymbol{y}\in\mathcal{C}_2} \|\boldsymbol{x} - \boldsymbol{y}\|^2 = d(\mathcal{C}_1, \mathcal{C}_2),$$

i.e., $\boldsymbol{x}^*$ and $\boldsymbol{y}^*$ are the two extreme points of the sets which minimizes the distance between the sets. We define $\boldsymbol{w} = \boldsymbol{x}^* - \boldsymbol{y}^*$, $\boldsymbol{x}_0 = \frac{\boldsymbol{x}^* + \boldsymbol{y}^*}{2}$. Thus,

$$g(\boldsymbol{x}) = \boldsymbol{w}^T(\boldsymbol{x} - \boldsymbol{x}_0) = \boldsymbol{w}^T\boldsymbol{x} - (\boldsymbol{x}^* - \boldsymbol{y}^*)^T\left(\frac{\boldsymbol{x}^* + \boldsymbol{y}^*}{2}\right) = \boldsymbol{w}^T\boldsymbol{x} - \frac{\|\boldsymbol{x}^*\|^2 - \|\boldsymbol{y}^*\|^2}{2},$$

and so we have $w_0 = -\frac{\|\boldsymbol{x}^*\|^2 - \|\boldsymbol{y}^*\|^2}{2}$. Our goal is check whether $g(\boldsymbol{x}) > 0$ for all $\boldsymbol{x} \in \mathcal{C}_1$.

Suppose not, then there exists $\boldsymbol{x} \in \mathcal{C}_1$ such that $g(\boldsymbol{x}) < 0$. This implies that

$$(\boldsymbol{x}^* - \boldsymbol{y}^*)^T\boldsymbol{x} - \frac{\|\boldsymbol{x}^*\|^2 - \|\boldsymbol{y}^*\|^2}{2} < 0. \tag{2.9}$$

Let $\boldsymbol{x}_\lambda = (1-\lambda)\boldsymbol{x}^* + \lambda\boldsymbol{x}$ for any $0 \le \lambda \le 1$. Convexity of $\mathcal{C}_1$ implies that $\boldsymbol{x}_\lambda \in \mathcal{C}_1$.

$$\begin{aligned}
\|\boldsymbol{x}_\lambda - \boldsymbol{y}^*\|^2 &= \|(1-\lambda)\boldsymbol{x}^* + \lambda\boldsymbol{x} - \boldsymbol{y}^*\|^2 \\
&= \|\boldsymbol{x}^* - \boldsymbol{y}^* + \lambda(\boldsymbol{x} - \boldsymbol{x}^*)\|^2 \\
&= \|\boldsymbol{x}^* - \boldsymbol{y}^*\|^2 + 2\lambda(\boldsymbol{x}^* - \boldsymbol{y}^*)^T(\boldsymbol{x} - \boldsymbol{x}^*) + \lambda^2\|\boldsymbol{x} - \boldsymbol{x}^*\|^2 \\
&\overset{(a)}{<} \|\boldsymbol{x}^* - \boldsymbol{y}^*\|^2 - \lambda\left[\|\boldsymbol{x}^* - \boldsymbol{y}^*\|^2\right] + \lambda^2\|\boldsymbol{x} - \boldsymbol{x}^*\|^2,
\end{aligned} \tag{2.10}$$

where the inequality in (a) holds because

$$\begin{aligned}(\boldsymbol{x}^*-\boldsymbol{y}^*)^T(\boldsymbol{x}-\boldsymbol{x}^*) &= (\boldsymbol{x}^*-\boldsymbol{y}^*)^T\boldsymbol{x} - (\boldsymbol{x}^*-\boldsymbol{y}^*)^T\boldsymbol{x}^* \\ &< \frac{\|\boldsymbol{x}^*\|^2-\|\boldsymbol{y}^*\|^2}{2} - \|\boldsymbol{x}^*\|^2 + (\boldsymbol{x}^*)^T\boldsymbol{y}^* \\ &= -\frac{\|\boldsymbol{x}^*\|^2}{2} + (\boldsymbol{x}^*)^T\boldsymbol{y}^* - \frac{\|\boldsymbol{y}^*\|^2}{2} \\ &= -\frac{1}{2}\|\boldsymbol{x}^*-\boldsymbol{y}^*\|^2.\end{aligned}$$

Finally, we just need to make sure the last two terms in Equation (2.10) is overall negative. To this end we note that

$$-\lambda\left[\|\boldsymbol{x}^*-\boldsymbol{y}^*\|^2\right] + \lambda^2\|\boldsymbol{x}-\boldsymbol{x}^*\|^2 = \lambda\left(-\|\boldsymbol{x}^*-\boldsymbol{y}^*\|^2 + \lambda\|\boldsymbol{x}-\boldsymbol{x}^*\|^2\right),$$

which is <0 if $-\|\boldsymbol{x}^*-\boldsymbol{y}^*\|^2+\lambda\|\boldsymbol{x}-\boldsymbol{x}^*\|^2<0$, i.e., $\lambda < \frac{\|\boldsymbol{x}^*-\boldsymbol{y}^*\|^2}{\|\boldsymbol{x}-\boldsymbol{x}^*\|^2}$. Therefore, for small enough λ, we are guaranteed that $-\lambda\left[\|\boldsymbol{x}^*-\boldsymbol{y}^*\|^2\right]+\lambda^2\|\boldsymbol{x}-\boldsymbol{x}^*\|^2<0$, and hence $\|\boldsymbol{x}_\lambda-\boldsymbol{y}^*\|^2<\|\boldsymbol{x}^*-\boldsymbol{y}^*\|^2$. This creates a contradiction, because by definition $\|\boldsymbol{x}^*-\boldsymbol{y}^*\|^2$ is the shortest distance between the two sets. □

What does separating hyperplane theorem buy us? The theorem tells us what to expect from a **linear classifier**. For example, if the one of the classes is not convex, then it is impossible to use a linear classifier to achieve perfect classification. If the two classes are overlapping, then it is also impossible to use a linear classifier to achieve perfect classification. See Figure 2.8 for an illustration.

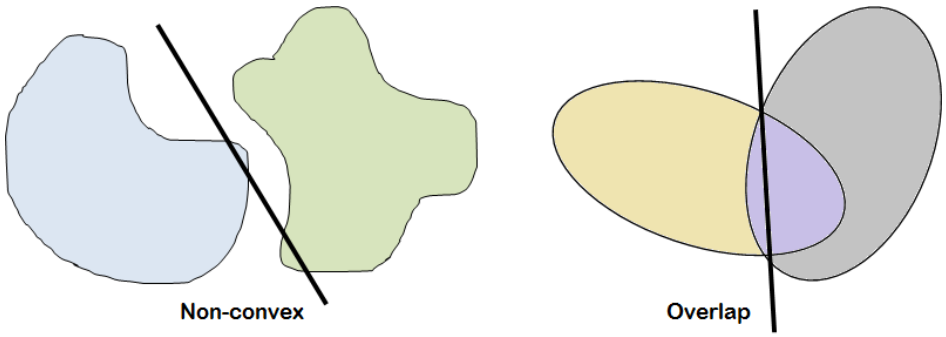


Figure 2.8: Examples where linear classifiers do not work. [Left] When at least of the two sets are not convex. [Right] When the two sets overlap. In both cases, if we use a linear classifier, there will be classification error even when the classifier is perfectly trained.

Separating hyperplane theorem also tells us that if the two classes are convex and non-overlapping, there will be a margin within which the separating hyperplane can still achieve perfect classification. Figure 2.9 shows an example. The implication is that the geometry

of the dataset can afford **uncertainty** of the classifier, e.g., due to insufficient amount of training data. However, there is a price we need to pay when picking a separating hyperplane that is not the one suggested by the theorem. When there is any perturbation of a testing data point, either through noise or through adversarial attack, a non-optimal separating hyperplane are more susceptible to misclassification. This brings a **robustness** issue to the classifier. The optimal linear classifier (optimal in the sense of maximum margin) is the **support vector machine** which we will discuss later in the chapter.

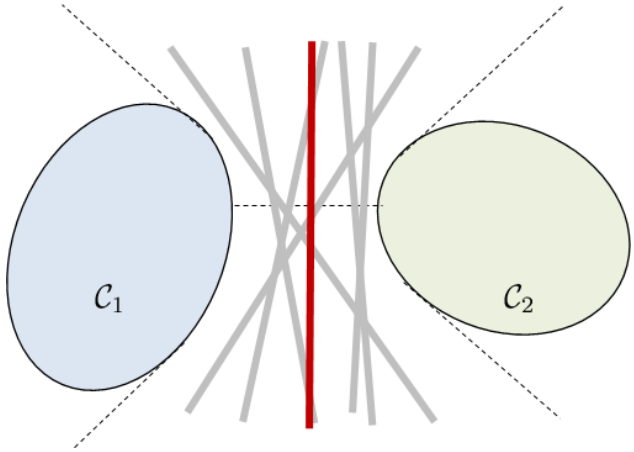


Figure 2.9: If the two classes are closed and convex, there exists other possible separating hyperplanes. However, these alternative separating hyperplanes are less robust to perturbation of the data, either through noise or adversarial attack.

Multi-Class Linear Discriminant Function

Generalizing a two-class linear discrimination to multi-class requires some additional decision rules. The two most commonly used rules are (i) **one-vs-one** rule, and (ii) **one-vs-all rule**.

One-vs-One. In one-vs-one rule, the classifier makes a pairwise comparison. For a problem with k classes, this amounts to $n = k(k-1)/2$ number of comparisons (which comes from $\binom{k}{2}$). More precisely, if for every class we define a discriminant function g_i for $i = 1, \ldots, k$, then we need to check the sign of every pair:

$$g_{ij}(\boldsymbol{x}) = g_i(\boldsymbol{x}) - g_j(\boldsymbol{x}) \gtrless 0.$$

The indices i and j go from $1, \ldots, k$, and $i \neq j$. Therefore, altogether we have $n = k(k-1)/2$ comparisons. When making decision, we check how many of these n discriminant functions has $g_{ij}(\boldsymbol{x}) > 0$, and the majority wins.

Figure 2.10 illustrates the concept of one-vs-one decision. Since we are making a pairwise comparison, the discriminant function (aka the hyperplane) can pass through the support of a class. For example $\mathcal{H}_{12}$ is the hyperplane separating $\mathcal{C}_1$ and $\mathcal{C}_2$ but it passes through $\mathcal{C}_3$.

One-vs-one rule has a potential problem that there will be undetermined regions. For example, the centerpiece of Figure 2.10 is a region where there is no majority win. In fact, as long as the three discriminant functions do not intersect at the same point, it is always possible to construct an undetermined region.

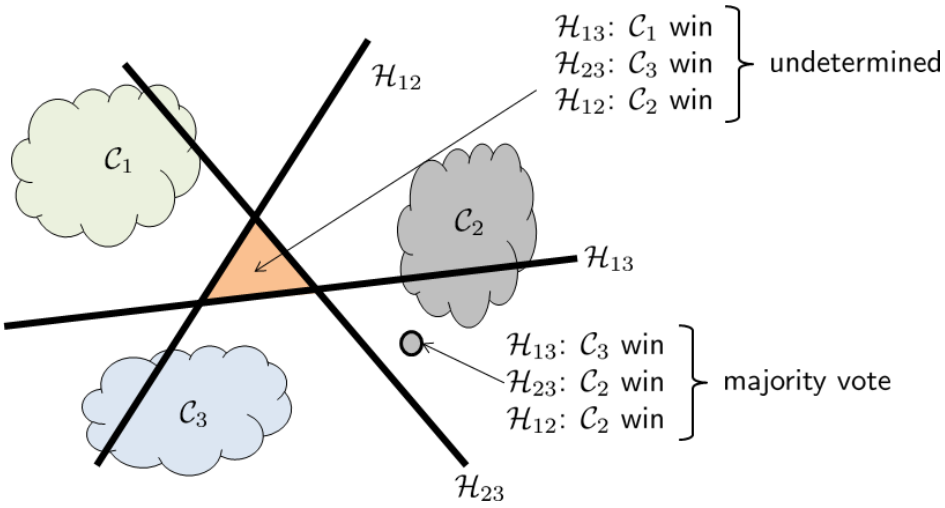


Figure 2.10: one-vs-one rule

One-vs-All. The one-vs-all rule is simpler than one-vs-one, as it check whether a data point $\boldsymbol{x}$ lives on the positive side of $g_i(\boldsymbol{x})$ or negative side of $g_i(\boldsymbol{x})$. That is, we simply check

$$g_i(\boldsymbol{x}) \gtrless 0. \tag{2.11}$$

Therefore, for a k-class problem there are only k separating hyperplanes, one for each class. It is called one-vs-all because it literally checks $\boldsymbol{x} \in \mathcal{C}_i$ versus $\boldsymbol{x} \notin \mathcal{C}_i$.

Because of the way we define one-vs-all, the decision is undetermined whenever there are more than two classes. For example, in Figure 2.11, the large triangular region on the top is undetermined because it is allowed by both $\mathcal{C}_1$ and $\mathcal{C}_3$. Similarly, the center triangle is also ambiguous because it is not allowed by any of $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$.

Unlike one-vs-one, the discriminant functions g_i do not pass through the classes. If it ever passes through a class, then the class being passed through will be divided into two halves, and one of which will be misclassified.

Linear Machine. In order to avoid the ambiguity caused by one-vs-one or one-vs-all, certain classes of multi-class classifier adopt a **linear machine** structure, where the decision is based on

$$g_i(\boldsymbol{x}) > g_j(\boldsymbol{x}) \qquad \text{for all} \quad j \neq i,$$

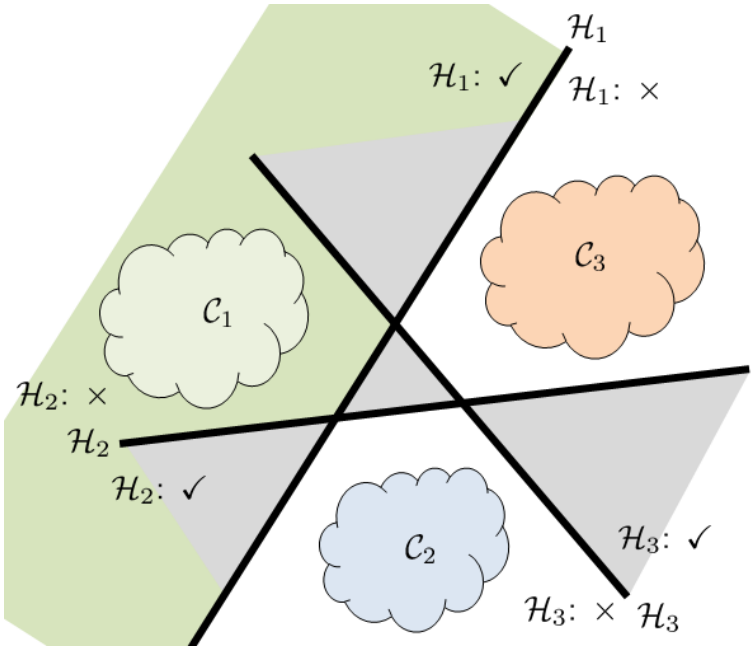


Figure 2.11: One-vs-all rule: The gray regions are the undetermined region because there are more than two discriminant functions being positive.

or more compactly

$$\max_{j \neq i} \{g_j(\boldsymbol{x})\} - g_i(\boldsymbol{x}) < 0, \tag{2.12}$$

as the maximum of $g_j(\boldsymbol{x})$ over all $j \neq i$ ensures that all $g_j(\boldsymbol{x})$ is less than $g_i(\boldsymbol{x})$. Linear machine avoids the ambiguity because the maximum is **point-wise**. That is, if $\boldsymbol{x}$ is moved to another position, the maximum will be different. In one-vs-one and one-vs-all, when $\boldsymbol{x}$ is moved we are still checking whether the discriminant function is positive or negative. There is no relative difference across the classes.

In a linear machine, the decision boundaries partition the space into k classes. There is no overlap between the classes, and there is no undetermined regions. See Figure 2.12 for an illustration. If g_i is linear, i.e., $g_i(\boldsymbol{x}) = \boldsymbol{w}_i^T \boldsymbol{x} + w_{i0}$, then we can show that each decision region is a convex set.

Theorem 3 (Convexity of decision regions)**.** *Consider a linear machine where* $g_i(\boldsymbol{x}) = \boldsymbol{w}_i^T \boldsymbol{x} + w_{i0}$*. Then, each decision region is a convex set.*

Proof. We note that $g_i(\boldsymbol{x}) > g_j(\boldsymbol{x})$ for $j \neq i$ is equivalent to

$$(\boldsymbol{w}_j - \boldsymbol{w}_i)^T \boldsymbol{x} + (w_{j0} - w_{i0}) < 0$$

Defining $\boldsymbol{a}_j = \boldsymbol{w}_j - \boldsymbol{w}_i$ and $b_j = -(w_{j0} - w_{i0})$, the decision is equivalent to

$$\boldsymbol{a}_j^T \boldsymbol{x} < b_j, \qquad \text{for all} \quad j \neq i,$$

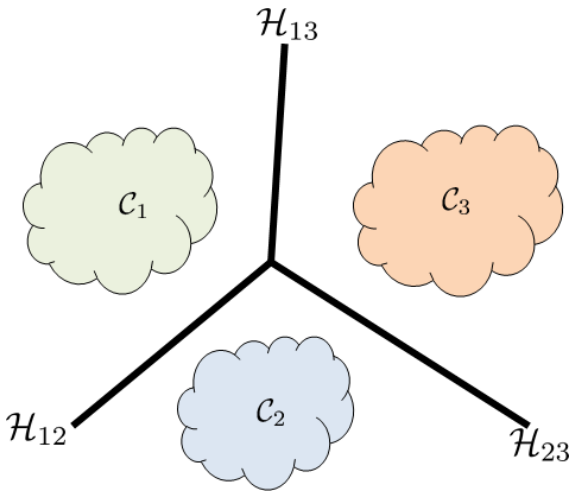


Figure 2.12: A linear machine partitions the decision space into k non-overlapping regions.

or in other words $\boldsymbol{A}^T\boldsymbol{x} < \boldsymbol{b}$, where $\boldsymbol{A} = [\boldsymbol{a}_1, \ldots, \boldsymbol{a}_k]$ contains all the $(k-1)$ columns of $\boldsymbol{a}_j$'s. When we say the decision region is convex, we say that the set $\mathcal{C}_i = \{\boldsymbol{x} \mid \boldsymbol{A}^T\boldsymbol{x} < \boldsymbol{b}\}$ is a convex set. Clearly, $\mathcal{C}_i$ is convex because $\boldsymbol{A}^T\boldsymbol{x} < \boldsymbol{b}$ is a polytope which is convex. □

Exercise 1.2. Consider the set $\mathcal{C} = \{\boldsymbol{x} \mid \boldsymbol{A}^T\boldsymbol{x} < \boldsymbol{b}\}$. Show that $\mathcal{C}$ is convex. That is, show that

$$\lambda\boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2 \in \mathcal{C}$$

for any $\boldsymbol{x}_1 \in \mathcal{C}$, $\boldsymbol{x}_2 \in \mathcal{C}$, and $0 \leq \lambda \leq 1$.

Unallowed Linear Classifiers. Is it ever possible to construct a linear classifier that occupies two sides of the decision space, such as the one shown in Figure 2.13? Unfortunately (fortunately) the answer is no. By construction of a linear classifier, the discriminant function divides the decision space into two half spaces. If there is a third half space, there must be contradiction. For example, the leftmost $\mathcal{C}_1$ of Figure 2.13 has $g_1(\boldsymbol{x}) > g_2(\boldsymbol{x})$, and so anything on the right of $\mathcal{H}_{12}$ is classified as NOT $\mathcal{C}_1$. However, the rightmost $\mathcal{C}_1$ requires that anything on the right of $\mathcal{H}'_{12}$ being $\mathcal{C}_1$, which violates the decision made by $\mathcal{H}_{12}$.

So how to construct a classifier that allows the same decision for more than one half space? The simplest answer is to go with non-linear discriminant functions. For example, the classifier shown in Figure 2.14 shows a classifier using a nonlinear discriminant function $g(\boldsymbol{x})$. When $g(\boldsymbol{x}) > 0$, the decision region contains two half spaces on the two sides of the decision space. When $g(\boldsymbol{x}) < 0$, the decision region is the middle region. In the special case of quadratic classifiers, the discriminant function is

$$g(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{W}\boldsymbol{x} + \boldsymbol{w}^T\boldsymbol{x} + w_0, \tag{2.13}$$

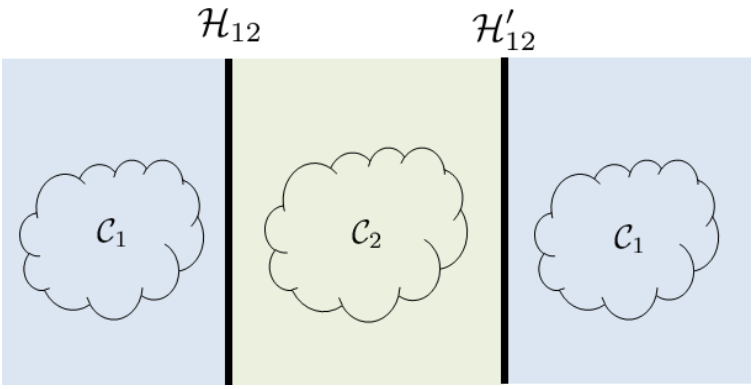


Figure 2.13: A linear classifier is not allowed to have the same decision on both sides of the decision space

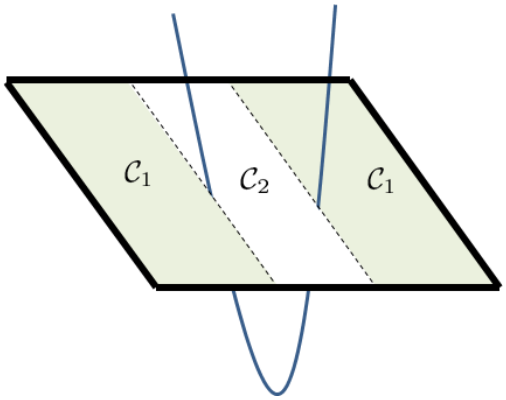


Figure 2.14: A nonlinear classifier can have the same decision on both sides of the decision space

where the geometry is determined by the eigenvalues of the matrix $\boldsymbol{W}$.

Alternatively, it is also possible to keep the linearity of g by applying a nonlinear transformation to the input vector $\boldsymbol{x}$, e.g., $\phi(\boldsymbol{x})$ for some function ϕ. The corresponding discriminant function is then

$$g(\boldsymbol{x}) = \boldsymbol{w}^T \phi(\boldsymbol{x}) + w_0. \tag{2.14}$$

This is called a **generalized linear discriminant function**, as the decision boundary remains linear.

2.2 Bayesian Decision Rule (BDR)

We now discuss a few methods to construct a meaningful discriminant function. We will start from a probabilistic approach called the Bayesian decision rule (BDR), as it offers many important insight about the geometry of a classifier.

Basic Principle

Let $\boldsymbol{X} \in \mathcal{X}$ be a d-dimensional random vector representing the input, and let $Y \in \mathcal{Y}$ be a random variable denoting the class label. We assume that we have access to the joint distribution $p_{\boldsymbol{X},Y}(\boldsymbol{x}, y)$, and so we can write down the conditional distribution $p_{\boldsymbol{X}|Y}(\boldsymbol{x}|i)$, the prior distribution $p_Y(i)$, and the marginal distribution $p_{\boldsymbol{X}}(\boldsymbol{x})$. Then, by Bayes Theorem, the posterior distribution of Y given $\boldsymbol{X}$ is

$$p_{Y\,|\,\boldsymbol{X}}(i|\boldsymbol{x}) = \frac{p_{\boldsymbol{X}\,|\,Y}(\boldsymbol{x}|i)p_Y(i)}{p_{\boldsymbol{X}}(\boldsymbol{x})}, \tag{2.15}$$

The Bayes decision rule states that among the k classes, we should decide class $\mathcal{C}_{i^*}$ if the corresponding posterior is the largest. That is,

$$p_{Y\,|\,\boldsymbol{X}}(i^*|\boldsymbol{x}) \geq p_{Y\,|\,\boldsymbol{X}}(j|\boldsymbol{x}), \qquad \text{for all} \quad j \neq i^*. \tag{2.16}$$

To determine the optimal class label, we solve

$$\begin{aligned} i^* = \underset{i}{\text{argmax}}\ p_{Y|\boldsymbol{X}}(i|\boldsymbol{x}) &= \underset{i}{\text{argmax}}\ \frac{p_{\boldsymbol{X}|Y}(\boldsymbol{x}|i)p_Y(i)}{p_{\boldsymbol{X}}(\boldsymbol{x})}, \qquad \text{let} \quad \pi_i \overset{\text{def}}{=} p_Y(i) \\ &\overset{(a)}{=} \underset{i}{\text{argmax}}\ \log p_{\boldsymbol{X}\,|\,Y}(\boldsymbol{x}|i) + \log \pi_i - \log p_{\boldsymbol{X}}(\boldsymbol{x}) \\ &\overset{(b)}{=} \underset{i}{\text{argmax}}\ \log p_{\boldsymbol{X}\,|\,Y}(\boldsymbol{x}|i) + \log \pi_i. \end{aligned}$$

In the above derivation, (a) holds because $-\log(\cdot)$ is monotonically decreasing so that the maximizer is unchanged. The last term $\log p_{\boldsymbol{X}}(\boldsymbol{x})$ in (b) is dropped because it is independent of the maximizer.

Example. (Bayesian Decision Rule for Gaussian) Consider a multi-dimensional Gaussian

$$p_{\boldsymbol{X}\,|\,Y}(\boldsymbol{x}\,|\,i) = \frac{1}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}_i|}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right\}. \tag{2.17}$$

The Bayes decision is

$$\begin{aligned} i^* &= \underset{i}{\text{argmax}}\ \log p_{\boldsymbol{X}\,|\,Y}(\boldsymbol{x}\,|\,i) + \log \pi_i \\ &= \underset{i}{\text{argmax}}\ -\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i) - \frac{d}{2}\log(2\pi) - \frac{1}{2}\log|\boldsymbol{\Sigma}_i| + \log \pi_i \\ &= \underset{i}{\text{argmax}}\ -\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i) - \frac{1}{2}\log|\boldsymbol{\Sigma}_i| + \log \pi_i. \end{aligned} \tag{2.18}$$

The result above shows some fundamental ingredient of a Bayes decision rule. First, it requires $\boldsymbol{X}$ and Y to have a **distribution**. For $\boldsymbol{X}$, the distribution is the **likelihood**

function $p_{\boldsymbol{X}|Y}(\boldsymbol{x}|i)$, and for Y, the distribution is the **prior** $p_Y(i)$. Therefore, we make an assumption that the data can be explained probabilistically according to the distributions. Depending what distribution we choose, there could be gap between the model and the data. Therefore, BDR requires a good sense of the distribution.

The other ingredient of the Bayes decision rule is the optimality criteria. For Bayes, we claim that the optimal class is determined by maximizing the posterior distribution. However, maximizing the posterior only guarantees optimality at the peak of the distribution (thus giving the name "maximum"-a-posteriori). There are other possible ways of defining optimality, e.g., picking an i that minimizes the squared error of $(Y-i)^2$ given $\boldsymbol{X}$, i.e., $\mathbb{E}[(Y-i)^2|\boldsymbol{X}]$. Such decision is called the **minimum mean squared error** (MMSE) decision. (Readers interested in this aspect can consult classic textbooks on Estimation Theory.) Therefore, depending on which optimality criteria we choose, we will land on different decision rules.

Linear Discriminant Function for Bayes

If we adopt the Bayes decision for a linear machine, then it follows immediately that the discriminant function is the following.

Definition 3. *The discriminant function for a Bayes decision is*

$$g_i(\boldsymbol{x}) = \log p_{\boldsymbol{X}|Y}(\boldsymbol{x}|i) + \log \pi_i, \tag{2.19}$$

where $p_{\boldsymbol{X}|Y}(\boldsymbol{x}|i)$ is the likelihood, and $\pi_i = p_Y(i)$ is prior.

For multi-dimensional Gaussian, the discriminant function is

$$g_i(\boldsymbol{x}) \stackrel{\text{def}}{=} -\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i) - \frac{1}{2}\log|\boldsymbol{\Sigma}_i| + \log\pi_i. \tag{2.20}$$

To decide a class, the Bayes decision seeks i^* such that

$$\max_{j\neq i^*}\{g_j(\boldsymbol{x})\} - g_{i^*}(\boldsymbol{x}) \leq 0. \tag{2.21}$$

Figure 2.15 below shows a pictorial illustration of a 1D Gaussian classification problem. Consider two classes of data $\mathcal{C}_1$ and $\mathcal{C}_2$, each being a Gaussian of different mean and variance. When given a testing data point x, the Bayesian decision rule computes the posterior $p_{Y|X}(i|x)$ to determine

$$p_{Y|X}(1|x) \gtrless_{\mathcal{C}_2}^{\mathcal{C}_1} p_{Y|X}(2|x).$$

By expressing the Gaussians, we can show that this is equivalent to

$$\frac{\pi_1}{\sqrt{2\pi\sigma_1^2}}e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} \gtrless_{\mathcal{C}_2}^{\mathcal{C}_1} \frac{\pi_2}{\sqrt{2\pi\sigma_2^2}}e^{-\frac{(x-\mu_2)^2}{2\sigma_2^2}}.$$

Taking negative logs and moving around the terms we can show that the decision is simplified to

$$-\frac{(x-\mu_1)^2}{2\sigma_1^2} - \log\sigma_1 + \log\pi_1 \gtrless_{\mathcal{C}_2}^{\mathcal{C}_1} -\frac{(x-\mu_2)^2}{2\sigma_2^2} - \log\sigma_2 + \log\pi_2. \tag{2.22}$$

For both analysis and practical implementation, we would like to express Equation (2.22)

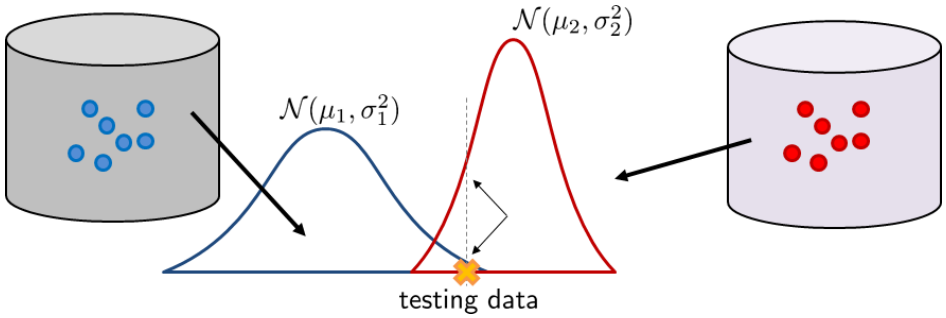


Figure 2.15: Pictorial illustration of a Bayesian decision rule. Given two classes $\mathcal{C}_1$ and $\mathcal{C}_2$, the method computes the posterior probability of a testing data point x and check if it has a higher probability for $\mathcal{C}_1$ or $\mathcal{C}_2$. The decision is equivalent to checking whether x is on the left hand side or right hand side of a cutoff threshold.

by comparing x against certain threshold. The following exercise shows how this can be done.

Exercise 2.1. Assume $\sigma_1 = \sigma_2 = \sigma$, whow that the result of Equation (2.22) can be written as

$$x \quad \gtrless_{\mathcal{C}_2}^{\mathcal{C}_1} \quad \frac{\mu_1+\mu_2}{2} - \frac{\sigma^2}{\mu_1-\mu_2}\log\frac{\pi_1}{\pi_2}.$$

Geometry of Bayesian Decision Rule: Gaussian Case

What is the geometry of the Bayes decision? To understand the geometry let us consider three cases: (i) $\boldsymbol{\Sigma}_i = \sigma^2\boldsymbol{I}$; (ii) $\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}$; (iii) arbitrary $\boldsymbol{\Sigma}_i$.

Case 1: $\boldsymbol{\Sigma}_i = \sigma^2\boldsymbol{I}$

In Case 1, we assume that the covariance matrices $\boldsymbol{\Sigma}_i$ are all identical, and each covariance is a scalar multiple of the identity matrix. This gives a discriminant function

$$g_i(\boldsymbol{x}) = -\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i) - \frac{1}{2}\log|\boldsymbol{\Sigma}| + \log\pi_i. \tag{2.23}$$

Theorem 4. *If $\boldsymbol{\Sigma}_i = \sigma^2 \boldsymbol{I}$, then the discriminant function g_i is given by*

$$g_i(\boldsymbol{x}) = \boldsymbol{w}_i^T \boldsymbol{x} + w_{i0}, \tag{2.24}$$

where

$$\boldsymbol{w}_i = \frac{\boldsymbol{\mu}_i}{\sigma^2}, \qquad \text{and} \qquad w_{i0} = -\frac{\|\boldsymbol{\mu}_i\|^2}{2\sigma^2} + \log \pi_i. \tag{2.25}$$

Proof. Substitute $\boldsymbol{\Sigma}_i = \sigma^2 \boldsymbol{I}$ into the discriminant function, we can show that

$$\begin{aligned} g_i(\boldsymbol{x}) &= -\frac{1}{2\sigma^2}\|\boldsymbol{x} - \boldsymbol{\mu}_i\|^2 + \log \pi_i && \text{, dropped } \log|\boldsymbol{\Sigma}| \\ &= -\frac{1}{2\sigma^2}\left(\|\boldsymbol{x}\|^2 - 2\boldsymbol{x}^T\boldsymbol{\mu}_i - \|\boldsymbol{\mu}_i\|^2\right) + \log \pi_i \\ &= \left(\frac{\boldsymbol{\mu}_i}{\sigma^2}\right)^T \boldsymbol{x} - \left(\frac{\|\boldsymbol{\mu}_i\|^2}{2\sigma^2} + \log \pi_i\right) && \text{, dropped } \|\boldsymbol{x}\|^2. \end{aligned}$$

Here, we dropped terms that do not affect the decision as these terms appear on both sides of $g_i(\boldsymbol{x}) > g_j(\boldsymbol{x})$. □

For any $\boldsymbol{x}$, the decision boundary between class i and class j is a **separating hyperplane** such that $g_i(\boldsymbol{x}) = g_j(\boldsymbol{x})$. That is, $\mathcal{H}_{ij} \stackrel{\text{def}}{=} \{\boldsymbol{x} \,|\, g_i(\boldsymbol{x}) = g_j(\boldsymbol{x})\}$. The equation of the separating hyperplane is given by the following corollary.

Corollary 1. *If $\boldsymbol{\Sigma}_i = \sigma^2 \boldsymbol{I}$, then the separating hyperplane is given by*

$$g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0 = 0, \tag{2.26}$$

where

$$\boldsymbol{w} = \frac{\boldsymbol{\mu}_i - \boldsymbol{\mu}_j}{\sigma^2}, \qquad \text{and} \qquad w_0 = -\frac{\|\boldsymbol{\mu}_i\|^2 - \|\boldsymbol{\mu}_j\|^2}{2\sigma^2} + \log \frac{\pi_i}{\pi_j}. \tag{2.27}$$

Proof. The separating hyperplane $\mathcal{H}_{ij}$ is

$$\begin{aligned} g_i(\boldsymbol{x}) - g_j(\boldsymbol{x}) &= (\boldsymbol{w}_i - \boldsymbol{w}_j)^T \boldsymbol{x} + (w_{i0} - w_{j0}) \\ &= \left(\frac{\boldsymbol{\mu}_i - \boldsymbol{\mu}_j}{\sigma^2}\right)^T \boldsymbol{x} - \left(\frac{\|\boldsymbol{\mu}_i\|^2}{2\sigma^2} - \frac{\|\boldsymbol{\mu}_j\|^2}{2\sigma^2}\right) + \log \frac{\pi_i}{\pi_j} \\ &= \left(\frac{\boldsymbol{\mu}_i - \boldsymbol{\mu}_j}{\sigma^2}\right)^T \left[\boldsymbol{x} - \frac{\boldsymbol{\mu}_i + \boldsymbol{\mu}_j}{2} + \sigma^2 \left(\log \frac{\pi_i}{\pi_j}\right) \frac{\boldsymbol{\mu}_i - \boldsymbol{\mu}_j}{\|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|^2}\right] \end{aligned}$$

Therefore, by defining

$$\boldsymbol{w} = \frac{\boldsymbol{\mu}_i - \boldsymbol{\mu}_j}{\sigma^2}, \qquad \text{and} \qquad \boldsymbol{x}_0 = \frac{\boldsymbol{\mu}_i + \boldsymbol{\mu}_j}{2} - \frac{\sigma^2}{\|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|^2}\left(\log \frac{\pi_i}{\pi_j}\right)(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j), \tag{2.28}$$

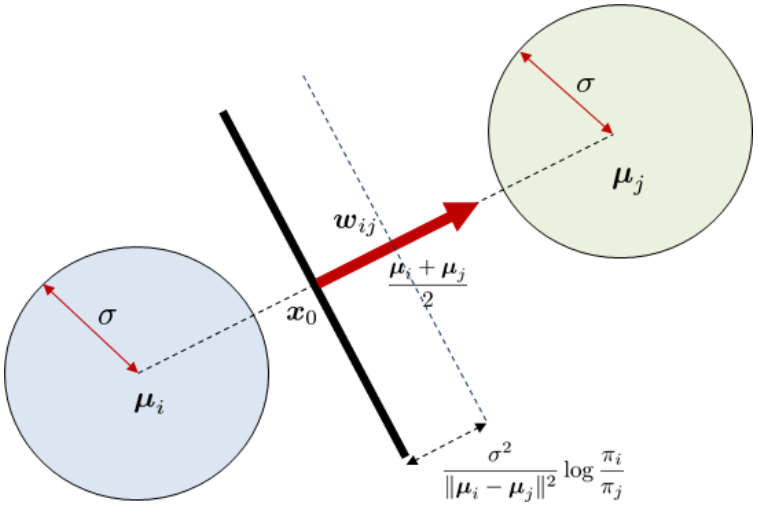


Figure 2.16: Geometry of a Bayes classifier when $\boldsymbol{\Sigma}_i = \sigma^2\boldsymbol{I}$.

the separating hyperplane is equivalent to $\mathcal{H}_{ij} = \big\{\boldsymbol{x} \,|\, \boldsymbol{w}^T(\boldsymbol{x}-\boldsymbol{x}_0) = 0\big\}$. □

Geometrically, $\boldsymbol{w}_{ij} = \boldsymbol{\mu}_i - \boldsymbol{\mu}_j$ specifies the line passing through the centers of the two classes $\boldsymbol{\mu}_i$ and $\boldsymbol{\mu}_j$. Thus, $\boldsymbol{w}_{ij}^T(\boldsymbol{x}-\boldsymbol{x}_0) = 0$ gives the normal of the line. The point $\boldsymbol{x}_0$ specifies the location of the normal. The first term $(\boldsymbol{\mu}_i + \boldsymbol{\mu}_j)/2$ is the mid-point between $\boldsymbol{\mu}_i$ and $\boldsymbol{\mu}_j$. When $\pi_i = \pi_j$ so that $\log \pi_i/\pi_j = 0$, the location is $\boldsymbol{x}_0 = (\boldsymbol{\mu}_i + \boldsymbol{\mu}_j)/2$ as there is no bias towards either class. As the ratio π_i/π_j changes, $\boldsymbol{x}_0$ will move along the line $(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)$. The amount of the displacement is $\frac{\sigma^2}{\|\boldsymbol{\mu}_i-\boldsymbol{\mu}_j\|^2}\log \pi_i/\pi_j$. Intuitively, the displacement scales with the relative emphasis of π_i and π_j: If π_i dominates, then $\log \pi_i/\pi_j$ is large and so $\boldsymbol{x}_0$ is shifted towards class 1. The displacement also scales with the reciprocal of $\|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|^2/\sigma^2$. If the two Gaussians are close, then for a positive $\log \pi_i/\pi_j$ the intercept $\boldsymbol{x}_0$ is shifted more towards class 1. If $\log \pi_i/\pi_j$ becomes negative, then $\boldsymbol{x}_0$ is shifted towards class 2.

Exercise 2.2. Consider two 1D Gaussians with $X|Y \sim \mathcal{N}(0,\sigma^2)$ if $Y = 0$, and $X|Y \sim \mathcal{N}(\mu,\sigma^2)$ if $Y = 1$. Assume $p_Y(i) = \pi_i$.

(i) Suppose we are given an observation $X = x$. Prove that the Bayesian decision rule is given by

$$x \quad \underset{\mathcal{C}_1}{\overset{\mathcal{C}_0}{\lessgtr}} \quad \frac{\sigma^2}{\mu}\log\frac{\pi_0}{\pi_1} + \frac{\mu}{2} \overset{\text{def}}{=} \tau.$$

(ii) Let $h(X)$ be the hypothesis function. Define the probability of error as

$$P_e = \pi_1\mathbb{P}[h(X) = 0|Y = 1] + \pi_0\mathbb{P}[h(X) = 1|Y = 0].$$

Express P_e in terms of the standard Normal CDF $\Phi(z) = \int_{-\infty}^{z}\frac{1}{\sqrt{2\pi}}e^{-t^2/2}dt$.

Hint: Note that $h(X) = 0$ iff $X > \tau$, and $h(X) = 1$ iff $X < \tau$.

Below is a MATLAB / Python code to generate a Bayesian decision rule for $\boldsymbol{\Sigma} = \sigma^2 \boldsymbol{I}$. In this demonstration, we use the population mean and covariance to compute the discriminant function. In practice, the population mean and covariance can be replaced by the empirical mean and empirical covariance.

```
% MATLAB code to generate a decision boundary of the BDR
mu0 = [0; 0];
mu1 = [2; 1];
s   = 0.5;
Sigma = (s^2)*[1 0; 0 1];
n  = 1000;
w  = (mu1-mu0)/s^2;
x0 = (mu1+mu0)/2;
y0 = mvnrnd(mu0,Sigma,[n,1]);
y1 = mvnrnd(mu1,Sigma,[n,1]);
```

To plot the decision boundary, we need to determine the two end points in the plot unless we have some specialized plotting libraries. The two end points can be found since $\boldsymbol{w}^T(\boldsymbol{x}-\boldsymbol{x}_0) = 0$. Rewriting this equation we have $\boldsymbol{w}^T\boldsymbol{x} = \boldsymbol{w}^T\boldsymbol{x}_0$. Now, consider a two-dimensional case so that $d = 2$. Then, $\boldsymbol{w}^T\boldsymbol{x} = \boldsymbol{w}^T\boldsymbol{x}_0$ is equivalent to

$$w_1 x_1 + w_2 x_2 = \boldsymbol{w}^T \boldsymbol{x}_0.$$

If we set x_1 to a constant, then we can determine $x_2 = (\boldsymbol{w}^T\boldsymbol{x}_0 - w_1 x_1)/w_2$. Vice versa, if we set x_2 to a constant, we can determine $x_1 = (\boldsymbol{w}^T\boldsymbol{x}_0 - w_2 x_2)/w_1$. In the plot below, we set $x_1 = 5$ and $x_2 = 5$ to locate the two extreme points.

```
figure;
scatter(y0(:,1),y0(:,2),'bo'); hold on;
scatter(y1(:,1),y1(:,2),'rx');
plot([5, (w'*x0-w(2)*5)/w(1)]', [(w'*x0-w(1)*5)/w(2), 5]', ...
    , 'k', 'LineWidth', 2);
axis([-2 3.5 -2 3.5]);
set(gcf, 'Position', [100, 100, 500, 500]);
```

The plot of this example is shown in Figure 2.17. Because of the display, the aspect ratio of the x-axis and the y-axis is not 1:1. Therefore, to properly visualize the decision boundary, we can reset the aspect ratio. From there, we will be able to see the normal vector $\boldsymbol{w} = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)/\sigma^2$.

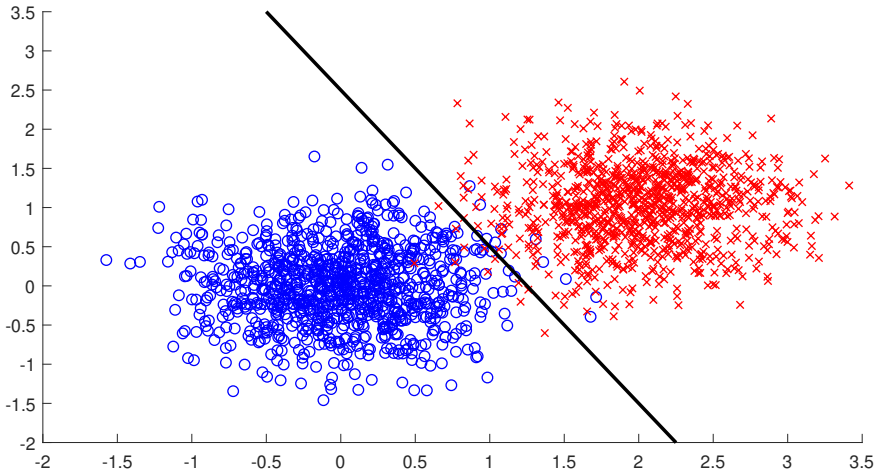


Figure 2.17: Bayesian decision rule using $\boldsymbol{\Sigma}_i = \sigma^2 \boldsymbol{I}$.

Case 2: $\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}$

In this case, we lift the assumption that $\boldsymbol{\Sigma}_i$ are scalar multiples of the identity matrix, but we still assume that all $\boldsymbol{\Sigma}_i$'s are the same. The discriminant function now becomes

$$g_i(\boldsymbol{x}) = -\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_i) - \frac{1}{2}\log|\boldsymbol{\Sigma}| + \log \pi_i. \tag{2.29}$$

Theorem 5. *If $\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}$, then the discriminant function g_i is given by*

$$g_i(\boldsymbol{x}) = \boldsymbol{w}_i^T \boldsymbol{x} + w_{i0}, \tag{2.30}$$

where

$$\boldsymbol{w}_i = \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i, \qquad \text{and} \qquad w_{i0} = -\frac{1}{2}\boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i + \log \pi_i. \tag{2.31}$$

Proof. The proof follows similarly with Case 1. We show that

$$\begin{aligned} g_i(\boldsymbol{x}) &= -\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_i) + \log \pi_i \\ &= -\frac{1}{2}\left(\boldsymbol{x}^T \boldsymbol{\Sigma}^{-1}\boldsymbol{x} - 2\boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1}\boldsymbol{x} + \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i\right) + \log \pi_i \\ &= \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1}\boldsymbol{x} - \frac{1}{2}\boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i + \log \pi_i \qquad \text{, dropped} \quad \boldsymbol{x}^T \boldsymbol{\Sigma}^{-1}\boldsymbol{x}. \end{aligned}$$

where terms irrelevant to the decision are dropped. □

The geometry of the decision boundary can be analyzed by comparing $g_i(\boldsymbol{x}) - g_j(\boldsymbol{x})$:

$$\begin{aligned} g_i(\boldsymbol{x}) - g_j(\boldsymbol{x}) &= (\boldsymbol{w}_i - \boldsymbol{w}_j)^T \boldsymbol{x} + (w_{i0} - w_{j0}) \\ &= \left[\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)\right]^T \boldsymbol{x} - \frac{1}{2}\left(\boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i - \boldsymbol{\mu}_j^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_j\right) + \log \frac{\pi_i}{\pi_j}. \end{aligned}$$

This gives the following corollary about the separating hyperplane $\mathcal{H}_{ij} = \{\boldsymbol{x} \mid \boldsymbol{w}^T \boldsymbol{x} + w_0 = 0\}$:

Corollary 2. *If $\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}$, then the separating hyperplane is given by*

$$g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0 = 0, \tag{2.32}$$

where

$$\boldsymbol{w} = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j), \qquad \text{and} \qquad w_0 = -\frac{1}{2}\left(\boldsymbol{\mu}_i + \boldsymbol{\mu}_j\right)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j) + \log \frac{\pi_i}{\pi_j}. \tag{2.33}$$

From this corollary, if we define

$$\boldsymbol{w} = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j), \quad \text{and} \quad \boldsymbol{x}_0 = \frac{\boldsymbol{\mu}_i + \boldsymbol{\mu}_j}{2} - \frac{\log \frac{\pi_i}{\pi_j}}{(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j). \tag{2.34}$$

then, we have $g_i(\boldsymbol{x}) - g_j(\boldsymbol{x}) = \boldsymbol{w}^T(\boldsymbol{x} - \boldsymbol{x}_0)$. Clearly, if $\boldsymbol{\Sigma} = \sigma^2 \boldsymbol{I}$, we will obtain the exact same result as Case 1.

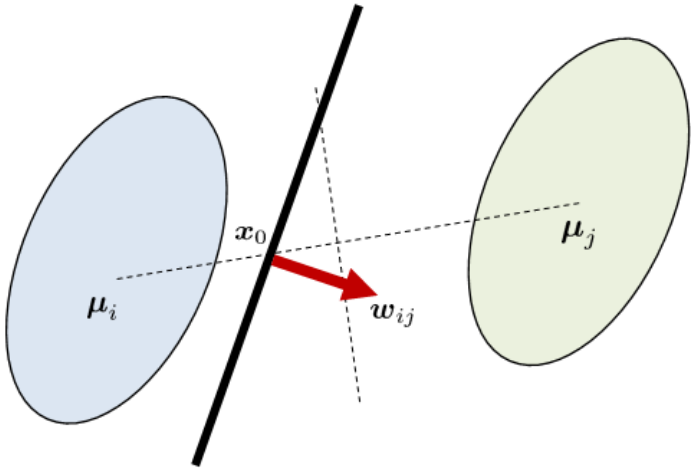


Figure 2.18: Geometry of a Bayes decision when the Gaussians have identical covariance matrices.

The inverse covariance matrix $\boldsymbol{\Sigma}^{-1}$ changes the decision boundary, as it applies a linear transformation to the vector $\boldsymbol{\mu}_i - \boldsymbol{\mu}_j$. Such linear transformation is needed, because the covariance matrix changes the shape of the two Gaussians. As a result, the decision boundary should adjust its orientation in order to match the underlying Gaussians.

Case 3: Arbitrary $\boldsymbol{\Sigma}_i$

In the most general setting, the discriminant function takes the form of

$$g_i(\boldsymbol{x}) = -\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i) - \frac{1}{2}\log|\boldsymbol{\Sigma}_i| + \log\pi_i. \tag{2.35}$$

Theorem 6. *If $\boldsymbol{\Sigma}_i$ is an arbitrary positive semi-definite matrix, then*

$$g_i(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{W}_i\boldsymbol{x} + \boldsymbol{w}_i^T\boldsymbol{x} + w_{i0}, \tag{2.36}$$

where

$$\boldsymbol{W}_i = \boldsymbol{\Sigma}_i^{-1}, \quad \boldsymbol{w}_i = -\boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i, \quad \textit{and} \quad w_{i0} = \frac{1}{2}\boldsymbol{\mu}_i^T\boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i + \frac{1}{2}\log|\boldsymbol{\Sigma}_i| - \log\pi_i.$$

Proof. The result follows from the fact that

$$\begin{aligned} g_i(\boldsymbol{x}) &= \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i) + \frac{1}{2}\log|\boldsymbol{\Sigma}_i| - \log\pi_i && \text{, changed to positive} \\ &= \frac{1}{2}\boldsymbol{x}^T\boldsymbol{\Sigma}_i^{-1}\boldsymbol{x} - \boldsymbol{\mu}_i^T\boldsymbol{\Sigma}_i^{-1}\boldsymbol{x} + \frac{1}{2}\boldsymbol{\mu}_i^T\boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i + \frac{1}{2}\log|\boldsymbol{\Sigma}_i| - \log\pi_i, \end{aligned}$$

where we flipped the sign of g_i as it does not affect the decision. □

If we look at the decision boundary, we note that the decision boundary is given by

$$g_i(\boldsymbol{x}) - g_j(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T(\boldsymbol{W}_i - \boldsymbol{W}_j)\boldsymbol{x} + (\boldsymbol{w}_i - \boldsymbol{w}_j)^T\boldsymbol{x} + (w_{i0} - w_{j0}). \tag{2.37}$$

As a result, the geometry of the decision boundary is subject to the positive definiteness of $\boldsymbol{W} \overset{\text{def}}{=} \boldsymbol{W}_i - \boldsymbol{W}_j$. If $\boldsymbol{W}$ is positive definite, then the decision boundary will be an ellipsoid. If $\boldsymbol{W}$ has one positive eigenvalue and one negative eigenvalue, then the decision boundary will be a hyperbolic paraboloid. Depending also on the magnitude of the eigenvalues, the ellipse can be simplified to a sphere. Several examples are shown in Figure 2.19.

Exercise 2.3 Consider two inverse covariance matrices

$$\boldsymbol{W}_1 = \begin{bmatrix} 4 & 2 \\ 2 & 3 \end{bmatrix} \quad \text{and} \quad \boldsymbol{W}_2 = \begin{bmatrix} 5 & 1 \\ 1 & 2 \end{bmatrix}. \tag{2.38}$$

(i) Compute the eigenvalues of $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ to show that both are positive semi-definite.

(ii) Show that $\boldsymbol{W}_1 - \boldsymbol{W}_2$ is symmetric. Does symmetry hold for all $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$? That is, if $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are both symmetric, is $\boldsymbol{W}_1 - \boldsymbol{W}_2$ also symmetric?

(iii) Show that $\boldsymbol{W}_1 - \boldsymbol{W}_2$ is neither positive semi-definite or negative semi-definite.

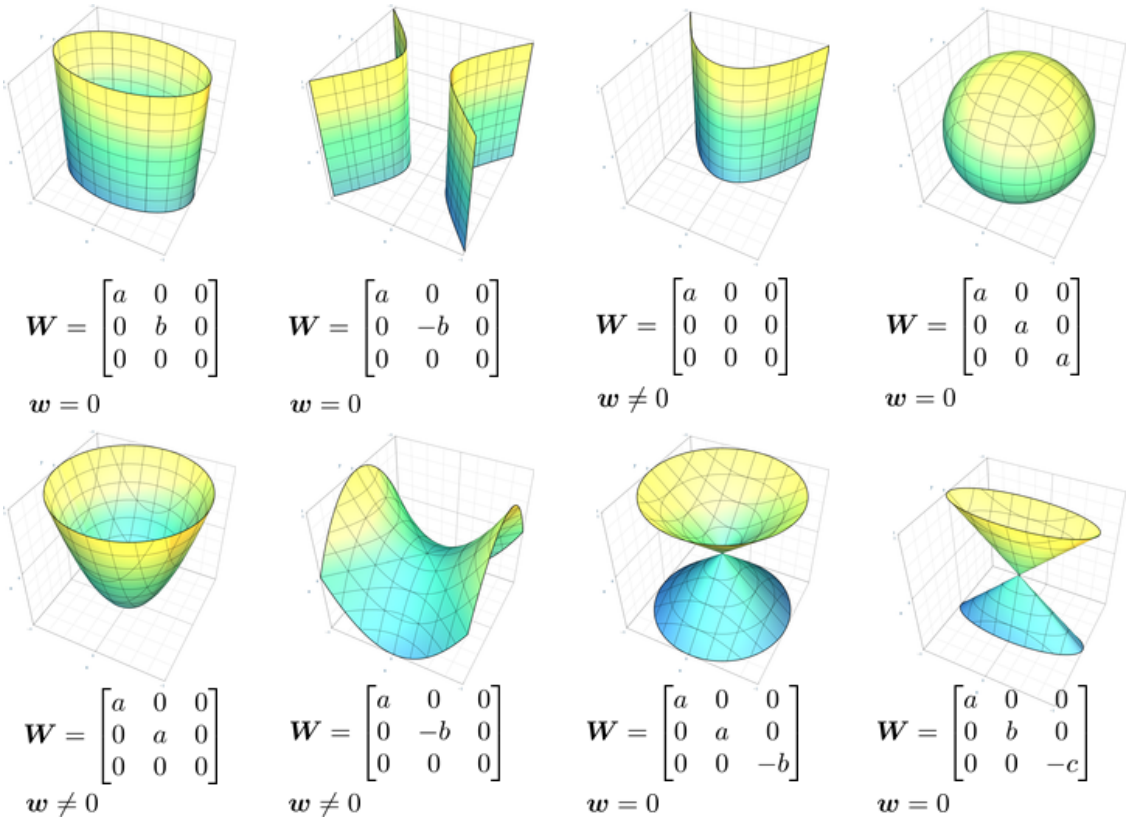


Figure 2.19: Geometry of a Bayes decision when the Gaussians are arbitrary. In this figure, $\boldsymbol{W} \stackrel{\text{def}}{=} \boldsymbol{W}_i - \boldsymbol{W}_j$, and $\boldsymbol{w} = \boldsymbol{w}_i - \boldsymbol{w}_j$. The contour plots are taken from Wikipedia `https://en.wikipedia.org/wiki/Quadric`.

Maximum Likelihood

After discussing the geometry of the discriminant function, we should now ask a practical problem: Given a dataset $\mathcal{D} = \{(\boldsymbol{x}_j, y_j)\}_{j=1}^n$, how can we determine the model parameters $\{(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\}_{i=1}^k$? This parameter estimation problem is the training phase of the Bayesian decision rule, for without such parameter estimation we will not be able to build the models.

For simplicity, let us assume that $\mathcal{D}$ is partitioned into k non-overlapping subsets $\mathcal{D}_1, \ldots, \mathcal{D}_k$. Each $\mathcal{D}_i = \{(\boldsymbol{x}_j^{(i)}, y_j^{(i)})\}_{j=1}^n$ is the training set for class i, and this training set will give us the parameter $(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$. To reduce notation burden we shall drop the subscript i unless confusion arises for different classes.

Given $\mathcal{D}$, our goal is to estimate the model parameter $\boldsymbol{\theta}$. If we assume that $\mathcal{D}$ is drawn from a distribution, and $\boldsymbol{\theta}$ is its model parameter, then the most straight-forward approach is to find a $\boldsymbol{\theta}$ that maximizes the probability of generating $\mathcal{D}$:

$$\begin{aligned} \widehat{\boldsymbol{\theta}} &= \operatorname*{argmax}_{\boldsymbol{\theta}} \; p(\mathcal{D}\,;\boldsymbol{\theta}) \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}} \; \prod_{j=1}^n p(\boldsymbol{x}_j;\boldsymbol{\theta}). \end{aligned} \tag{2.39}$$

Here, the distribution $p(\mathcal{D}\,;\boldsymbol{\theta})$ means the joint distribution of all samples in $\mathcal{D}$, and this joint distribution is specified by a parameter $\boldsymbol{\theta}$. The second equality in the above equation is based on an important assumption that all the m samples in $\mathcal{D}$ are **independent**.

Let us look at one example. Suppose that we assume $\mathcal{D}$ is a multi-dimensional Gaussian with known $\boldsymbol{\Sigma}$. Then the parameter is the unknown mean: $\boldsymbol{\theta} = \boldsymbol{\mu}$. The optimization is therefore

$$\begin{aligned}\widehat{\boldsymbol{\mu}} &= \underset{\boldsymbol{\mu}}{\text{argmax}} \;\; \prod_{j=1}^{n} \frac{1}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}|}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}_j-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_j-\boldsymbol{\mu})\right\} \\ &= \underset{\boldsymbol{\mu}}{\text{argmax}} \;\; \left(\frac{1}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}|}}\right)^n \exp\left\{-\frac{1}{2}\sum_{j=1}^{n}(\boldsymbol{x}_j-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_j-\boldsymbol{\mu})\right\}. \end{aligned} \tag{2.40}$$

Now we can apply a useful trick (again!): Since log is monotonically increasing, we can apply log to the right hand side and it will not affect the maximizer. Further, if we take $-\log$ we can flip the maximum to minimum. Therefore, we can show that

$$\widehat{\boldsymbol{\mu}} = \underset{\boldsymbol{\mu}}{\text{argmin}} \;\; \frac{1}{2}\sum_{j=1}^{n}(\boldsymbol{x}_j-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_j-\boldsymbol{\mu}). \tag{2.41}$$

Taking the derivative and setting to zero, we can show that

$$\nabla_{\boldsymbol{\mu}}\left\{\frac{1}{2}\sum_{j=1}^{n}(\boldsymbol{x}_j-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_j-\boldsymbol{\mu})\right\} = \sum_{j=1}^{n}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_j-\boldsymbol{\mu}) = \mathbf{0}.$$

Moving the terms around, and noting that $\sum_{j=1}^{n}\boldsymbol{\mu} = n\boldsymbol{\mu}$, the solution is

$$\widehat{\boldsymbol{\mu}} = \frac{1}{n}\sum_{j=1}^{n}\boldsymbol{x}_j. \tag{2.42}$$

We call this solution the **maximum-likelihood** estimate of the multi-dimensional Gaussian. The implication of this result is that for a given dataset $\mathcal{D}$, the maximum-likelihood estimate of mean of a multi-dimensional Gaussian is simply the **empirical average** of the dataset.

Theorem 7 (Maximum-Likelihood Estimate for Mean)**.** *Assume $\mathcal{D}$ is generated from a multi-dimensional Gaussian with an unknown mean $\boldsymbol{\mu}$ and a fixed covariance $\boldsymbol{\Sigma}$. The* ***maximum-likelihood estimate*** *of $\boldsymbol{\mu}$ is*

$$\begin{aligned}\widehat{\boldsymbol{\mu}} &= \underset{\boldsymbol{\mu}}{\textit{argmax}} \;\; \prod_{j=1}^{n} \frac{1}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}|}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}_j-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_j-\boldsymbol{\mu})\right\} \\ &= \frac{1}{n}\sum_{j=1}^{n}\boldsymbol{x}_j. \end{aligned} \tag{2.43}$$

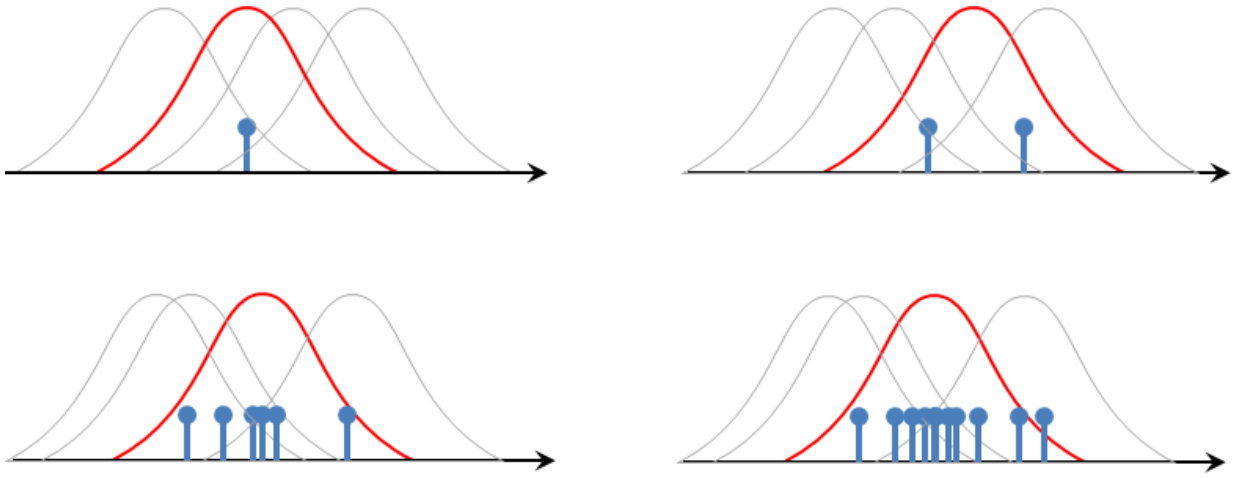


Figure 2.20: Pictorial illustration of maximum likelihood estimation.

Let us take a look at the geometric interpretation. Consider a 1D problem so that $d = 1$. By switching the mean, we will construct a family of Gaussians $\mathcal{N}(\mu, \sigma^2)$, where the running index is the mean μ. When there is only one observation ($n = 1$) so that we only have x_1, the MLE estimate of the mean is exactly the sample $\mu = x_1$. This is equivalent to ask: Among the different Gaussians, which one can maximize the probability of getting the sample x_1. Clearly, this happens when x_1 is located at the center of the Gaussian, and so the best Gaussian is $\mathcal{N}(x_1, \sigma^2)$.

Now, if we increase the number of observations to $n = 2$ so that we have x_1 and x_2, the MLE estimate of the mean is $\mu = (x_1 + x_2)/2$. This is equivalent to asking: Given x_1 and x_2, which Gaussian is the most likely one that can simultaneously give x_1 and x_2? And from our calculation, we know that the Gaussian should be centered at the mid point of x_1 and x_2, and so that best Gaussian is $\mathcal{N}((x_1 + x_2)/2, \sigma^2)$. As n increases, we eventually have $\mu = \frac{1}{n}\sum_{j=1}^{n} x_j$.

Exercise 2.3. The theorem above requires a known covariance $\boldsymbol{\Sigma}$. If we assume that both the mean $\boldsymbol{\mu}$ and the covariance $\boldsymbol{\Sigma}$ are unknown, then the maximum-likelihood estimate of $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ becomes

$$\widehat{\boldsymbol{\mu}} = \frac{1}{n}\sum_{j=1}^{n} \boldsymbol{x}_j, \quad \text{and} \quad \widehat{\boldsymbol{\Sigma}} = \frac{1}{n}\sum_{j=1}^{n} (\boldsymbol{x}_j - \boldsymbol{\mu})(\boldsymbol{x}_j - \boldsymbol{\mu})^T, \tag{2.44}$$

which are called the **sample mean**, and the **sample covariance**, respectively.

(i) Prove that $\mathbb{E}[\widehat{\boldsymbol{\Sigma}}] \neq \boldsymbol{\Sigma}$, where the expectation is taken with respect to $\boldsymbol{x}_j$. This shows the sample covariance is **biased**.

(ii) Prove that if $\widehat{\boldsymbol{\Sigma}} = \frac{1}{n-1}\sum_{j=1}^{n} (\boldsymbol{x}_j - \boldsymbol{\mu})(\boldsymbol{x}_j - \boldsymbol{\mu})^T$, then we have $\mathbb{E}[\widehat{\boldsymbol{\Sigma}}] = \boldsymbol{\Sigma}$.

As a visualization of the MLE, we demonstrate a MATLAB / Python code as below. In this example, we assume $\boldsymbol{\mu} = [0,\ 0]^T$ and $\boldsymbol{\Sigma} = \begin{bmatrix} 0.9 & 0.4 \\ 0.4 & 0.3 \end{bmatrix}$. We generate N samples where $N \in \{3, 10, 50, 100\}$, and check how close the estimate parameters are compared to the true parameter.

```
%MATLAB code to simulate the MLE as more samples are observed.
N = 1000;
mu = [0 0];
Sigma = [.9 .4; .4 .3];
Xfulldata = mvnrnd(mu, Sigma, N);

x1 = -5:0.01:5; x2 = -5:0.01:5;
[X1,X2] = meshgrid(x1,x2);
Ftrue = mvnpdf([X1(:) X2(:)],mu,Sigma);
Ftrue = reshape(Ftrue,length(x2),length(x1));

Nset = [3 10 50 100];
for idx = 1:4
    N = Nset(idx);
    X = Xfulldata(1:N, :);

    muHat    = mean(X);
    SigmaHat = cov(X);
    F = mvnpdf([X1(:) X2(:)],muHat,SigmaHat);
    F = reshape(F,length(x2),length(x1));

    figure(1);
    contour(X2,X1,Ftrue, 'r', 'LineWidth', 2);hold on;
    contour(X2,X1,F, 'b', 'LineWidth', 2);
    plot(X(:,2),X(:,1),'kx', 'LineWidth', 1.5, 'MarkerSize', 10);
    axis image;
    axis([-2 2 -2 2]);
    hold off;
    set(gcf, 'Position', [100, 100, 300, 300]);
end
```

The result of this experiment is shown in Figure 2.21. When N is small, e.g., $N = 3$, the estimated Gaussian is highly skewed and is different from the ground truth. This should not be a surprise, because for the $N = 3$ samples we have, the best estimate of the mean and the covariance will give a skewed Gaussian. As N increases, e.g., when $N = 10$, the estimated

Gaussian becomes more similar to the ground truth; And ultimately when we have enough samples, e.g., $N = 100$, the estimated Gaussian starts to overlap with the ground truth Gaussian.

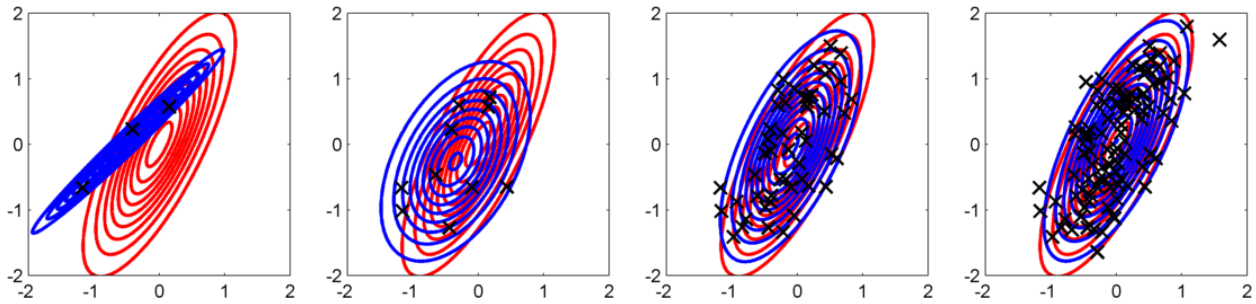


Figure 2.21: Estimating the Gaussian parameters using MLE as number of samples increases. From left to right: $N = 3$, $N = 10$, $N = 50$, and $N = 100$. As more samples become available, the estimated Gaussian becomes more similar to the ground truth Gaussian.

Maximum-a-Posteriori

While maximum-likelihood estimate is simple and straight-forward, a practical question we ask is how to improve the ML estimates when m is small? However, when m is small, there is really not much we can do because the dataset $\mathcal{D}$ is small. In this case, we need to make some prior assumptions about $\boldsymbol{\theta}$. In Bayesian terminology, we assume a $\boldsymbol{\theta}$ is sampled from a **prior** distribution $p_{\boldsymbol{\Theta}}(\boldsymbol{\theta})$, and write the posterior distribution as

$$p(\boldsymbol{\theta} \,|\, \mathcal{D}) \propto p(\mathcal{D} \,|\, \boldsymbol{\theta}) p(\boldsymbol{\theta}). \tag{2.45}$$

Let us look at the terms on the right hand side. What is the difference between $p(\mathcal{D}\,;\,\boldsymbol{\theta})$ in maximum likelihood and $p(\mathcal{D}\,|\,\boldsymbol{\theta})$ in this expression? The probability in maximum-likelihood $p(\mathcal{D}\,;\,\boldsymbol{\theta})$ assumes that $\boldsymbol{\theta}$ is a deterministic **parameter**. There is nothing random about $\boldsymbol{\theta}$. The probability we have here $p(\mathcal{D}\,|\,\boldsymbol{\theta})$ assumes that $\boldsymbol{\theta}$ is a realization of a **random variable** $\boldsymbol{\Theta}$. Thus, there is a distribution of $\boldsymbol{\theta}$, which is $p_{\boldsymbol{\Theta}}(\boldsymbol{\theta})$ or simply $p(\boldsymbol{\theta})$. Because of the prior distribution, we can talk about the posterior distribution of $\boldsymbol{\theta}$ given $\mathcal{D}$. This posterior distribution is undefined for maximum likelihood, because there is no distribution for $\boldsymbol{\theta}$.

If we take this route, then a natural solution for the estimation problem is to maximize the posterior distribution:

$$\widehat{\boldsymbol{\theta}} = \operatorname*{argmax}_{\boldsymbol{\theta}} \;\; p(\mathcal{D} \,|\, \boldsymbol{\theta}) p(\boldsymbol{\theta}) = \operatorname*{argmax}_{\boldsymbol{\theta}} \;\; \prod_{j=1}^{n} p(\boldsymbol{x}_j \,|\, \boldsymbol{\theta}) p(\boldsymbol{\theta}). \tag{2.46}$$

The solution to this problem is called the **maximum-a-posteriori** (MAP) estimation.

Let us look at a 1D example. Suppose that the observations $\{x_j\}_{j=1}^n$ are generated from a Gaussian $\mathcal{N}(\mu, \sigma^2)$ where σ is a known and fixed constant. Our goal is to estimate the unknown mean θ. We assume that θ is generated from a prior distribution $p(\theta)$ which is another Gaussian:

$$p(x|\mu) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}, \quad \text{and} \quad p(\mu) = \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{\mu^2}{2}\right\}.$$

Now, given the observations $\{x_j\}_{j=1}^n$, the MAP estimate is determined by solving the problem

$$\begin{aligned}
\widehat{\mu} &= \underset{\mu}{\text{argmax}} \quad \prod_{j=1}^n p(x_j|\mu)p(\mu) \\
&= \underset{\mu}{\text{argmax}} \quad \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^N \exp\left\{-\sum_{j=1}^n \frac{(x_j-\mu)^2}{2\sigma^2}\right\} \cdot \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{\mu^2}{2}\right\} \\
&= \underset{\mu}{\text{argmax}} \quad -\left(\sum_{j=1}^n \frac{(x_j-\mu)^2}{2\sigma^2} + \frac{\mu^2}{2}\right) \\
&= \frac{\sum_{j=1}^n x_j}{n+\sigma^2}.
\end{aligned}$$

The above equation shows that the effect of the prior $p(\mu)$ lies in the denominator of the fractions. For small n, the factor σ^2 has more influence. This can be understood as when we have few observations, we should rely more on the prior. However, as n grows, the influence of the prior reduces and we put more emphasis on the observations.

Pictorially, the presence of a prior changes the "best" Gaussian from a purely observation-based result to a experience plus observation result. An illustration is shown in Figure 2.22.

Figure 2.22: The green curve is the prior distribution $p(\mu)$. When $N = 1$, the best Gaussian is shifted from x_1 according to the prior. Similarly for $n = 2$, the best Gaussian is shifted according to the prior and the observations $x_1 + x_2$.

Now, let us generalize to a high-dimensional Gaussian. Consider $\boldsymbol{x}_j \,|\, \boldsymbol{\mu} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}_0)$, and $\boldsymbol{\mu} \sim \mathcal{N}(\boldsymbol{\mu}_\theta, \boldsymbol{\Sigma}_\theta)$. Assume that $\boldsymbol{\Sigma}_0$ is fixed and known. The interpretation of this setting is that the sample $\boldsymbol{x}_j$ has a conditional distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}_0)$. The conditional mean $\boldsymbol{\mu}$ has

its own distribution $\mathcal{N}(\boldsymbol{\mu}_\theta, \boldsymbol{\Sigma}_\theta)$. In this case, the posterior distribution can be written as

$$\begin{aligned}
\widehat{\boldsymbol{\mu}} &= \underset{\boldsymbol{\mu}}{\text{argmax}} \;\; p(\boldsymbol{\mu} \,|\, \mathcal{D}) = \underset{\boldsymbol{\mu}}{\text{argmax}} \;\; \prod_{j=1}^{n} p(\boldsymbol{x}_j \,|\, \boldsymbol{\mu}) p(\boldsymbol{\mu}) \\
&= \underset{\boldsymbol{\mu}}{\text{argmax}} \;\; \left(\prod_{j=1}^{n} \exp\left\{ -\frac{1}{2} (\boldsymbol{x}_j - \boldsymbol{\mu})^T \boldsymbol{\Sigma}_0^{-1} (\boldsymbol{x}_j - \boldsymbol{\mu}) \right\} \right) \cdot \exp\left\{ -\frac{1}{2} (\boldsymbol{\mu} - \boldsymbol{\mu}_\theta)^T \boldsymbol{\Sigma}_\theta^{-1} (\boldsymbol{\mu} - \boldsymbol{\mu}_\theta) \right\} \\
&= \underset{\boldsymbol{\mu}}{\text{argmax}} \;\; C \exp\left\{ -\frac{1}{2} \left(\boldsymbol{\mu}^T (m\boldsymbol{\Sigma}_0^{-1} + \boldsymbol{\Sigma}_\theta^{-1}) \boldsymbol{\mu} - 2\boldsymbol{\mu}^T \left(\boldsymbol{\Sigma}_0^{-1} \sum_{j=1}^{n} \boldsymbol{x}_j + \boldsymbol{\Sigma}_\theta^{-1} \boldsymbol{\mu}_\theta \right) \right) \right\} \\
&= \underset{\boldsymbol{\mu}}{\text{argmax}} \;\; C' \exp\left\{ -\frac{1}{2} (\boldsymbol{\mu} - \widetilde{\boldsymbol{\mu}})^T \widetilde{\boldsymbol{\Sigma}}^{-1} (\boldsymbol{\mu} - \widetilde{\boldsymbol{\mu}}) \right\} = \widetilde{\boldsymbol{\mu}},
\end{aligned}$$

where with some (slightly tedious) algebra we can show that

$$\begin{aligned}
\widetilde{\boldsymbol{\mu}} &= \boldsymbol{\Sigma}_\theta \left(\boldsymbol{\Sigma}_\theta + \frac{1}{n} \boldsymbol{\Sigma}_0 \right)^{-1} \left(\frac{1}{n} \sum_{j=1}^{n} \boldsymbol{x}_j \right) + \frac{1}{n} \boldsymbol{\Sigma}_0 \left(\boldsymbol{\Sigma}_\theta + \frac{1}{n} \boldsymbol{\Sigma}_0 \right)^{-1} \boldsymbol{\mu}_\theta, \\
\widetilde{\boldsymbol{\Sigma}} &= \boldsymbol{\Sigma}_\theta \left(\boldsymbol{\Sigma}_\theta + \frac{1}{n} \boldsymbol{\Sigma}_0 \right)^{-1} \frac{1}{n} \boldsymbol{\Sigma}_0.
\end{aligned} \tag{2.47}$$

Theorem 8 (Maximum-a-Posteriori Estimate for Mean)**.** *Assume $\mathcal{D}$ is generated from a multi-dimensional Gaussian with $\boldsymbol{x}_j \,|\, \boldsymbol{\mu} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}_0)$, and $\boldsymbol{\mu} \sim \mathcal{N}(\boldsymbol{\mu}_\theta, \boldsymbol{\Sigma}_\theta)$.* ***maximum-a-posteriori estimate*** *of $\boldsymbol{\mu}$ is*

$$\begin{aligned}
\widehat{\boldsymbol{\mu}} &= \underset{\boldsymbol{\mu}}{\textit{argmax}} \;\; \prod_{j=1}^{n} p(\boldsymbol{x}_j \,|\, \boldsymbol{\mu}) p(\boldsymbol{\mu}) \\
&= \boldsymbol{\Sigma}_\theta \left(\boldsymbol{\Sigma}_\theta + \frac{1}{n} \boldsymbol{\Sigma}_0 \right)^{-1} \left(\frac{1}{n} \sum_{j=1}^{n} \boldsymbol{x}_j \right) + \frac{1}{n} \boldsymbol{\Sigma}_0 \left(\boldsymbol{\Sigma}_\theta + \frac{1}{n} \boldsymbol{\Sigma}_0 \right)^{-1} \boldsymbol{\mu}_\theta.
\end{aligned} \tag{2.48}$$

Can we say something about the Theorem? If we let $m \to \infty$, we can show that

$$\lim_{n\to\infty} \left\{ \boldsymbol{\Sigma}_\theta \left(\boldsymbol{\Sigma}_\theta + \frac{1}{n} \boldsymbol{\Sigma}_0 \right)^{-1} \left(\frac{1}{n} \sum_{j=1}^{n} \boldsymbol{x}_j \right) + \frac{1}{n} \boldsymbol{\Sigma}_0 \left(\boldsymbol{\Sigma}_\theta + \frac{1}{n} \boldsymbol{\Sigma}_0 \right)^{-1} \boldsymbol{\mu}_\theta \right\} = \lim_{n\to\infty} \frac{1}{n} \sum_{j=1}^{n} \boldsymbol{x}_j,$$

which means that the MAP estimate is **asymptotically equivalent** to the ML estimate. Putting in simple words, the result implies that when m is large, MAP estimates and ML estimates will return the same result. If we want to be specific about this limit, we can apply Law of Large Number to show that $\lim_{n\to\infty} \frac{1}{n} \sum_{j=1}^{n} \boldsymbol{x}_j \overset{a.s.}{\to} \mathbb{E}[\boldsymbol{X}]$.

If n is small, Equation (2.48) suggests that the estimate mean is a linear combination of the ML estimate $\frac{1}{n}\sum_{j=1}^{n}\boldsymbol{x}_j$ and the prior mean $\boldsymbol{\mu}_\theta$. Therefore, MAP offers side information about the estimation using the prior. The side information is useful when n is small, but will be override by the ML estimate when n grows.

Exercise 2.4. Consider a 1D problem where the Gaussian are $x_j \,|\, \mu \sim \mathcal{N}(\mu, \sigma_0)$, and $\mu \sim \mathcal{N}(\mu_\theta, \sigma_\theta)$.

(i) Show that the MAP estimate of the mean is

$$\widehat{\mu} = \left(\frac{n\sigma_\theta^2}{n\sigma_\theta^2 + \sigma_0^2}\right)\left(\frac{1}{n}\sum_{j=1}^{n} x_j\right) + \frac{\sigma_0^2}{n\sigma_\theta^2 + \sigma_0^2}\mu_\theta. \tag{2.49}$$

(ii) Comment on the situations when $n \to \infty$, when $\sigma_0 \to 0$, or when $\sigma_\theta \to 0$. What are the physical interpretations?

2.3 Linear Regression

The Bayes decision rule we see in the previous section is a **generative method**, as we need to assume a generative model that generates the data. In this section we study another method called the **discriminative method** which does not assume any generative model. Generative methods and discriminative methods will both lead to a discriminant function $g_i(\boldsymbol{x})$. If we assume a linear discriminant function $g_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0$, then the ultimate goal of both methods is to determine the model parameters $\boldsymbol{\theta} = (\boldsymbol{w}, w_0)$. In this section we discuss a discriminative method called linear regression.

To make things simple, let us focus on a binary classification problem with a discriminant function $g_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0$. We assume that we have a set of training data $(\boldsymbol{x}_j, y_j)_{j=1}^n$, where $\boldsymbol{x}_j$ is the feature vector and $y_j \in \{-1, +1\}$ is the corresponding label. The overall training **loss** of the dataset is defined as

$$\begin{aligned} J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{j=1}^{n}(g_{\boldsymbol{\theta}}(\boldsymbol{x}_j) - y_j)^2 &= \frac{1}{n}\sum_{j=1}^{n}(\boldsymbol{w}^T\boldsymbol{x}_j + w_0 - y_j)^2 \\ &= \left\| \begin{bmatrix} \boldsymbol{x}_1^T & 1 \\ \vdots & \vdots \\ \boldsymbol{x}_n^T & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{w} \\ w_0 \end{bmatrix} - \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \right\|^2 = \|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|^2, \end{aligned} \tag{2.50}$$

where we define

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^T & 1 \\ \vdots & \vdots \\ \boldsymbol{x}_n^T & 1 \end{bmatrix}, \quad \boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{w} \\ w_0 \end{bmatrix} \quad \text{and} \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}.$$

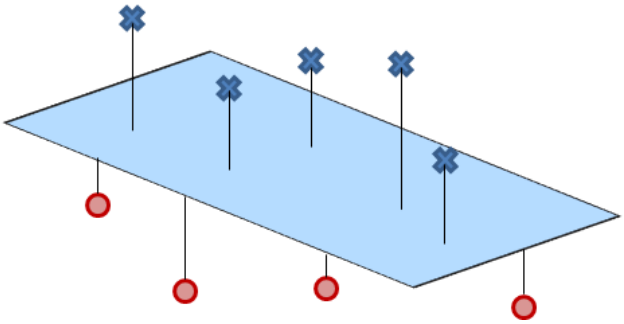


Figure 2.23: Geometric illustration of linear regression. The classes of data are marked as the crosses and the dots. The optimal separating hyperplane is the one that minimizes the residue between the data point and the plane.

The geometric illustration of linear regression is shown in Figure 2.23. In this figure, the crosses $\times$ and dots $\circ$ represent two classes of data. The value of the j-th label is y_j, and the prediction made by the discriminant function is $g_{\boldsymbol{\theta}}(\boldsymbol{x}_j)$. The optimal separating hyperplane (which is also a linear discriminant function) separates the two classes by minimizing the residue between the data point and the hyperplane. The residue of the j-th term is defined as $\varepsilon_j = (g_{\boldsymbol{\theta}}(\boldsymbol{x}_j) - y_j)^2$, and the overall residue is the sum of all the data points.

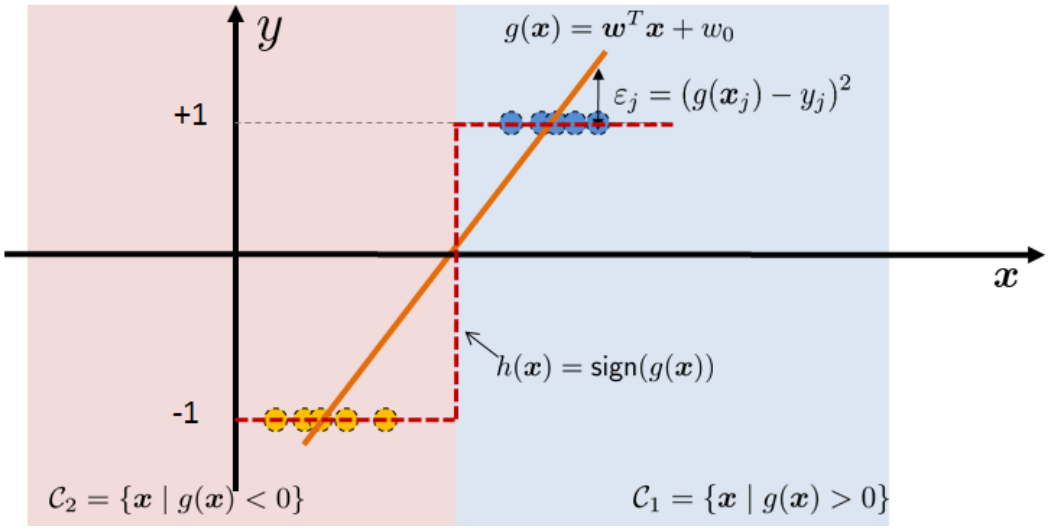


Figure 2.24: Geometry of linear regression for 1D data. While the discriminant function is linear, the hypothesis function is a unit step function with a sharp transition at $g(\boldsymbol{x}) = 0$.

Another pictorial illustration trying to link the linear discriminant function $g_{\boldsymbol{\theta}}$ with the actual hypothesis function $h_{\boldsymbol{\theta}}$ is shown in Figure 2.24. The hypothesis function $h_{\boldsymbol{\theta}}$ is defined

as

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \text{sign}(g_{\boldsymbol{\theta}}(\boldsymbol{x})) = \begin{cases} +1, & \text{if} \quad g_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0 > 0, \\ -1, & \text{if} \quad g_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0 \leq 0. \end{cases} \tag{2.51}$$

Figure 2.24 shows two classes of data with labels $y = +1$ and $y = -1$. By assuming a linear model, we seek a straight line $\boldsymbol{w}^T\boldsymbol{x} + w_0$ that best describes these data points by minimizing the sum of the individual error $\varepsilon_j = (g_{\boldsymbol{\theta}}(\boldsymbol{x}_j) - y_j)^2$. Once the model parameters $\boldsymbol{\theta}$ are determined, we form a decision rule by checking whether $g_{\boldsymbol{\theta}}(\boldsymbol{x}) > 0$ for any testing data $\boldsymbol{x}$. This gives us the two decision regions $\mathcal{C}_1$ and $\mathcal{C}_2$.

Solution of Linear Regression

To obtain the optimal parameter $\boldsymbol{\theta}$, we minimize the loss function:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\text{argmin}} \quad J(\boldsymbol{\theta}) = \|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|^2.$$

So how do we solve this optimization problem? If we take the gradient and set to zero, we will have

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|^2 \\ &= \nabla_{\boldsymbol{\theta}} \left\{ \boldsymbol{\theta}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\theta} - 2\boldsymbol{y}^T\boldsymbol{X}\boldsymbol{\theta} + \|\boldsymbol{y}\|^2 \right\} \\ &= 2\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\theta} - 2\boldsymbol{X}^T\boldsymbol{y} \\ &= \boldsymbol{0}. \end{aligned}$$

Rearranging terms we can show the so called **normal equation**: $\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\theta} = \boldsymbol{X}^T\boldsymbol{y}$, of which the solution is given by

$$\boldsymbol{\theta}^* = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}. \tag{2.52}$$

We can summarize these findings in the following Theorem.

Theorem 9 (Linear Regression Solution)**.** *The loss function of a linear regression model is given by*

$$J(\boldsymbol{\theta}) = \|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|^2, \tag{2.53}$$

of which the minimizer is

$$\boldsymbol{\theta}^* = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}. \tag{2.54}$$

For some large-scale datasets with many variables, we may want to use iterative algorithms such as gradient descent to compute the solution. In this case, the gradient descent algorithm is given by the iteration

$$\begin{aligned} \boldsymbol{\theta}^{(k+1)} &= \boldsymbol{\theta}^{(k)} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(k)}) \\ &= \boldsymbol{\theta}^{(k)} - \eta(2\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\theta}^{(k)} - 2\boldsymbol{X}^T\boldsymbol{y}). \end{aligned}$$

for some step sizes η. A pictorial illustration of the gradient descent step is shown in Figure 2.25. As we update $\boldsymbol{\theta}^{(k)}$, we construct a sequence $\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots$. Each $\boldsymbol{\theta}^{(k)}$ is the parameter of a linear discriminant function. When $\boldsymbol{\theta}^{(k)}$ is closer to the minimum value of the loss function $J(\boldsymbol{\theta})$, the corresponding discriminant function will be able to separate more data points into two classes.

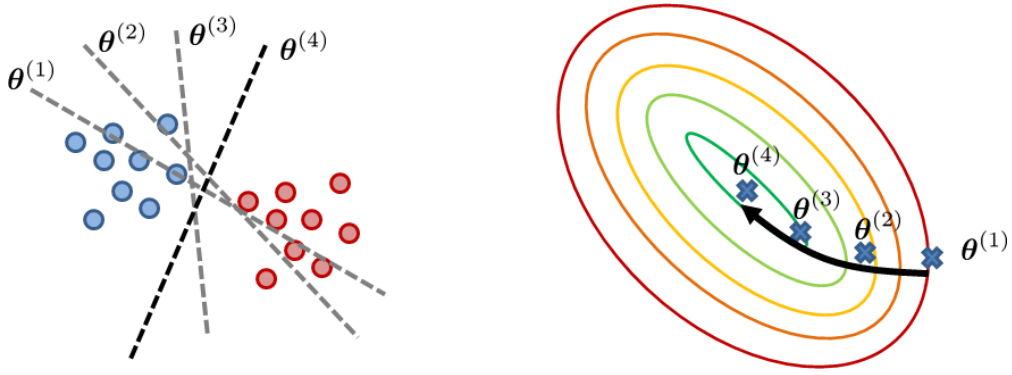


Figure 2.25: Gradient descent of linear regression. [Right] Gradient descent algorithm seeks a sequence $\boldsymbol{\theta}^{(k)}$ by searching along the steepest descent direction. The sequence $\boldsymbol{\theta}^{(k)}$ moves towards the minimum of $J(\boldsymbol{\theta})$. [Left] As $\boldsymbol{\theta}^{(k)}$ moves closer to the minimum point, the discriminant function improves its ability to separate the two classes.

Does the gradient descent algorithm converge, and if it does, is the solution unique? In the following exercise we will show that J is convex for any $\boldsymbol{X}$. Therefore, gradient descent with appropriate step size is guaranteed to find the optimal solution, and the objective value $J(\boldsymbol{\theta}^*)$ is unique. However, the solution $\boldsymbol{\theta}^*$ is unique only when J is strictly convex.

Exercise 3.1. Consider the loss function $J(\boldsymbol{\theta}) = \|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|^2$.

(i) Prove that J is always convex in $\boldsymbol{\theta}$, regardless of the choice of $\boldsymbol{X}$. Hint: We can either prove that $J(\lambda\boldsymbol{\theta}_1 + (1-\lambda)\boldsymbol{\theta}_2) \leq \lambda J(\boldsymbol{\theta}_1) + (1-\lambda)J(\boldsymbol{\theta}_2)$, or show that $\nabla^2 J(\boldsymbol{\theta})$ is positive semi-definite.

(ii) Prove that the optimal solution $\boldsymbol{\theta}^*$ is unique if and only if J is strictly convex. Hint: Show by contradiction that if $\boldsymbol{\theta}_1^*$ and $\boldsymbol{\theta}_2^*$ are both optimal with $\boldsymbol{\theta}_1^* \neq \boldsymbol{\theta}_2^*$, then any convex combination will also be optimal.

Regularization

What should we do if $\boldsymbol{X}^T\boldsymbol{X}$ is not invertible? One simple but practical approach is to consider a **regularized** linear regression:

$$J(\boldsymbol{\theta}) = \|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|^2 + \lambda\rho(\boldsymbol{\theta}), \tag{2.55}$$

where $\rho : \mathbb{R}^{d+1} \to \mathbb{R}^{d+1}$ is the **regularization function**. For example, if $\rho(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|^2$, then

$$J(\boldsymbol{\theta}) = \|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|^2 + \lambda\|\boldsymbol{\theta}\|^2. \tag{2.56}$$

In signal processing, this minimization is called the **Tikhonov regularized least squares**; In statistics, this is called the **ridge regression**. In either case, the minimizer of J satisfies

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}}\|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|^2 + \lambda\|\boldsymbol{\theta}\|^2 \\ &= 2\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}) + 2\boldsymbol{\theta} = \boldsymbol{0},\end{aligned}$$

which gives us

$$\boldsymbol{\theta}^* = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}.$$

It is not difficult to see that the new matrix $\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}$ is always invertible. If we do an eigen-decomposition of $\boldsymbol{X}^T\boldsymbol{X}$ by writing $\boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{U}|\boldsymbol{S}|^2\boldsymbol{U}^T$, then

$$\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I} = \boldsymbol{U}|\boldsymbol{S}|^2\boldsymbol{U}^T + \lambda\boldsymbol{U}\boldsymbol{U}^T = \boldsymbol{U}(|\boldsymbol{S}|^2 + \lambda\boldsymbol{I})\boldsymbol{U}^T.$$

Therefore, the j-th eigenvalue of $\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}$ is $s_j^2 + \lambda > 0$. However, in getting better matrices to invert, the price we pay is that we no longer allow large entries in $\boldsymbol{\theta}$, for large entries in $\boldsymbol{\theta}$ will be heavily penalized by the regularization $\lambda\|\boldsymbol{\theta}\|^2$. In the extreme if $\lambda \to \infty$, then the optimal solution will become $\boldsymbol{\theta}^* = \boldsymbol{0}$.

Exercise 3.2. Consider the loss function

$$J(\boldsymbol{\theta}) = \|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|^2 + \lambda\rho(\boldsymbol{\theta}).$$

(i) Prove that if ρ is convex, then J is convex.

(ii) Let $\rho(\boldsymbol{\theta}) = \|\boldsymbol{B}\boldsymbol{\theta}\|^2$ for some matrix $\boldsymbol{B} \in \mathbb{R}^{(d+1)\times(d+1)}$. Find the optimal $\boldsymbol{\theta}^*$.

In modern statistical learning, powerful regularization functions ρ are typically non-differentiable, e.g., $\rho(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1$. The corresponding optimization

$$J(\boldsymbol{\theta}) = \|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|^2 + \lambda\|\boldsymbol{\theta}\|_1 \tag{2.57}$$

is called the **LASSO** (stands for **L**east **A**bsolute **S**hrinkage and **S**election **O**perator). LASSO has many nice properties such as: (i) It promotes sparsity of $\boldsymbol{\theta}$, i.e., forcing many entries of $\boldsymbol{\theta}$ to zero so that we can compress the set of active features; (ii) It is the tightest convex relaxation of the NP-hard $\|\boldsymbol{\theta}\|_0$; (iii) Although $\|\boldsymbol{\theta}\|_1$ is non-differentiable (at $\boldsymbol{\theta} = \boldsymbol{0}$), there exits polynomial time convex solvers to solve the problem, e.g., interior point methods, or alternating direction method of multipliers.

Linear Regression beyond Classification

Linear regression can be used for applications beyond classification. As a demonstration of linear regression and illustrating how ℓ_1 regularization is used, we study a data fitting problem. Given a dataset consisting of multiple attributes, we can construct a data matrix

$$\boldsymbol{X} = \begin{bmatrix} | & | & & | \\ \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_d \\ | & | & & | \end{bmatrix}, \tag{2.58}$$

where each column $\boldsymbol{x}_j \in \mathbb{R}^n$ is a feature vector. These feature vectors can be considered as the contributing factors of observing the vector $\boldsymbol{y} \in \mathbb{R}^n$ via a set of regression coefficients $\boldsymbol{\theta} \in \mathbb{R}^d$. This means

$$\boldsymbol{y} \approx \sum_{j=1}^{d} \theta_j \boldsymbol{x}_j = \boldsymbol{X}\boldsymbol{\theta}. \tag{2.59}$$

The optimal regression coefficient can then be found as

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\text{argmin}} \quad \|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|^2.$$

To see a concrete example, we use the crime rate data obtained from `https://web.stanford.edu/~hastie/StatLearnSparsity/data.html`. A snapshot of the data is shown in Figure 2.26. In this dataset, the vector $\boldsymbol{y}$ is the crime rate, which is the last column of Figure 2.26. The feature vectors are `funding`, `hs`, `not-hs`, `college`, `college4`.

city	funding	hs	not-hs	college	college4	crime rate
1	40	74	11	31	20	478
2	32	72	11	43	18	494
3	57	70	18	16	16	643
4	31	71	11	25	19	341
5	67	72	9	29	24	773
⋮	⋮	⋮	⋮	⋮		
50	66	67	26	18	16	940

Figure 2.26: Crime rate data. Extracted from Hastie and Tibshirani's Statistical Learning with Sparsity.

The following MATLAB / Python code is used to extract the data.

```
data = load('data_crime.txt');
y       = data(:,1);        % The observed crime rate
A       = data(:,3:end);    % Feature vectors
[m,n]   = size(A);
```


Once we have extracted the data from the dataset, we consider two optimizations

$$\widehat{\boldsymbol{\theta}}_1(\lambda) = \underset{\boldsymbol{\theta}}{\text{argmin}} \quad J_1(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{\theta}\|_1,$$
$$\widehat{\boldsymbol{\theta}}_2(\lambda) = \underset{\boldsymbol{\theta}}{\text{argmin}} \quad J_2(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{\theta}\|_2^2.$$

As we have discussed, the first optimization is the ℓ_1 regularized least squares which is also called the LASSO problem. The second optimization is the standard ℓ_2 regularized least squares. We use regularization instead of solving the bare linear regression because the raw data does not guarantee an invertible matrix $\boldsymbol{X}^T\boldsymbol{X}$. Now, in this experiment, we would like to visualize the linear regression coefficients $\widehat{\boldsymbol{\theta}}_1(\lambda)$ and $\widehat{\boldsymbol{\theta}}_2(\lambda)$ as the regularization parameter λ changes. To solve the optimization, we use `CVX` with the MATLAB / Python implementation shown below.

```
% MATLAB code to solve L1 and L2 minimization, and visualize the result
lambdaset = logspace(-1,8,50);
x_store   = zeros(5,length(lambdaset));
for i=1:length(lambdaset)
    lambda = lambdaset(i);
    cvx_begin
        variable x(n)
        minimize( sum_square( A*x - y ) + lambda * norm(x , 1) )
%         minimize( sum_square( A*x - y ) + lambda * sum_square(x) )
    cvx_end
    x_store(:,i) = x(:);
end
semilogx(lambdaset, x_store, 'LineWidth', 2);
legend('funding','high', 'no high', 'college', 'graduate', 'Location','NW');
xlabel('lambda');
ylabel('feature attribute');
```

The result in Figure 2.27 shows some interesting differences between the two linear regression models. For the ℓ_2 estimate $\boldsymbol{\theta}_2(\lambda)$, the trajectory of the regression coefficients is smooth as λ changes. This is attributed to the fact that the ℓ_2 optimization cost $J_2(\boldsymbol{\theta})$ is continuously differentiable in $\boldsymbol{\theta}$, and so the solution trajectory is smooth. In contrast, the ℓ_1 estimate $\boldsymbol{\theta}_1(\lambda)$ has a more disruptive trajectory. As λ reduces, the feature `percentage of high-school` is first activated. A possible implication is that if we are to limit ourselves to *one* most salient feature, then the percentage of high-school is the feature we should select. As such, the trajectory of the regression coefficient shows an ordered list of features that are contributing to the observation. As $\lambda \to 0$, both $\boldsymbol{\theta}_1(\lambda)$ and $\boldsymbol{\theta}_2(\lambda)$ reach the same solution, as the cost functions become identical.

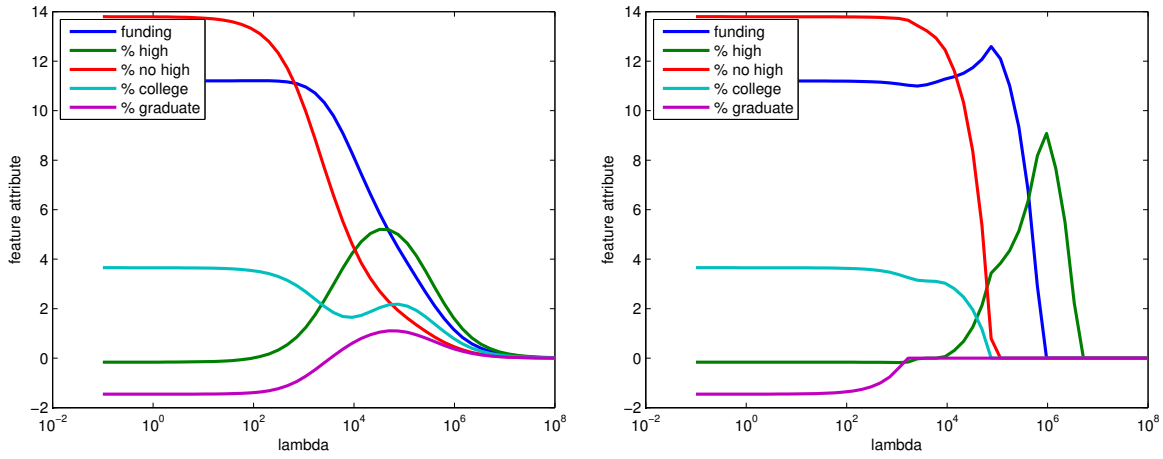


Figure 2.27: Crime rate data. Extracted from Hastie and Tibshirani's Statistical Learning with Sparsity.

Connection with Bayesian Decision Rule

Unlike the Bayes decision rule, linear regression does not assume any probabilistic modeling of the data. However, if we insist of putting a probabilistic model, then we can argue that the loss function $J(\boldsymbol{\theta})$ converges almost surely to its expectation (via Strong / Weak Law of Large Number):

$$\frac{1}{n}\sum_{j=1}^{n}(g(\boldsymbol{x}_j)-y_j)^2 \xrightarrow{p} \mathbb{E}_{\boldsymbol{X},Y}[g(\boldsymbol{X})-Y)^2]. \tag{2.60}$$

Here, we have a small abuse of notation to denote $\boldsymbol{X}$ as a random variable in $\mathbb{R}^d$ that generates the samples $\boldsymbol{x}_j$. By minimizing $J(\boldsymbol{\theta})$, we are essentially minimizing the expectation:

$$\begin{aligned}\boldsymbol{\theta}^* &= \underset{\boldsymbol{w},w_0}{\text{argmin}} \quad \frac{1}{n}\sum_{j=1}^{n}(g(\boldsymbol{x}_j)-y_j)^2 \\ &= \underset{\boldsymbol{w},w_0}{\text{argmin}} \quad \mathbb{E}_{\boldsymbol{X},Y}\left[(\boldsymbol{w}^T\boldsymbol{X}+w_0-Y)^2\right].\end{aligned} \tag{2.61}$$

Taking the derivatives with respect to $(\boldsymbol{w}, w_0)$ yields

$$\frac{d}{d\boldsymbol{w}}\mathbb{E}_{\boldsymbol{X},Y}\left[(\boldsymbol{w}^T\boldsymbol{X}+w_0-Y)^2\right] = 2\left(\mathbb{E}[\boldsymbol{X}\boldsymbol{X}^T]\boldsymbol{w}+\mathbb{E}[\boldsymbol{X}]w_0-\mathbb{E}[\boldsymbol{X}Y]\right), \tag{2.62}$$

$$\frac{d}{dw_0}\mathbb{E}_{\boldsymbol{X},Y}\left[(\boldsymbol{w}^T\boldsymbol{X}+w_0-Y)^2\right] = 2\left(\mathbb{E}[\boldsymbol{X}]^T\boldsymbol{w}+w_0-\mathbb{E}[Y]\right), \tag{2.63}$$

where the expectations are taken over the joint distribution $p_{\boldsymbol{X},Y}(\boldsymbol{x},y)$. Bayes Theorem allows us to decompose $p_{\boldsymbol{X},Y}(\boldsymbol{x},y) = p_{\boldsymbol{X}|Y}(\boldsymbol{x}|i)p_Y(i)$. If we assume that the conditional distribution $p_{\boldsymbol{X}|Y}(\boldsymbol{x}|i)$ is Gaussian with identical covariance:

$$p_{\boldsymbol{X}|Y}(\boldsymbol{x}|i) = \frac{1}{\sqrt{2\pi|\boldsymbol{\Sigma}|}}\exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right\}, \qquad i\in\{-1,+1\} \tag{2.64}$$

and the prior is equal $p_Y(i) = \frac{1}{2}$, then we have $\mathbb{E}[Y] = 0$. Moreover, we can show that the expectation of $\boldsymbol{X}$ and $\mathbb{E}[\boldsymbol{X}Y]$ are

$$\mathbb{E}[\boldsymbol{X}] = \sum_{i\in\{-1,+1\}} \mathbb{E}[\boldsymbol{X}|Y=i]p_Y(i) = \frac{1}{2}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_{-1})$$
$$\mathbb{E}[\boldsymbol{X}Y] = \sum_{i\in\{-1,+1\}} \mathbb{E}[\boldsymbol{X}Y|Y=i]p_Y(i) = \frac{1}{2}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}).$$

The covariance can be shown as

$$\begin{aligned}\mathbb{E}[\boldsymbol{X}\boldsymbol{X}^T] - \mathbb{E}[\boldsymbol{X}]\mathbb{E}[\boldsymbol{X}]^T &= \mathbb{E}[(\boldsymbol{X} - \mathbb{E}[\boldsymbol{X}])(\boldsymbol{X} - \mathbb{E}[\boldsymbol{X}])^T] \\ &= \sum_{i\in\{-1,+1\}} \mathbb{E}\left[(\boldsymbol{X} - \mathbb{E}[\boldsymbol{X}])(\boldsymbol{X} - \mathbb{E}[\boldsymbol{X}])^T|Y=i\right] p_Y(i) \\ &= \frac{1}{2}\boldsymbol{\Sigma} + \frac{1}{2}\boldsymbol{\Sigma} = \boldsymbol{\Sigma}.\end{aligned}$$

Consequently, if we multiply $\mathbb{E}[\boldsymbol{X}]$ to Equation (2.63) and subtract from Equation (2.62), we will obtain

$$(\mathbb{E}[\boldsymbol{X}\boldsymbol{X}^T] - \mathbb{E}[\boldsymbol{X}]\mathbb{E}[\boldsymbol{X}]^T)\boldsymbol{w} = \mathbb{E}[\boldsymbol{X}Y] - \mathbb{E}[\boldsymbol{X}]\mathbb{E}[Y], \tag{2.65}$$

which gives

$$\boldsymbol{\Sigma}\boldsymbol{w} = \frac{1}{2}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}) \quad \Longrightarrow \quad \boldsymbol{w} = \frac{1}{2}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}). \tag{2.66}$$

Substituting into Equation (2.63), we have

$$\begin{aligned}w_0 = -\mathbb{E}[\boldsymbol{X}]^T\boldsymbol{w} &= -\frac{1}{2}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_{-1}) \cdot \frac{1}{2}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}) \\ &= -\frac{1}{4}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_{-1})\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}).\end{aligned} \tag{2.67}$$

Hence, the optimal $(\boldsymbol{w}^*, w_0^*)$ satisfies

$$\begin{aligned}\boldsymbol{w}^* &= \frac{1}{2}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}) = \widetilde{\boldsymbol{\Sigma}}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}) \\ w_0^* &= -\frac{1}{4}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_{-1})\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}) = -\frac{1}{2}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_{-1})\widetilde{\boldsymbol{\Sigma}}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}),\end{aligned} \tag{2.68}$$

if we define $\widetilde{\boldsymbol{\Sigma}} \stackrel{\text{def}}{=} \boldsymbol{\Sigma}/2$.

In summary, the separating hyperplane returned by linear regression coincides with the separating hyperplane returned by Bayes (Case 1 and Case 2) if $\pi_i = \pi_j$, i.e., the priors are identical. (Case 3 does not work here because Case 3 is intrinsically quadratic, which cannot be Bayesped to a linear classifier.) The result is summarized as below.

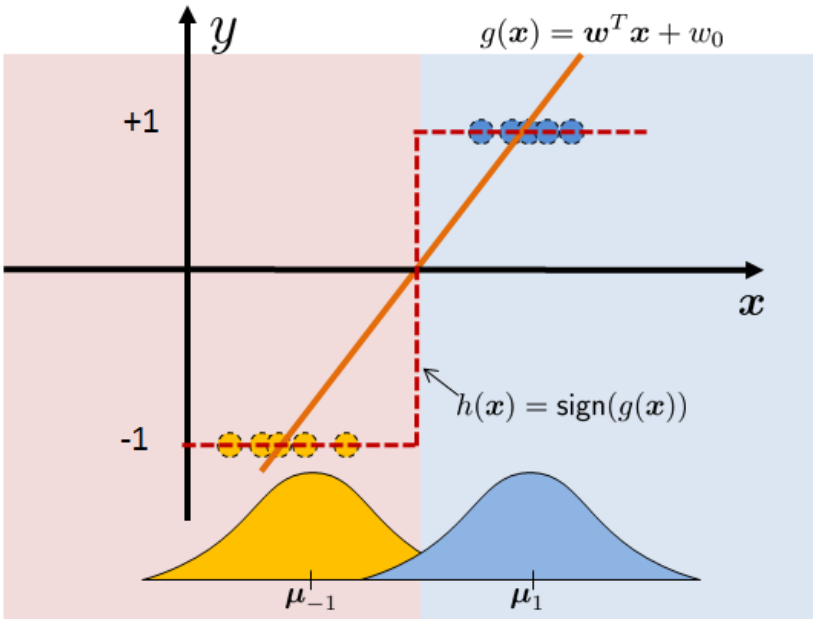


Figure 2.28: Linear regression coincides with the Gaussian Bayes when the covariances are identical, and the priors are the same.

Theorem 10 (Conditions for Linear Regression = Bayes)**.** *Suppose that all the following two conditions are satisfied:*
(i) the likelihood $p_{\boldsymbol{X}|Y}(\boldsymbol{x}|i)$ is Gaussian satisfying Equation (2.64),
(ii) the prior is uniform: $p_Y(i) = \frac{1}{2}$ for $i \in \{+1, -1\}$,
(iii) the number of training samples goes to infinity, i.e., the empirical average in in Equation (2.60) converges to the expectation.
Then, the linear regression model parameter $(\boldsymbol{w}, w_0)$ is given by

$$\boldsymbol{w} = \widetilde{\boldsymbol{\Sigma}}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}), \qquad w_0 = -\frac{1}{2}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_{-1})\widetilde{\boldsymbol{\Sigma}}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}), \tag{2.69}$$

where $\widetilde{\boldsymbol{\Sigma}} \overset{def}{=} \boldsymbol{\Sigma}/2$, and $\boldsymbol{\Sigma}$ is the covariance of the Gaussian.

As a special case, if $\widetilde{\boldsymbol{\Sigma}} = \frac{\sigma^2}{2}\boldsymbol{I}$, then $(\boldsymbol{w}, w_0)$ is given by

$$\boldsymbol{w} = \frac{\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}}{2\sigma^2}, \qquad w_0 = -\frac{\|\boldsymbol{\mu}_1\|^2 - \|\boldsymbol{\mu}_{-1}\|^2}{2\sigma^2}, \tag{2.70}$$

which is exactly the same as Case 1 with $\pi_i = \pi_j$.

Figure 2.28 illustrates the situation in 1D, where we pose a Gaussian distribution to each class. By assuming that the covariance matrices of the two Gaussians are equal, and by assuming that the priors are the same, we force the decision boundary $\boldsymbol{w}^T\boldsymbol{x} + w_0$ to be right

at the middle of the two classes. This could be sub-optimal, because by assuming $\pi_{+1} = \pi_{-1}$ we implicitly assume that the two classes have equal influence. We are also losing control over cases where $\boldsymbol{\Sigma}_{+1} \neq \boldsymbol{\Sigma}_{-1}$, which could again be sub-optimal.

2.4 Logistic Regression

Putting aside the equal covariance assumption, there are two additional drawbacks of linear regression. First, the discriminant function $g(\boldsymbol{x})$ is a straight line through the two clusters. Since the line passes through the center of each cluster, data points sitting farther from the center will have a large fitting error. Thus, it would be better to replace the linear model with a function that is closer to sign$(\cdot)$. The second drawback is the smooth transition at the decision boundary. Again, if we can use a function that is closer to sign$(\cdot)$, we can make the transition sharp.

Logistic regression is an alternative to the linear regression. Instead of constructing a classifier through a two-step approach, i.e., first define a discriminant function $g(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0$ and then define a hypothesis function $h(\boldsymbol{x}) = \text{sign}(g(\boldsymbol{x}))$, logistic regression directly approximates the hypothesis function $h(\boldsymbol{x})$ using a logistic function:

$$h(\boldsymbol{x}) = \frac{1}{1+e^{-g(\boldsymbol{x})}} = \frac{1}{1+e^{-(\boldsymbol{w}^T\boldsymbol{x}+w_0)}} = \frac{e^{(\boldsymbol{w}^T\boldsymbol{x}+w_0)}}{1+e^{(\boldsymbol{w}^T\boldsymbol{x}+w_0)}}. \tag{2.71}$$

How does a logistic function look like? If we ignore the linear component $\boldsymbol{w}^T\boldsymbol{x} + w_0$ for a while by just looking at a 1D function

$$h(x) = \frac{1}{1+e^{-a(x-x_0)}}, \qquad \text{for some } a \text{ and } x_0, \tag{2.72}$$

we note that

$$\begin{aligned} h(x) &\to 1, \quad \text{as} \quad x \to \infty, \\ h(x) &\to 0, \quad \text{as} \quad x \to -\infty, \end{aligned} \tag{2.73}$$

and there is a cutoff at $x = x_0$, which gives $h(x_0) = 1/2$. The sharpness of the transient is determined by the magnitude of the constant a: Larger a gives a sharper transient, and smaller a gives s slower transient. The location of the cutoff is determined by the offset x_0. If we make x_0 larger, then the cut is shifted towards the right; otherwise it is shifted towards the left. A pictorial illustration is shown in Figure 2.29.

For high-dimensional vectors, the geometry of the hypothesis function is determined by the linear component $\boldsymbol{w}^T\boldsymbol{x} + w_0$. The cutoff happens at a point $\boldsymbol{x}_0 \in \mathbb{R}^d$ such that $\boldsymbol{w}^T\boldsymbol{x}_0 = w_0$. The orientation and sharpness of the cutoff transient is determined by the plane $\boldsymbol{w}^T\boldsymbol{x}$. Since $\boldsymbol{w}$ is often bounded, the hypothesis function $h(\boldsymbol{x}) \to 1$ when $\|\boldsymbol{x}\| \to \infty$, and $h(\boldsymbol{x}) \to 0$ when $\|\boldsymbol{x}\| \to -\infty$. Essentially, this separates the space into two half-spaces, one taking the value 1 and the other taking the value 0.

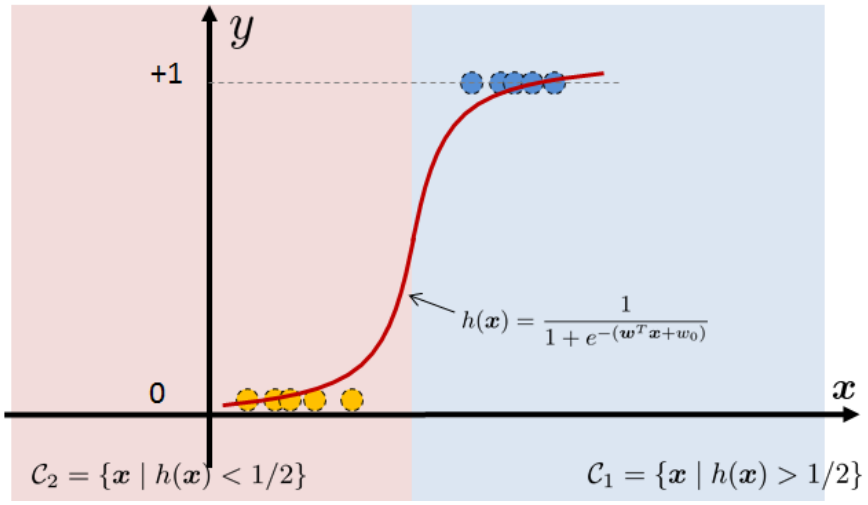


Figure 2.29: Logistic regression: Given the two classes of data points, the model tries to directly determine a hypothesis function h.

In order to train a logistic model, we need to define an cost function or the training loss function. To simplify the notation we denote $\boldsymbol{\theta} \stackrel{\text{def}}{=} (\boldsymbol{w}, w_0)$, and write $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ instead of $h(\boldsymbol{x})$ to emphasize the dependency of $\boldsymbol{\theta}$. Then, we define the cost function as

$$J(\boldsymbol{\theta}) = \sum_{j=1}^{n} -\Big\{ y_j \log h_{\boldsymbol{\theta}}(\boldsymbol{x}_j) + (1-y_j)\log(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}_j)) \Big\}, \tag{2.74}$$

where the summation is taken over the entire training set $\{(\boldsymbol{x}_j, y_j)\}_{j=1}^n$, and we assume that $y_j \in \{0,1\}$. The optimal parameter $\boldsymbol{\theta}^*$ is then given by

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\text{argmin}} \quad J(\boldsymbol{\theta}). \tag{2.75}$$

Why do we want to choose this cost function? First of all, since y_j takes a binary value, and $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ also takes a (approximately) binary value, we have

$$y_j \log h_{\boldsymbol{\theta}}(\boldsymbol{x}_j) = \begin{cases} 0, & \text{if} \quad y_j = 1, \quad \text{and} \quad h_{\boldsymbol{\theta}}(\boldsymbol{x}_j) = 1, \\ -\infty, & \text{if} \quad y_j = 1, \quad \text{and} \quad h_{\boldsymbol{\theta}}(\boldsymbol{x}_j) = 0, \end{cases}$$

$$(1-y_j)(1-\log h_{\boldsymbol{\theta}}(\boldsymbol{x}_j)) = \begin{cases} 0, & \text{if} \quad y_j = 0, \quad \text{and} \quad h_{\boldsymbol{\theta}}(\boldsymbol{x}_j) = 0, \\ -\infty, & \text{if} \quad y_j = 0, \quad \text{and} \quad h_{\boldsymbol{\theta}}(\boldsymbol{x}_j) = 1. \end{cases}$$

Therefore, unless y_j matches with $h_{\boldsymbol{\theta}}(\boldsymbol{x}_j)$, the cost function $J(\boldsymbol{\theta})$ will be $+\infty$ (because there is a negative sign). In other words, by minimizing $J(\boldsymbol{\theta})$ we are forced to find a $h_{\boldsymbol{\theta}}$ such that all the $\boldsymbol{x}_j$'s are matched with the labels y_j as much as they can.

Exercise 4.1. The cost function $J(\boldsymbol{\theta})$ defined above is for $y_j \in \{0,1\}$. How would you modify the cost fucntion for $y_j \in \{+1,-1\}$?

Remark. It would be tempted to consider the following cost function

$$J(\boldsymbol{\theta}) = \sum_{j=1}^{n} (h_{\boldsymbol{\theta}}(\boldsymbol{x}_j) - y_j)^2, \tag{2.76}$$

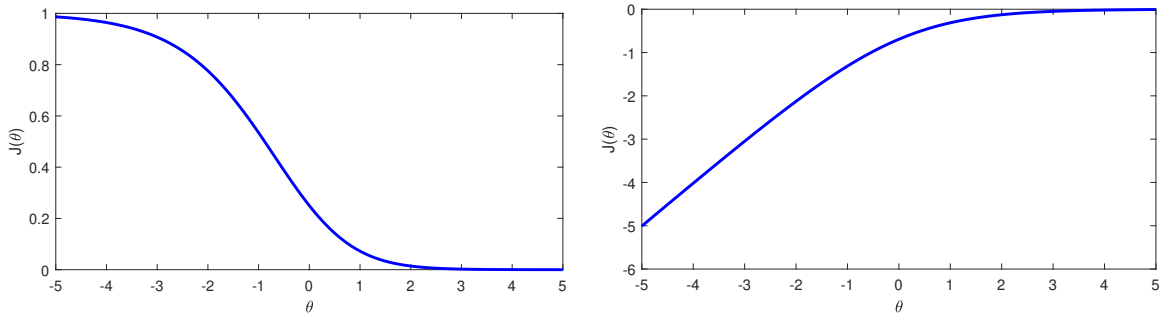
which resembles everything from the linear regression case. However, such choice is problematic because J is non-convex. To see this, we can consider a scalar case with only one training sample. This gives us the cost function

$$J(\theta) = \left(\frac{1}{1 + e^{-\theta x}} - y \right)^2. \tag{2.77}$$

By fixing a pair of (x, y), we can plot the function $J(\theta)$. As shown in Figure 2.30, such J is not convex. On the other hand, the proposed logistic cost function

$$J(\theta) = y \log \left(\frac{1}{1 + e^{-\theta x}} \right) \tag{2.78}$$

is convex, see Figure 2.30. Convexity plays an important role here because if the cost function is convex, then any minimizer of the cost function is the global minimizer.



$J(\theta) = \left(\frac{1}{1+e^{-\theta x}} - y \right)^2$ $\qquad$ $J(\theta) = y \log \left(\frac{1}{1+e^{-\theta x}} \right)$

Figure 2.30: Convexity of different choices of J. In both experiments, we set $x = 1$ and $y = 1$. The result indicates that a squared error cost does not preserve convexity, whereas the logistic cost does.

Probabilistic Interpretation

The cost function defined in Equation (2.74) has a probabilistic interpretation. If we take the exponential of the cost (which is allowed because an exponential function does not alters the minimizer as it is an increasing function), then we can obtain a **maximum likelihood**

estimation of a **Bernoulli** distribution:

$$\begin{aligned}\underset{\boldsymbol{\theta}}{\text{argmin}} \quad J(\boldsymbol{\theta}) &= \underset{\boldsymbol{\theta}}{\text{argmin}} \quad \sum_{j=1}^{n} -\Big\{ y_j \log h_{\boldsymbol{\theta}}(\boldsymbol{x}_j) + (1-y_j)\log(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}_j))\Big\} \\ &= \underset{\boldsymbol{\theta}}{\text{argmin}} \quad -\log\left(\prod_{j=1}^{n} h_{\boldsymbol{\theta}}(\boldsymbol{x}_j)^{y_j}(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}_j))^{1-y_j}\right) \\ &= \underset{\boldsymbol{\theta}}{\text{argmax}} \quad \prod_{j=1}^{n}\Big\{ h_{\boldsymbol{\theta}}(\boldsymbol{x}_j)^{y_j}(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}_j))^{1-y_j}\Big\}. \end{aligned} \tag{2.79}$$

Putting in an other way, we can interpret $h_{\boldsymbol{\theta}}(\boldsymbol{x}_j)$ as the posterior probability of a Bernoulli random variable, i.e.,

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}_j) = p_{Y|\boldsymbol{X}}(1 \mid \boldsymbol{x}_j), \quad \text{and} \quad 1-h_{\boldsymbol{\theta}}(\boldsymbol{x}_j) = p_{Y|\boldsymbol{X}}(0 \mid \boldsymbol{x}_j). \tag{2.80}$$

The label y_j is the random realization of this Bernoulli. Since we interpret $h_{\boldsymbol{\theta}}$ as the probability, $h_{\boldsymbol{\theta}}(\boldsymbol{x}_j)$ must be a value between 0 and 1. This is enabled by the definition of the logistic function. (If $h_{\boldsymbol{\theta}}$ is linear, i.e., $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta}^T\boldsymbol{x}$, then it cannot be interpreted as a probability as it goes beyond $[0,1]$.)

In statistics, the term

$$\log\left(\frac{h_{\boldsymbol{\theta}}(\boldsymbol{x})}{1-h_{\boldsymbol{\theta}}(\boldsymbol{x})}\right) \tag{2.81}$$

is called the **log-odds**. It turns out that for logistic regression, the log-odds is in fact linear.

Lemma 1. *Suppose* $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{\theta}^T\boldsymbol{x}}}$, *then*

$$\log\left(\frac{h_{\boldsymbol{\theta}}(\boldsymbol{x})}{1-h_{\boldsymbol{\theta}}(\boldsymbol{x})}\right) = \boldsymbol{\theta}^T\boldsymbol{x}. \tag{2.82}$$

Proof. To prove the result, we just need to show

$$\frac{h_{\boldsymbol{\theta}}(\boldsymbol{x})}{1-h_{\boldsymbol{\theta}}(\boldsymbol{x})} = \frac{\frac{1}{1+e^{-\boldsymbol{\theta}^T\boldsymbol{x}}}}{\frac{e^{-\boldsymbol{\theta}^T\boldsymbol{x}}}{1+e^{-\boldsymbol{\theta}^T\boldsymbol{x}}}} = e^{\boldsymbol{\theta}^T\boldsymbol{x}}.$$

Hence, taking the log on both sides yields the result. □

The result of this lemma suggests that in logistic regression, the linearity is now shifted from the input $\boldsymbol{x}$ to the log-odds. But more importantly, the lemma allows us to simplify the cost function, and eventually allows us to claim convexity.

Convexity and Gradient Descent

Theorem 11 (Convexity of $J(\theta)$)**.** *The logistic cost function*

$$J(\boldsymbol{\theta}) \stackrel{def}{=} \sum_{j=1}^{n} -\Big\{ y_j \log h_{\boldsymbol{\theta}}(\boldsymbol{x}_j) + (1-y_j)\log(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}_j)) \Big\} \tag{2.83}$$

is convex in $\boldsymbol{\theta}$.

Proof. By the previous lemma, we can first rewrite the cost function in terms of the log-odds.

$$\begin{aligned} J(\boldsymbol{\theta}) &= \sum_{j=1}^{n} -\Big\{ y_j \log\left(\frac{h_{\boldsymbol{\theta}}(\boldsymbol{x}_j)}{1-h_{\boldsymbol{\theta}}(\boldsymbol{x}_j)}\right) + \log(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}_j)) \Big\} \\ &= \sum_{j=1}^{n} -\Big\{ y_j \boldsymbol{\theta}^T \boldsymbol{x}_j + \log(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}_j)) \Big\}. \end{aligned} \tag{2.84}$$

Since the first term is linear, it remains to prove the second term is convex. To this end, we first show that

$$\begin{aligned} \nabla_{\boldsymbol{\theta}}[-\log(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}))] &= -\nabla_{\boldsymbol{\theta}}\left[\log\left(1-\frac{1}{1+e^{-\boldsymbol{\theta}^T\boldsymbol{x}}}\right)\right] \\ &= -\nabla_{\boldsymbol{\theta}}\left[\log\frac{e^{-\boldsymbol{\theta}^T\boldsymbol{x}}}{1+e^{-\boldsymbol{\theta}^T\boldsymbol{x}}}\right] \\ &= -\nabla_{\boldsymbol{\theta}}\left[-\boldsymbol{\theta}^T\boldsymbol{x} - \log(1+e^{-\boldsymbol{\theta}^T\boldsymbol{x}})\right] \\ &= \boldsymbol{x} + \nabla_{\boldsymbol{\theta}}\left[\log\left(1+e^{-\boldsymbol{\theta}^T\boldsymbol{x}}\right)\right] \\ &= \boldsymbol{x} + \left(\frac{-e^{-\boldsymbol{\theta}^T\boldsymbol{x}}}{1+e^{-\boldsymbol{\theta}^T\boldsymbol{x}}}\right)\boldsymbol{x} = h_{\boldsymbol{\theta}}(\boldsymbol{x})\boldsymbol{x}. \end{aligned}$$

The Hessian is therefore:

$$\begin{aligned} \nabla^2_{\boldsymbol{\theta}}[-\log(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}))] &= \nabla_{\boldsymbol{\theta}}\left[h_{\boldsymbol{\theta}}(\boldsymbol{x})\boldsymbol{x}\right] \\ &= \nabla_{\boldsymbol{\theta}}\left[\left(\frac{1}{1+e^{-\boldsymbol{\theta}^T\boldsymbol{x}}}\right)\boldsymbol{x}\right] \\ &= \left(\frac{1}{(1+e^{-\boldsymbol{\theta}^T\boldsymbol{x}})^2}\right)\left(-e^{-\boldsymbol{\theta}^T\boldsymbol{x}}\right)\boldsymbol{x}\boldsymbol{x}^T \\ &= \left(\frac{1}{1+e^{-\boldsymbol{\theta}^T\boldsymbol{x}}}\right)\left(1-\frac{1}{1+e^{-\boldsymbol{\theta}^T\boldsymbol{x}}}\right)\boldsymbol{x}\boldsymbol{x}^T \\ &= h_{\boldsymbol{\theta}}(\boldsymbol{x})[1-h_{\boldsymbol{\theta}}(\boldsymbol{x})]\boldsymbol{x}\boldsymbol{x}^T. \end{aligned}$$

Therefore, $\nabla^2_{\boldsymbol{\theta}}[-\log(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}))]$ is positive semi-definite because for any $\boldsymbol{v}\in\mathbb{R}^d$

$$\begin{aligned}\boldsymbol{v}^T\nabla^2_{\boldsymbol{\theta}}[-\log(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}))]\boldsymbol{v} &= \boldsymbol{v}^T\left[h_{\boldsymbol{\theta}}(\boldsymbol{x})[1-h_{\boldsymbol{\theta}}(\boldsymbol{x})]\boldsymbol{x}\boldsymbol{x}^T\right]\boldsymbol{v}\\ &= \left(h_{\boldsymbol{\theta}}(\boldsymbol{x})[1-h_{\boldsymbol{\theta}}(\boldsymbol{x})]\right)\|\boldsymbol{v}^T\boldsymbol{x}\|^2\geq 0.\end{aligned}$$

The last inequality holds because $0\leq h_{\boldsymbol{\theta}}(\boldsymbol{x})\leq 1$ and hence $h_{\boldsymbol{\theta}}(\boldsymbol{x})[1-h_{\boldsymbol{\theta}}(\boldsymbol{x})]\geq 0$. □

The implication of the Theorem is that since $J(\boldsymbol{\theta})$ is convex in $\boldsymbol{\theta}$, the optimal solution can be determined by most of the numerical solvers. In particular, if one choose gradient descent, then the algorithm should land on the global minimizer, provided appropriate step sizes. The gradient descent step is given by

$$\begin{aligned}\boldsymbol{\theta}^{(k+1)} &= \boldsymbol{\theta}^{(k)}-\alpha_k\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta}^{(k)})\\ &= \boldsymbol{\theta}^{(k)}-\alpha_k\left(\sum_{j=1}^n(h_{\boldsymbol{\theta}}(\boldsymbol{x}_j)-y_j)\boldsymbol{x}_j\right).\end{aligned}\tag{2.85}$$

For readers prefer an off-the-shelf optimization solver, we can use the following MATLAB / Python program to construct a logistic regression model. The optimization solver we use is `CVX`. In order to write the optimization into a standard `CVX` format, we write the logistic regression cost function as

$$\begin{aligned}J(\boldsymbol{\theta}) &= \sum_{j=1}^n-\Big\{y_j\boldsymbol{\theta}^T\boldsymbol{x}_j+\log(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}_j))\Big\} = \sum_{j=1}^n-\left\{y_j\boldsymbol{\theta}^T\boldsymbol{x}_j+\log\left(1-\frac{e^{\boldsymbol{\theta}^T\boldsymbol{x}}}{1+e^{\boldsymbol{\theta}^T\boldsymbol{x}}}\right)\right\}\\ &= \sum_{j=1}^n-\left\{y_j\boldsymbol{\theta}^T\boldsymbol{x}_j-\log\left(1+e^{\boldsymbol{\theta}^T\boldsymbol{x}}\right)\right\} = -\left\{\left(\sum_{j=1}^n y_j\boldsymbol{x}_j\right)^T\boldsymbol{\theta}-\sum_{j=1}^n\log\left(1+e^{\boldsymbol{\theta}^T\boldsymbol{x}}\right)\right\}.\end{aligned}$$

The first term in this expression is linear, whereas the second term is the sum of a log-sum-exp function, i.e., $\log(e^{x_1}+e^{x_2}+\ldots+e^{x_n})$. In `CVX`, the function we can use is `log_sum_exp`. The following program is a 1D example implemented in MATLAB provided by Boyd and Lievenberg's *Convex Optimization*. If $\boldsymbol{x}_j$ is a scalar, then the first term $\sum_{j=1}^n y_jx_j$ is simplified to the inner product $\boldsymbol{y}^T\boldsymbol{x}$ where $\boldsymbol{y}=[y_1,\ldots,y_n]^T$ and $\boldsymbol{x}=[x_1,\ldots,x_n]^T$.

```
% MATLAB code to perform a logistic regression
% Credit to Convex Optimization Chapter 7.1, and the example in CVX.
% Generate data
w_true  =  1;
w0_true = -5 ;
n       = 200;
x = 10*rand(n,1);
y = (rand(n,1) < exp(w_true*x+w0_true)./(1+exp(w_true*x+w0_true)));
```



```
% Solve CVX
xtilde = [x ones(n,1)];
cvx_expert true
cvx_begin
    variables theta(2)
    maximize(y'*xtilde*theta-sum(log_sum_exp([zeros(1,n); theta'*xtilde'])))
cvx_end
```

The result of this experiment is demonstrated in Figure 2.31, with the corresponding plotting commands shown below.

```
% To plot the result.
ind1 = find(y==1);
ind2 = find(y==0);

w    = theta(1);
w0   = theta(2);
xs   = linspace(-1,11,1000)';
ys   = exp(w*xs + w0)./(1+exp(w*xs + w0));
ytrue= exp(w_true*xs + w0_true)./(1+exp(w_true*xs + w0_true));

plot(x,y,'o','LineWidth',2,'MarkerSize',8); hold on;
plot(xs, ys, 'k', 'LineWidth', 2);
plot(xs, ytrue, 'r', 'LineWidth', 2); hold off;
legend('Data','Estimated','True','Location','Best');
axis([-1, 11,-0.1,1.1]);
xlabel('x');
ylabel('y');
```

As we can see from the code, the true parameter is $\boldsymbol{\theta} = [1,\ -5]^T$, and the corresponding logistic function is shown in red. The blue circles are the random samples drawn from the Bernoulli distribution with $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ as the posterior probability (See Equation (2.80)). Therefore, small $\boldsymbol{x}$ has more samples coming from class 0, and large $\boldsymbol{x}$ has more samples coming from class 1. The black curve is the logistic model estimated by solving the convex optimization problem. The black curve is not exactly the same as the red curve, because we are only using $n = 200$ samples to train the logistic model. In terms of runtime, `CVX` takes approximately 20 seconds on this toy problem. For real practical usage of logistic regression, there are more customized optimization libraries using accelerated gradient methods instead of the interior point method in `CVX`.

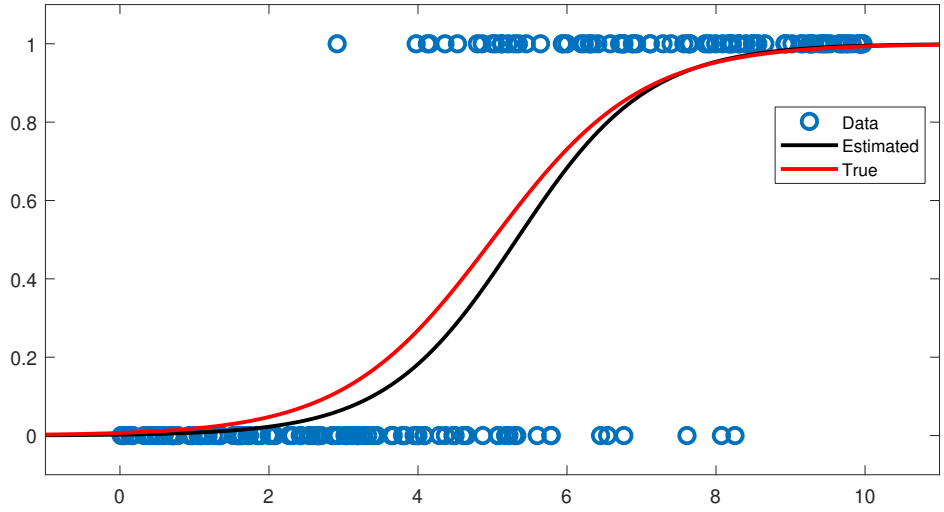


Figure 2.31: Logistic regression using `CVX`. Example provided by CVX user guideline, and the *Convex Optimization* textbook chapter 7.1.

Connection with Bayes

We can now draw connection between logistic regression and Bayes. Our approach is to start from Bayes, and derive the equivalent logistic regression.

Recall that in a two-class Bayes, the conditional distributions are

$$p_{\boldsymbol{X}|Y}(\boldsymbol{x}|i) = \frac{1}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}|}}\exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right\}, \quad \text{and} \quad p_Y(i) = \pi_i$$

for $i = 0, 1$. The posterior distribution is therefore

$$\begin{aligned} p_{Y|\boldsymbol{X}}(1|\boldsymbol{x}) &= \frac{p_{\boldsymbol{X}|Y}(\boldsymbol{x}|1)p_Y(1)}{p_{\boldsymbol{X}|Y}(\boldsymbol{x}|1)p_Y(1)+p_{\boldsymbol{X}|Y}(\boldsymbol{x}|0)p_Y(0)} \\ &= \frac{1}{1+\frac{p_{\boldsymbol{X}|Y}(\boldsymbol{x}|0)p_Y(0)}{p_{\boldsymbol{X}|Y}(\boldsymbol{x}|1)p_Y(1)}} = \frac{1}{1+\exp\left\{-\log\left(\frac{p_{\boldsymbol{X}|Y}(\boldsymbol{x}|1)p_Y(1)}{p_{\boldsymbol{X}|Y}(\boldsymbol{x}|0)p_Y(0)}\right)\right\}} \\ &= \frac{1}{1+\exp\left\{-\log\left(\frac{\pi_1}{\pi_0}\right)-\log\left(\frac{p_{\boldsymbol{X}|Y}(\boldsymbol{x}|1)}{p_{\boldsymbol{X}|Y}(\boldsymbol{x}|0)}\right)\right\}}. \end{aligned}$$

Let us look at the second log term in the denominator.

$$\begin{aligned}\log\left(\frac{p_{\boldsymbol{X}|Y}(\boldsymbol{x}|1)}{p_{\boldsymbol{X}|Y}(\boldsymbol{x}|0)}\right) &= \log\left(\frac{\frac{1}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}|}}\exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_1)\right\}}{\frac{1}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}|}}\exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_0)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_0)\right\}}\right)\\ &= -\frac{1}{2}\Big[(\boldsymbol{x}-\boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_1) - (\boldsymbol{x}-\boldsymbol{\mu}_2)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_2)\Big]\\ &= (\boldsymbol{\mu}_1-\boldsymbol{\mu}_0)^T\boldsymbol{\Sigma}^{-1}\boldsymbol{x} - \frac{1}{2}\left(\boldsymbol{\mu}_1^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_2\right).\end{aligned}$$

By defining $(\boldsymbol{w}, w_0)$ as

$$\begin{aligned}\boldsymbol{w} &= \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_0)\\ w_0 &= -\frac{1}{2}\left(\boldsymbol{\mu}_1^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_0\right) + \log\left(\frac{\pi_1}{\pi_0}\right),\end{aligned} \tag{2.86}$$

we can then show that the posterior is equivalent to

$$p_{Y|\boldsymbol{X}}(1|\boldsymbol{x}) = \frac{1}{1+\exp\{-(\boldsymbol{w}^T\boldsymbol{x}+w_0)\}}.$$

Theorem 12 (Bayesian Posterior = Logistic Hypothesis Function)**.** *For a Gaussian Bayes with equal variance $\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}$, its posterior distribution $p_{Y|\boldsymbol{X}}(i|\boldsymbol{x})$ is exactly the same as the hypothesis function of the logistic regression:*

$$\begin{aligned}p_{Y|\boldsymbol{X}}(1|\boldsymbol{x}) &= \frac{1}{1+\exp\{-(\boldsymbol{w}^T\boldsymbol{x}+w_0)\}} = h_{\boldsymbol{\theta}}(\boldsymbol{x})\\ p_{Y|\boldsymbol{X}}(0|\boldsymbol{x}) &= \frac{\exp\{-(\boldsymbol{w}^T\boldsymbol{x}+w_0)}{1+\exp\{-(\boldsymbol{w}^T\boldsymbol{x}+w_0)\}} = 1-h_{\boldsymbol{\theta}}(\boldsymbol{x}),\end{aligned} \tag{2.87}$$

where $\boldsymbol{\theta} \stackrel{def}{=} (\boldsymbol{w}, w_0)$ is defined by Equation (2.86).

Note also that when making decision in Bayes, the posterior comparison $p_{Y|\boldsymbol{X}}(1|\boldsymbol{x}) > p_{Y|\boldsymbol{X}}(0|\boldsymbol{x})$ is equivalent to checking $\boldsymbol{w}^T\boldsymbol{x}+w_0 > 0$. Equation (2.87) then suggest that $h_{\boldsymbol{\theta}}(\boldsymbol{x}) > 1-h_{\boldsymbol{\theta}}(\boldsymbol{x}) \Leftrightarrow \boldsymbol{w}^T\boldsymbol{x}+w_0 > 0$. From here we can draw a few observations.

- Bayesian decision rule is a **generative** approach: We use $p_{\boldsymbol{X}|Y}(\boldsymbol{x}|i)$ to describe how $\boldsymbol{X}$ is generated conditionally from Y. Logistic regression is a **discriminative** approach: We directly estimate the posterior distribution $p_{Y|\boldsymbol{X}}(i|\boldsymbol{x})$, which is the ultimate quantity Bayesian decision uses to determine the class. Linear regression is also discriminative in this sense, as the hypothesis function $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T\boldsymbol{x}+w_0)$ can be considered as the hard threshold of the posterior.

- Bayesian can return both linear and non-linear classifiers, depending on $\boldsymbol{\Sigma}_i$. Logistic and linear regression can only return a linear classifier. Logistic regression can handle priors π_1 and π_0 that are not equal, but linear regression assumes $\pi_0 = \pi_1 = 1/2$. The three are the same only when $\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}$, and $\pi_1 = \pi_0$.

- Bayesian, logistic and linear return the same classifier **asymptotically**, i.e., when the number of training samples goes to infinity. For very few samples, Bayesian tends to work better (Ng and Jordan 2002) because the Bayesian modeling add bias to the classifier which are good for fewer samples. When the Gaussian modeling does not fit the data, logistic and linear regression works better.

2.5 Perceptron Algorithm

Logistic regression uses a logistic function to define the hypothesis function, which is a two-state hypothesis with a smooth transition from 0 to 1. It is natural to ask what will happen if we make the transition sharp. In other words, is it possible to directly use a sign function to define the hypothesis function, i.e., define $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ as

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T\boldsymbol{x} + w_0), \tag{2.88}$$

where $\boldsymbol{\theta} \stackrel{\text{def}}{=} (\boldsymbol{w}, w_0)$. If we can do that, then maybe we can make the decision boundary sharp.

An immediate thought about the problem will lead to two questions. First, by using the sign function we change the label from $\{0, 1\}$ to $\{-1, +1\}$, because the sign returns either a positive sign or a negative sign. This is not a big issue, because the labeling of the classes is arbitrary. Second, how should we define the training loss function? This is a more important problem. For the hypothesis function we just defined, one possible choice of the loss is

$$J(\boldsymbol{\theta}) = \sum_{j=1}^{n} \max\Big\{ - y_j h_{\boldsymbol{\theta}}(\boldsymbol{x}_j), 0\Big\}. \tag{2.89}$$

Why should we choose this loss function? At the minimal level, this loss function has everything we want. We note that $h_{\boldsymbol{\theta}}(\boldsymbol{x}_j)$ is either -1 or $+1$. If the decision is correct, then we must have either $h_{\boldsymbol{\theta}}(\boldsymbol{x}_j) = +1$ and $y_j = +1$, or $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = -1$ and $y_j = -1$. In both cases, $y_j h_{\boldsymbol{\theta}}(\boldsymbol{x}_j) = +1$, and so the loss $\max\{-y_j h_{\boldsymbol{\theta}}(\boldsymbol{x}_j), 0\}$ is 0. If the decision is wrong, then we must have either $h_{\boldsymbol{\theta}}(\boldsymbol{x}_j) = +1$ and $y_j = -1$, or $h_{\boldsymbol{\theta}}(\boldsymbol{x}_j) = -1$ and $y_j = +1$. No matter which happens, we will have $y_j h_{\boldsymbol{\theta}}(\boldsymbol{x}_j) = -1$, and so the loss is 1. Therefore, if the two classes are linearly separable, then by minimizing $J(\boldsymbol{\theta})$, we are guaranteed that the hypothesis function $h_{\boldsymbol{\theta}}(\boldsymbol{x}_j)$ will match the label y_j. And in this case, the optimal $J(\boldsymbol{\theta}^*)$ will reach 0.

What could be wrong about Equation (2.89)? While Equation (2.89) makes sense from the loss function perspective, it is computationally very difficult because the gradient of a

sign function is zero everywhere except at 0. Therefore, if we try to use gradient descent (or other gradient based algorithms) to numerically compute a solution, we will not be able to do it. So the question now is can we relax the sign function somewhat so that we can actually solve the problem. This leads to the **perceptron** loss function, which uses the discriminant function $g_{\boldsymbol{\theta}}(\boldsymbol{x})$ rather than the hypothesis function $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ when defining the loss. For linear classifiers, recall that the discriminant function $g_{\boldsymbol{\theta}}(\boldsymbol{x})$ is a linear function:

$$g_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0. \tag{2.90}$$

The associated loss function is then given by the definition below.

Definition 4 (Perceptron Loss Function I)**.** *The loss function of training a perceptron algorithm is*

$$J(\boldsymbol{\theta}) = \sum_{j=1}^{n} \max\Big\{ -y_j g_{\boldsymbol{\theta}}(\boldsymbol{x}_j),\ \ 0\Big\}, \tag{2.91}$$

where $y_j \in \{+1, -1\}$*, and* $g_{\boldsymbol{\theta}}(\boldsymbol{x}) \stackrel{def}{=} \boldsymbol{w}^T\boldsymbol{x} + w_0$*.*

The above training loss function is called the **perceptron loss function**, originally developed by Frank Rosenblatt in 1957. The interpretation of the perceptron loss can be seen from two cases:

- $y_j = +1$ and $g_{\boldsymbol{\theta}}(\boldsymbol{x}) > 0$ OR $y_j = -1$ and $g_{\boldsymbol{\theta}}(\boldsymbol{x}) < 0$: In this case, $y_j g_{\boldsymbol{\theta}}(\boldsymbol{x}) > 0$, and the pointwise maximum with 0 will make the loss to 0. This makes sense because $g_{\boldsymbol{\theta}}(\boldsymbol{x}) > 0$ means $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = +1$, which matches with $y_j = +1$.
- $y_j = +1$ and $g_{\boldsymbol{\theta}}(\boldsymbol{x}) < 0$ OR $y_j = -1$ and $g_{\boldsymbol{\theta}}(\boldsymbol{x}) > 0$: When this happens, $y_j g_{\boldsymbol{\theta}}(\boldsymbol{x}) < 0$, and so the pointwise maximum will return $\max(-y_j g_{\boldsymbol{\theta}}(\boldsymbol{x}), 0) = -y_j g_{\boldsymbol{\theta}}(\boldsymbol{x})$.

As we can see, the loss function is capturing the **misclassified** samples. If sample $\boldsymbol{x}_j$ is misclassified, then the loss-function will move the classifier until the $\boldsymbol{x}_j$ is correctly classified (assuming that the two classes are linearly separable).

Because that the loss function is capturing the misclassified samples, when the perceptron algorithm was first developed the loss function was defined using the set of misclassifications. The lemma below shows why such definition is equivalent to our definition.

Lemma 2 (Perceptron Loss Function II)**.** *The perceptron loss function is equivalent to*

$$J(\boldsymbol{\theta}) = -\sum_{j\in\mathcal{M}(\boldsymbol{\theta})} y_j g_{\boldsymbol{\theta}}(\boldsymbol{x}_j), \tag{2.92}$$

where $\mathcal{M}(\boldsymbol{\theta}) \subseteq \{1, \ldots, m\}$ *is the set of misclassified samples.*

Here, we emphasize that the misclassification set $\mathcal{M}(\boldsymbol{\theta})$ is a function of the parameter $\boldsymbol{\theta}$: If $\boldsymbol{\theta}$ changes, the misclassification set $\mathcal{M}(\boldsymbol{\theta})$ also changes.

Proof. A sample is misclassified if and only if sign $\{g_{\boldsymbol{\theta}}(\boldsymbol{x}_j)\} \neq y_j$. Thus, $j \in \mathcal{M}(\boldsymbol{\theta})$ if and only if $\max\{-y_j g_{\boldsymbol{\theta}}(\boldsymbol{x}_j), \quad 0\} = -y_j g_{\boldsymbol{\theta}}(\boldsymbol{x}_j)$. Similarly, a sample is correctly classified if and only if sign $\{g_{\boldsymbol{\theta}}(\boldsymbol{x}_j)\} = y_j$. Thus, $j \notin \mathcal{M}(\boldsymbol{\theta})$ if and only if $\max\{-y_j g_{\boldsymbol{\theta}}(\boldsymbol{x}_j), \quad 0\} = -0$. This proves the lemma. □

Remark. [Hinge Loss and ReLU]. The function $f(s) = \max(-s, 0)$ we use to define the perceptron loss is called the **Hinge Loss**. One can easily prove that the gradient of $f(\boldsymbol{s}) = \max(-\boldsymbol{x}^T\boldsymbol{s}, 0)$ is

$$\nabla_{\boldsymbol{s}} \max(-\boldsymbol{x}^T\boldsymbol{s}, 0) = \begin{cases} -\boldsymbol{x}, & \text{if} \quad \boldsymbol{x}^T\boldsymbol{s} < 0, \\ 0, & \text{if} \quad \boldsymbol{x}^T\boldsymbol{s} \geq 0. \end{cases}$$

A closely related function in the neural network literature is called the **rectified linear unit** (ReLU), defined as $f(s) = \max(s, 0)$. Its gradient can be derived analogously as the Hinge Loss case. The plots are shown below.

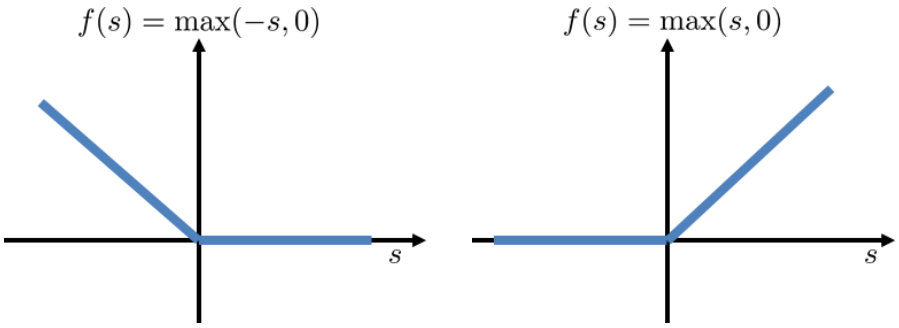


Figure 2.32: Two closely related functions in neural network / perceptron algorithms. [Left] Hinge Loss; [Right] ReLU.

What is so special about the loss function? It turns out that the perceptron loss function is **convex**. Such result should not be of any surprise, because the Hinge-Loss is a convex function. The following lemma just makes this observation more formal.

Lemma 3 (Convexity of Perceptron Loss Function)**.** *The perceptron loss function is convex in* $\boldsymbol{\theta}$*, i.e., for any* $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \in domJ$*,*

$$J(\lambda\boldsymbol{\theta}_1 + (1-\lambda)\boldsymbol{\theta}_2) \leq \lambda J(\boldsymbol{\theta}_1 + (1-\lambda)J(\boldsymbol{\theta}_2), \tag{2.93}$$

for any $\lambda \in [0, 1]$.

Proof. We prove convexity for the Hinge-Loss form of the loss function. Since $y_j h_{\boldsymbol{\theta}}(\boldsymbol{x}_j) = y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0) = (y_j\boldsymbol{x}_j)^T\boldsymbol{w} + y_j w_0$ which is linear in $(\boldsymbol{w}, w_0)$, it remains to show that the Hinge-Loss is convex. Consider the function $f(s) = \max(-s, 0)$. Let $\lambda \in [0,1]$, and consider two points $s_1, s_2 \in \text{dom} f$. Then,

$$\begin{aligned} f(\lambda s_1 + (1-\lambda)s_2) &= \max(\lambda s_1 + (1-\lambda)s_2, 0) \\ &\leq \max\Big\{-\lambda s_1, 0\Big\} + \max\Big\{-(1-\lambda)s_2, 0\Big\} \\ &= \lambda \max(-s_1, 0) + (1-\lambda)\max(-s_2, 0) = \lambda f(s_1) + (1-\lambda) f(s_2). \end{aligned}$$

Since a sum of convex functions remains convex, we conclude that J is convex. □

Convexity of J is crucial here, because if J is convex, then there must exists $\boldsymbol{\theta}^* \in \text{dom} J$ such that $J(\boldsymbol{\theta}^*) \leq J(\boldsymbol{\theta})$ for any $\boldsymbol{\theta} \in \text{dom} J$. That is, there must be a global minimizer $\boldsymbol{\theta}^*$ such that the loss is minimized. Now, if we have an algorithm and if we can prove that the algorithm has a property that $\|\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^*\| \to 0$ as $k \to \infty$, then we can claim convergence. In addition, if the two classes are linearly separable, then the global minimum is achieved when $J(\boldsymbol{\theta}^*) = 0$, i.e., every sample is correctly classified.

Caution. [Non-uniqueness of Global Minimizer]. Being convex does not guarantee that the global minimizer is unique. In fact, if $\boldsymbol{\theta}$ is a minimizer of the loss function J, then any scalar multiple $\alpha\boldsymbol{\theta}$ for some constant $\alpha > 0$ will also be a global minimizer, because

$$g_{\alpha\boldsymbol{\theta}}(\boldsymbol{x}) = (\alpha\boldsymbol{w})^T\boldsymbol{x} + (\alpha w_0) = \alpha(\boldsymbol{w}^T\boldsymbol{x} + w_0),$$

which does not alter the sign of $\boldsymbol{w}^T\boldsymbol{x} + w_0$ as long as $\alpha > 0$.

Perceptron Algorithm

Now that we know the perceptron loss function is convex, we are guaranteed that any convex solver will find a solution. In what follows, we will discuss a simple gradient descent algorithm to minimize the loss.

If at the k-th iteration we fix $\mathcal{M}(\boldsymbol{\theta})$ as $\mathcal{M}(\boldsymbol{\theta}^{(k)})$ (denote as $\mathcal{M}_k$), then the $(k+1)$-th gradient descent estimate is given by

$$\begin{aligned} \boldsymbol{\theta}^{(k+1)} &= \boldsymbol{\theta}^{(k)} - \alpha_k \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(k)}) \\ &= \boldsymbol{\theta}^{(k)} - \alpha_k \sum_{j \in \mathcal{M}_k} \nabla_{\boldsymbol{\theta}}\Big(-y_j g_{\boldsymbol{\theta}}(\boldsymbol{x}_j)\Big). \end{aligned} \tag{2.94}$$

The gradient $\nabla_{\boldsymbol{\theta}}\Big(-y_j g_{\boldsymbol{\theta}}(\boldsymbol{x}_j)\Big)$ can be computed as

$$\nabla_{\boldsymbol{\theta}}\Big(-y_j g_{\boldsymbol{\theta}}(\boldsymbol{x}_j)\Big) = \begin{cases} -y_j \nabla_{\boldsymbol{\theta}}\Big(\boldsymbol{w}^T\boldsymbol{x}_j + w_0\Big) = -y_j \begin{bmatrix} \boldsymbol{x}_j \\ 1 \end{bmatrix} & \text{if} \quad j \in \mathcal{M}_k, \\ 0, & \text{if} \quad j \notin \mathcal{M}_k, \end{cases} \tag{2.95}$$

Therefore, the overall algorithm is given by

$$\begin{bmatrix} \boldsymbol{w}^{(k+1)} \\ w_0^{(k+1)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{w}^{(k)} \\ w_0^{(k)} \end{bmatrix} + \alpha_k \sum_{j \in \mathcal{M}_k} \begin{bmatrix} y_j \boldsymbol{x}_j \\ y_j \end{bmatrix}.$$

At the next iteration, we first update the misclassification set $\mathcal{M}_k$, and then recompute the gradient again. Such algorithm is known as the **perceptron algorithm**.

Definition 5 (Perceptron Algorithm (Batch Mode))**.** *The perceptron algorithm is defined as*

$$\begin{bmatrix} \boldsymbol{w}^{(k+1)} \\ w_0^{(k+1)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{w}^{(k)} \\ w_0^{(k)} \end{bmatrix} + \alpha_k \sum_{j \in \mathcal{M}_k} \begin{bmatrix} y_j \boldsymbol{x}_j \\ y_j \end{bmatrix}, \tag{2.96}$$

for some step size α_k, where $\mathcal{M}_k$ is the misclassification set at the k-th iteration, i.e., $\mathcal{M}_k = \{j \mid h_{\boldsymbol{\theta}^{(k)}}(\boldsymbol{x}_j) \neq y_j\}$.

Example. A pictorial illustration of the perceptron algorithm is shown in Figure 2.33. There are two classes of data points: $+1$ and -1. The k-th iterate of the perceptron algorithm gives a decision boundary with a normal $\boldsymbol{w}^{(k)}$. With this decision boundary, data points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are misclassified. The perceptron algorithm therefore computes a gradient $y_1\boldsymbol{x}_1 + y_2\boldsymbol{x}_2$, and adds it to the current weight $\boldsymbol{w}^{(k)}$ to yield a new normal vector

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + \alpha_k \left(y_1\boldsymbol{x}_1 + y_2\boldsymbol{x}_2\right).$$

This normal vector $\boldsymbol{w}^{(k+1)}$ defines a new decision boundary, and as the figure all the samples are now correctly classified.

Caution. It is possible that $\boldsymbol{w}^{(k+1)}$ will make part of the correctly classified samples misclassified. In this case, the algorithm will repeat until all samples are correctly classified, which will eventually happen if the data is linearly separable.

In Equation (2.96), we update $\boldsymbol{\theta}^{(k)}$ using all the misclassified points at the current iteration. We call this algorithm the **batch mode** preceptron algorithm. It is also possible to define the perceptron algorithm using an **online mode**, which updates one single data point per every iteration. The difference between the batch mode and the online mode is that for online mode, the misclassification set $\mathcal{M}_k$ only contains one sample, i.e., $|\mathcal{M}_k| = 1$ for all k. In this case, the perceptron algorithm is simplified to

Definition 6 (Perceptron Algorithm (Online Mode))**.** *The online perceptron algorithm is defined as*

$$\begin{bmatrix} \boldsymbol{w}^{(k+1)} \\ w_0^{(k+1)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{w}^{(k)} \\ w_0^{(k)} \end{bmatrix} + \alpha_k \begin{bmatrix} y_j \boldsymbol{x}_j \\ y_j \end{bmatrix}, \tag{2.97}$$

for some step size α_k, and where $j \in \mathcal{M}_k$ is the index of any misclassified sample in $\mathcal{M}_k$.

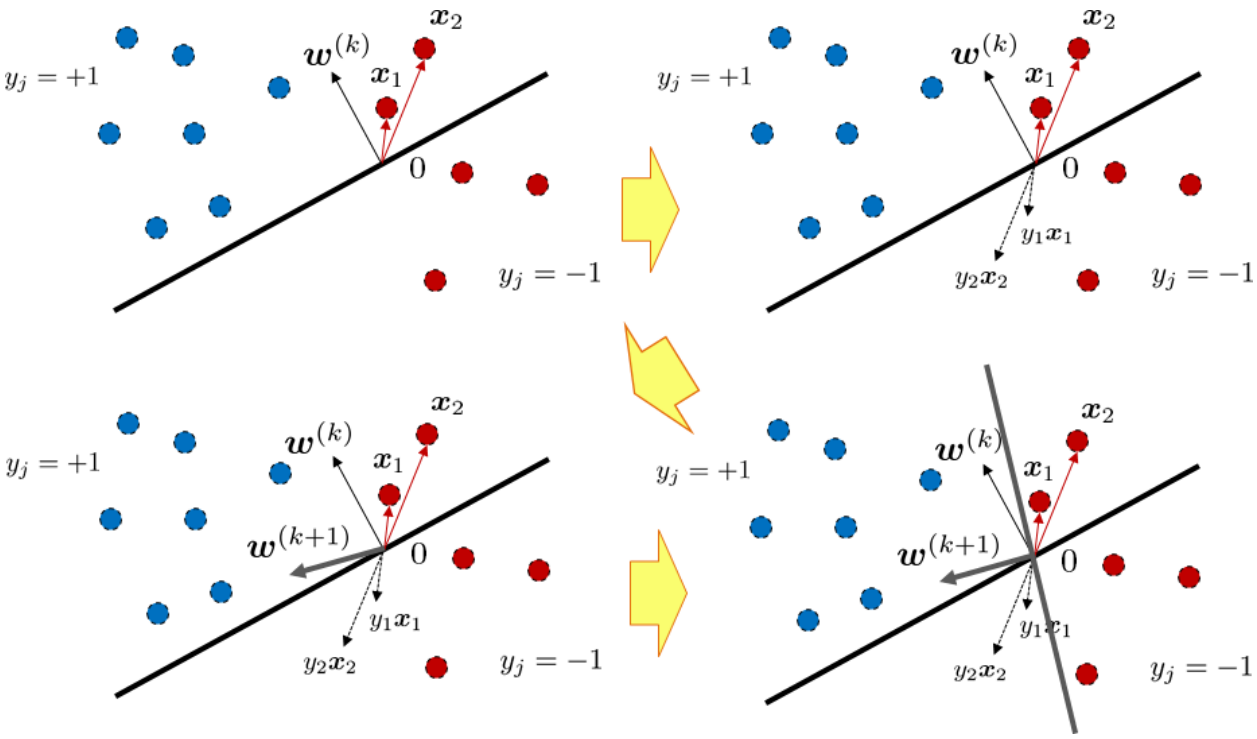


Figure 2.33: Perceptron algorithm (Batch Mode): At iteration k, the normal vector $\boldsymbol{w}^{(k)}$ causes two misclassified points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. The perceptron algorithm then constructs a new update $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + \alpha \sum_{j \in \mathcal{M}_k} y_j \boldsymbol{x}_j$, where in this case $\mathcal{M}_k = \{1, 2\}$. By perturbing $\boldsymbol{w}^{(k)}$, we can now see that $\boldsymbol{w}^{(k+1)}$ is able to correctly classify all the samples.

If the set of misclassification samples is $\mathcal{M}_k$, then at every iteration the online perceptron algorithm has to pick one and only one sample from $\mathcal{M}_k$. If the sample picking procedure is randomized, i.e., the sample we pick from $\mathcal{M}_k$ has certain probability, then the online perceptron algorithm will become a **stochastic gradient descent** algorithm. Many theoretical results about the stochastic gradient descent will then become applicable, e.g., the

stochastic gradient is asymptotically the same the expectation of the samples in the set $\mathcal{M}_k$.

Exercise 5.1. (AML, pp.8) Consider a simplified Online Perceptral Algorithm by absorbing w_0 into $\boldsymbol{w}$ and setting $\alpha_k = 1$:

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + y_j \boldsymbol{x}_j, \qquad j \in \mathcal{M}_k. \tag{2.98}$$

We want to show that this rule moves the direction of the classifying $\boldsymbol{x}_j$ correctly.

(i) Show that $y_j(\boldsymbol{w}^{(k+1)})^T \boldsymbol{x}_j < 0$. Hint: $\boldsymbol{x}_j$ is misclassified by $\boldsymbol{w}^{(k)}$.

(ii) Show that $y_j(\boldsymbol{w}^{(k+1)})^T \boldsymbol{x}_j > y_j(\boldsymbol{w}^{(k)})^T \boldsymbol{x}_j$.

(iii) As far as classify $\boldsymbol{x}_j$ is concerned, argue that the move from $\boldsymbol{w}^{(k)}$ to $\boldsymbol{w}^{(k+1)}$ is a move in the right direction.

Convergence

Now we can present the convergence proof. To make the convergence proof work, we need two assumptions. The first assumption is that the two classes are **linearly separable**, for otherwise the algorithm will not be able to find a solution $\boldsymbol{\theta}^*$: The loss function will never reach $J(\boldsymbol{\theta}) = 0$, because there exists at least one y_j which does not have the same sign with $g_{\boldsymbol{\theta}}(\boldsymbol{x}_j)$. Therefore, in order to claim convergence to $\boldsymbol{\theta}^*$, it is necessary to assume that the data is linearly separable. If we assume that linear separability, then there must exists a margin $\gamma > 0$ such that $y_j(\boldsymbol{\theta}^*)^T \boldsymbol{x}_j \geq \gamma$ for all j.

The second assumption is the boundedness of $\boldsymbol{x}_j$, which is typically mild.

Theorem 13 (Convergence of Perceptron Algorithm)**.** *Assume that the two classes are linearly separable so that $y_j(\boldsymbol{\theta}^*)^T \boldsymbol{x}_j \geq \gamma$ for some $\gamma > 0$. Also, assume that $\|\boldsymbol{x}_j\|_2 \leq R$ for some constant R. Let $\boldsymbol{\theta}^{(0)} = \mathbf{0}$. Then, batch mode perceptron algorithm converges to the true solution $\boldsymbol{\theta}^*$*

$$\|\boldsymbol{\theta}^{(k+1)} - \boldsymbol{\theta}^*\|^2 = 0, \tag{2.99}$$

when the number of iterations k exceeds the following upper limit

$$k \leq \sum_{i=1}^{k} |\mathcal{M}_i| < \frac{\max_j \|\boldsymbol{\theta}^*\|^2 \|\boldsymbol{x}_j\|^2}{\min_j \left((\boldsymbol{\theta}^*)^T \boldsymbol{x}_j\right)^2} = \frac{\|\boldsymbol{\theta}^*\|^2 R^2}{\gamma^2}. \tag{2.100}$$

The theorem we present here is general in the sense that it covers both the batch mode and the online mode convergence. For **online mode**, the number of iterations is upper bounded by $\frac{\|\boldsymbol{\theta}^*\|^2 R^2}{\gamma^2}$, which is typically more than the number of samples in the training dataset. Therefore, online perceptron algorithm will visit the samples more than once during

the iterations. For **batch mode**, if we assume $|\mathcal{M}_i|$ is distributed according to certain probability distribution, then we can roughly claim that $\sum_{i=1}^{k}|\mathcal{M}_i| \approx k\mathbb{E}[|\mathcal{M}_i|]$. Thus, the number of iteration is now upper bounded by $\frac{1}{\mathbb{E}[|\mathcal{M}_i|]}\frac{\|\boldsymbol{\theta}^*\|^2R^2}{\gamma^2}$, which is smaller than the online mode by a factor of $\mathbb{E}[|\mathcal{M}_i|]$. $\mathbb{E}[\mathcal{M}_i]$ measures the average number of misclassified samples at every iteration. If it is the online mode, then $|\mathcal{M}_i| = 1$ for all i. For batch mode, the general expectation is that $\mathbb{E}[\mathcal{M}_i] > 1$.

Proof. Without loss of generality, we assume $w_0 = 0$. Otherwise we can just replace $\boldsymbol{x}_j$ by $[\boldsymbol{x}_j^T, 1]^T$. We also define the sum of the misclassified $\boldsymbol{x}_j$:

$$\overline{\boldsymbol{x}}^{(k)} = \sum_{j\in\mathcal{M}_k}\boldsymbol{x}_j. \tag{2.101}$$

Since $J(\boldsymbol{\theta})$ is convex in $\boldsymbol{\theta}$, there exists $\boldsymbol{\theta}^* \in \text{dom}J$ such that $J(\boldsymbol{\theta}^*)$ is minimized. For this $\boldsymbol{\theta}^*$, we can show that

$$\begin{aligned}\|\boldsymbol{\theta}^{(k+1)} - \boldsymbol{\theta}^*\|^2 &= \|\boldsymbol{\theta}^{(k)} + \alpha_k\overline{\boldsymbol{x}}^{(k)} - \boldsymbol{\theta}^*\|^2 = \|(\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^*) + \alpha_k\overline{\boldsymbol{x}}^{(k)}\|^2\\ &= \|\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^*\|^2 + 2\alpha_k(\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^*)^T\overline{\boldsymbol{x}}^{(k)} + \alpha_k^2\|\overline{\boldsymbol{x}}^{(k)}\|^2\\ &= \|\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^*\|^2 + 2\alpha_k\left(\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^*\right)^T\left(\sum_{j\in\mathcal{M}_k}y_j\boldsymbol{x}_j\right) + \alpha_k^2\left\|\sum_{j\in\mathcal{M}_k}y_j\boldsymbol{x}_j\right\|^2.\end{aligned}$$

Since, by construction, $\boldsymbol{\theta}^{(k)}$ updates only the misclassified samples (during the k-th iteration), for any $j \in \mathcal{M}_k$ we must have $(\boldsymbol{\theta}^{(k)})^T(y_j\boldsymbol{x}_j) \le 0$. This implies that the inner product with $\overline{\boldsymbol{x}}^{(k)}$, which is the sum of $(\boldsymbol{\theta}^{(k)})^T(y_j\boldsymbol{x}_j)$, is also non-positive:

$$(\boldsymbol{\theta}^{(k)})^T\overline{\boldsymbol{x}}^{(k)} = \sum_{j\in\mathcal{M}_k}(\boldsymbol{\theta}^{(k)})^Ty_j\boldsymbol{x}_j \le 0. \tag{2.102}$$

Therefore, we can show that

$$\|\boldsymbol{\theta}^{(k+1)} - \boldsymbol{\theta}^*\|^2 \le \|\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^*\|^2 - 2\alpha_k(\boldsymbol{\theta}^*)^T\overline{\boldsymbol{x}}^{(k)} + \alpha_k^2\|\overline{\boldsymbol{x}}^{(k)}\|^2.$$

The sum of the last two terms can be rewritten as

$$-2\alpha_k(\boldsymbol{\theta}^*)^T\overline{\boldsymbol{x}}^{(k)} + \alpha_k^2\|\overline{\boldsymbol{x}}^{(k)}\|^2 = \alpha_k\left(-2(\boldsymbol{\theta}^*)^T\overline{\boldsymbol{x}}^{(k)} + \alpha_k\|\overline{\boldsymbol{x}}^{(k)}\|^2\right),$$

which is negative if and only if $\alpha_k < \frac{2(\boldsymbol{\theta}^*)^T\overline{\boldsymbol{x}}^{(k)}}{\|\overline{\boldsymbol{x}}^{(k)}\|^2}$. (Note that α_k is positive by definition of a step size.) Therefore, if we choose half of that upper bound, i.e.,

$$\alpha_k = \frac{(\boldsymbol{\theta}^*)^T\overline{\boldsymbol{x}}^{(k)}}{\|\overline{\boldsymbol{x}}^{(k)}\|^2}, \tag{2.103}$$

then we are guaranteed that the sum is negative. In this case, we have

$$-2\alpha_k(\boldsymbol{\theta}^*)^T\overline{\boldsymbol{x}}^{(k)} + \alpha_k^2\|\overline{\boldsymbol{x}}^{(k)}\|^2 = -\frac{\left((\boldsymbol{\theta}^*)^T\overline{\boldsymbol{x}}^{(k)}\right)^2}{\|\overline{\boldsymbol{x}}^{(k)}\|^2}.$$

Since by assumption $\|\boldsymbol{x}_j\|^2 \leq R$ for any j, and $y_j(\boldsymbol{\theta}^*)^T\boldsymbol{x}_j \geq \gamma$ for any j, it holds that

$$\frac{\left((\boldsymbol{\theta}^*)^T\overline{\boldsymbol{x}}^{(k)}\right)^2}{\|\overline{\boldsymbol{x}}^{(k)}\|^2} = \frac{\left(\sum_{j\in\mathcal{M}_k} y_j(\boldsymbol{\theta}^*)^T\boldsymbol{x}_j\right)^2}{\sum_{j\in\mathcal{M}_k}\|\boldsymbol{x}_j\|^2} \geq \frac{\left(\sum_{j\in\mathcal{M}_k}\gamma\right)^2}{\sum_{j\in\mathcal{M}_k}R^2} = |\mathcal{M}_k|\frac{\gamma^2}{R^2},$$

where $|\mathcal{M}_k|$ is the size of $\mathcal{M}_k$. Therefore, by induction we can show that

$$\|\boldsymbol{\theta}^{(k+1)} - \boldsymbol{\theta}^*\|^2 < \|\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*\|^2 - \sum_{i=1}^{k}|\mathcal{M}_i|\frac{\gamma^2}{R^2}.$$

The left hand side is a square term, and so it must be non-negative. Also, because $|\mathcal{M}_i| > 0$ for any i (otherwise we have already converged), we can conclude that

$$\sum_{i=1}^{k}|\mathcal{M}_i|\frac{\gamma^2}{R^2} < \|\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*\|^2 = \|\boldsymbol{\theta}^*\|^2, \tag{2.104}$$

if we assume that $\boldsymbol{\theta}^{(0)} = \mathbf{0}$. Therefore, we have

$$k \leq \sum_{i=1}^{k}|\mathcal{M}_i| < \frac{\max_j\|\boldsymbol{\theta}^*\|^2\|\boldsymbol{x}_j\|^2}{\left(\min_j(\boldsymbol{\theta}^*)^T\boldsymbol{x}_j\right)^2} \tag{2.105}$$

□

Exercise 5.2. (AML, pp.33) If we consider a simplified online perceptron algorithm illustrated in Exercise 5.1, we can derive convergence using a similar manner. Let $\boldsymbol{w}^*$ be the ultimate solution, and let $\rho = \min_j y_j(\boldsymbol{w}^{*T})\boldsymbol{x}_j$.

(i) Show that $(\boldsymbol{w}^{(k)})^T\boldsymbol{w}^* \geq (\boldsymbol{w}^{(k-1)})^T\boldsymbol{w}^* + \rho$, and conclude that $(\boldsymbol{w}^{(k)})^T\boldsymbol{w}^* \geq k\rho$.

(ii) Show that $\|\boldsymbol{w}^{(k)}\|^2 \leq |\boldsymbol{w}^{(k-1)}\|^2 + \|\boldsymbol{x}_j\|^2$. Hint: $y_j((\boldsymbol{w}^{(k-1)})^T\boldsymbol{x}_j) \leq 0$ because $\boldsymbol{x}_j$ was misclassified by $\boldsymbol{w}^{(k-1)}$.

(iii) Show that $\|\boldsymbol{w}^{(k)}\|^2 \leq kR^2$, where $R = \max_j\|\boldsymbol{x}_j\|_2$.

(iv) Using (i) and (iii) to show that

$$\frac{(\boldsymbol{w}^{(k)})^T\boldsymbol{w}^*}{\|\boldsymbol{w}^{(k)}\|} \geq \sqrt{k}\frac{\rho}{R}.$$

Hence, show that $k \leq \frac{R^2\|\boldsymbol{w}^*\|^2}{\rho^2}$.

Remark. [Special case: $k \leq 1/\gamma^2$]. If we further assume that $\|\boldsymbol{x}_j\|_2 = 1$ and $\|\boldsymbol{\theta}^*\|_2 = 1$, i.e., both are normalized, then we will reach a special case on the upper bound where

$$k \leq \frac{1}{\gamma^2}. \tag{2.106}$$

The quantity γ can be considered as the margin of the classifier. Note that γ is the lower bound of the inner product $y_j(\boldsymbol{\theta}^*)^T\boldsymbol{x}_j$. Since $y_j(\boldsymbol{\theta}^*)^T\boldsymbol{x}_j$ measures the cosine of the angle between the two vectors, γ sets up a lower bound on the angle, which will give larger margin if the angle is bigger.

Pros and Cons of Perceptron

Perceptron algorithm is a very simple algorithm with guaranteed performance if the two classes are linearly separable. It is simple because we can update the decision boundary iteratively by sweeping through all the points in the dataset.

On the downside, a decision boundary given by the perceptron algorithm is not unique. Depending on the initial condition, and depending on the specific order of visiting the samples, the perceptron algorithm can land on different decision boundaries. Besides, the decision boundary is always touching one of the data points. This could potentially cause problems in robustness, because points too close to the decision boundary can be perturbed and become misclassified.

2.6 Support Vector Machine

The perceptron algorithm is guaranteed to find a hypothesis function (i.e. a separating hyperplane $\mathcal{H}$) if the data is linearly separable. However, depending on the initial guess of the percetron algorithm and the order of how we pick the misclassified data points, the perceptron algorithm can land on different hypothesis functions that are equally optimal according to the perceptron design criterion. Figure 2.34 shows three hypothesis functions that can be returned by a perceptron algorithm. Among the three, a simple eyeball inspection will tell us that the left and the middle hypothesis functions are not good because the decision boundary is too close to a data point. The rightmost hypothesis is better because it makes the spacing between the two classes equal.

The above illustration shows that besides being able to separate the two classes, a good classifier should also be **robust** in the sense that small perturbation of the data point will not drastically change the decision from one class to another. But what do we mean by perturbation? Perturbations are **noise** associated with the observed data. Let us take a closer look. For every data point $\boldsymbol{x}_j$, assume there is certain amount of noise surrounding the data point. The noise creates a ball centered at $\boldsymbol{x}_j$ with a radius δ_j. A robust classifier can tolerate a larger δ_j, whereas a fragile classifier can tolerate only a small δ_j. Figure 2.35

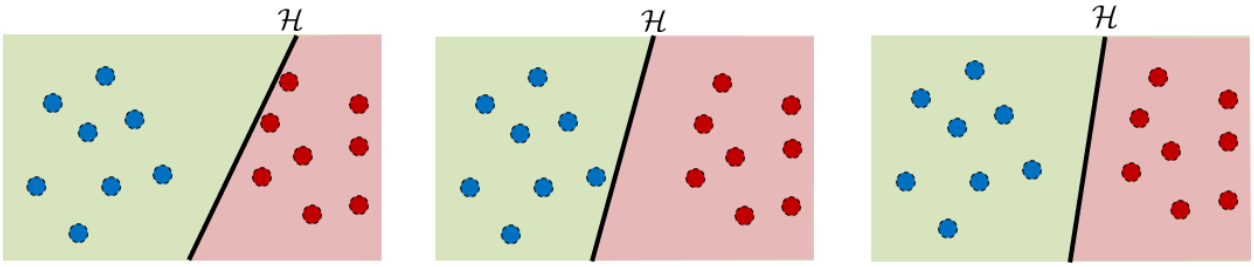


Figure 2.34: Three separating hyperplanes that can be returned by the perceptron algorithm. [Left] A bad separating hyperplane because it is too close to the red class. [Middle] Another bad separating hyperplane because it is too close to the blue class. [Right] A good separating hyperplane because the spacing between the two classes is balanced.

shows two examples. On the left hand side, the allowable perturbation is smaller because there is a blue data point located near the separating hyperplane. On the right hand side, the allowable perturbation is larger because the decision is in the middle of the two closest data points. Therefore, robustness of a classifier can be measured in terms of the largest radius before hitting the separating hyperplane.

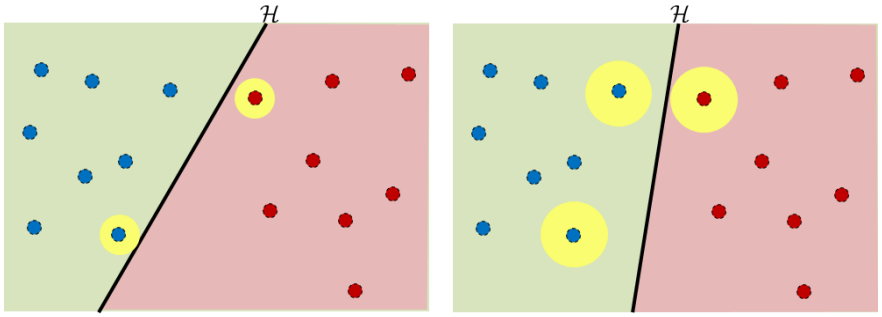


Figure 2.35: Perturbation of data points. For every data point $\boldsymbol{x}_j$, we put a ball centered at $\boldsymbol{x}_j$ with radius δ_j. A robust classifier can tolerate a larger perturbation [Right] whereas a fragile classifier can tolerate a smaller perturbation [Left].

An alternative perspective of quantifying the robustness of the classifier is to consider the **margin**, or a "cushion" (we can also think of it as a safety zone). Given a separating hyperplane, we draw a band which is parallel to the separating hyperplane and treat it as a cushion area. Inside the cushion area there is no data point. We grow the bandwidth of the cushion as large as possible until we hit a data point. The largest bandwidth we can grow is called the margin of the classifier. The margin and the perturbation radius is related, since a bigger margin tolerates a larger perturbation. For example, the two classifiers shown in Figure 2.36 have different bandwidths of its cushion. The classifier on the left is more susceptible to perturbation because its margin is thin, whereas the classifier on the right is more robust to any perturbation occurring to the data point. Therefore, if we want to design a robust classifier, then the design criteria would be to maximize the margin. Such classifier is called a **max margin classifier**. Support vector machine is a max margin classifier.

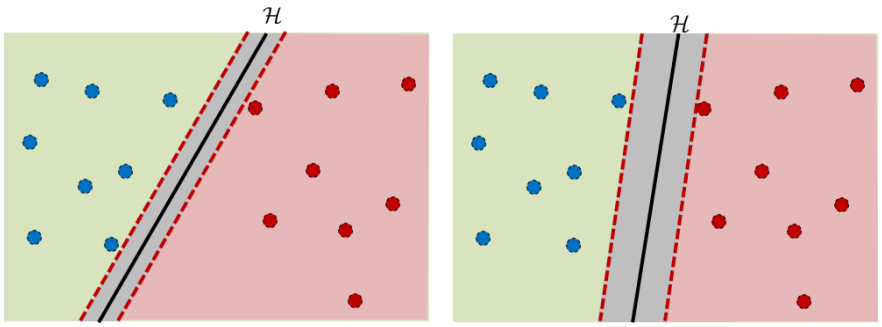


Figure 2.36: Margin of a classifier. For every data point $\boldsymbol{x}_j$, we put a ball centered at $\boldsymbol{x}_j$ and grow its radius until it hits the separating hyperplane. The smallest radius defines the margin of the separating hyperplane. [Left] Small margin because a point is close the separating hyperplane. [Right] Large margin.

The Concept of Margin

Let us go into some mathematical details. Recall that a linear classifier has a discriminant function $g_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0$. The **distance** between a data point $\boldsymbol{x}_j$ and the separating hyperplane $\mathcal{H} = \{\boldsymbol{x} \mid g_{\boldsymbol{\theta}}(\boldsymbol{x}) = 0\}$ is (See Theorem 1)

$$d(\boldsymbol{x}_j, \mathcal{H}) = \frac{|g_{\boldsymbol{\theta}}(\boldsymbol{x}_j)|}{\|\boldsymbol{w}\|_2} = \frac{|\boldsymbol{w}^T\boldsymbol{x}_j + w_0|}{\|\boldsymbol{w}\|_2}.$$

Since there is no sign for this distance, we call $d(\boldsymbol{x}_j, \mathcal{H})$ the **unsigned distance** between $\boldsymbol{x}_j$ and $\mathcal{H}$.

We define the **margin** of the classifier as the **shortest** distance among all the unsigned distances. We must pick the shortest distance as the margin, for otherwise there will be data point staying inside the cushion area. Computationally, if we have a dataset $\mathcal{D} = \{(\boldsymbol{x}_j, y_j)\}_{j=1}^n$, we can first compute all the distances between $\boldsymbol{x}_j$ and $\mathcal{H}$ and then pick the smallest one. This will give us a scalar γ:

$$\gamma = \min_{j=1,\ldots,n} \quad d(\boldsymbol{x}_j, \mathcal{H}). \tag{2.107}$$

We call γ the margin. Without loss of generality we assume that γ is attained by sample $\boldsymbol{x}_n$ (Otherwise we can pick $\boldsymbol{x}_{j^*}$ for some index j^*, which is more cumbersome in terms of notation):

$$\boldsymbol{x}_n = \operatorname*{argmin}_{j=1,\ldots,n} \quad d(\boldsymbol{x}_j, \mathcal{H}). \tag{2.108}$$

Note that γ is technically a function of $\boldsymbol{\theta} = (\boldsymbol{w}, w_0)$, i.e., $\gamma(\boldsymbol{\theta})$. If we change the decision boundary, the margin γ will change accordingly. However, regardless of the choice of $\boldsymbol{\theta}$, γ must exist because $\mathcal{D}$ is a finite set and $d(\cdot, \cdot) \geq 0$. The smallest value γ can be 0, which happens when at least one data point $\boldsymbol{x}_j$ lives on $\mathcal{H}$.

Since $d(\boldsymbol{x}_j, \mathcal{H})$ is unsigned, γ itself does not tell about whether a point $\boldsymbol{x}_j$ is correctly classified or not. It could happen that $\boldsymbol{x}_j$ is on the wrong side of the decision boundary, and

its distance from the decision boundary is large. How can we measure the distance of the correctly classified samples? If we assume that the labels are defined as $y_j \in \{-1, +1\}$, then we can define the **signed distance** between a data point $\boldsymbol{x}_j$ and $\mathcal{H}$:

$$d_{\text{signed}}(\boldsymbol{x}_j, \mathcal{H}) = y_j \left(\frac{\boldsymbol{w}^T \boldsymbol{x}_j + w_0}{\|\boldsymbol{w}\|_2} \right) = \begin{cases} \geq 0, & \text{correctly classify } \boldsymbol{x}_j \\ < 0, & \text{incorrectly classify } \boldsymbol{x}_j. \end{cases} \tag{2.109}$$

To see how this definition works, we note that if $\boldsymbol{x}_j$ belongs to class $\mathcal{C}_{+1}$ and $y_j = +1$, then $\boldsymbol{w}^T \boldsymbol{x}_j + w_0 \geq 0$ and so $y_j(\boldsymbol{w}^T \boldsymbol{x}_j + w_0) \geq 0$. Similar argument holds for $\boldsymbol{x}_j$ belonging to class $\mathcal{C}_{-1}$ and $y_j = -1$. When this happens, we have $d_{\text{signed}}(\boldsymbol{x}_j, \mathcal{H}) = d(\boldsymbol{x}_j, \mathcal{H})$. However, if $\boldsymbol{x}_j$ belongs to $\mathcal{C}_{+1}$ and $y_j = -1$ or vice versa, then $y_j(\boldsymbol{w}^T \boldsymbol{x}_j + w_0) \leq 0$. Figure 2.37 below illustrates the difference between an unsigned distance and the a signed distance. A signed distance can tell (i) how big the margin is, (ii) is the point correctly classifier.

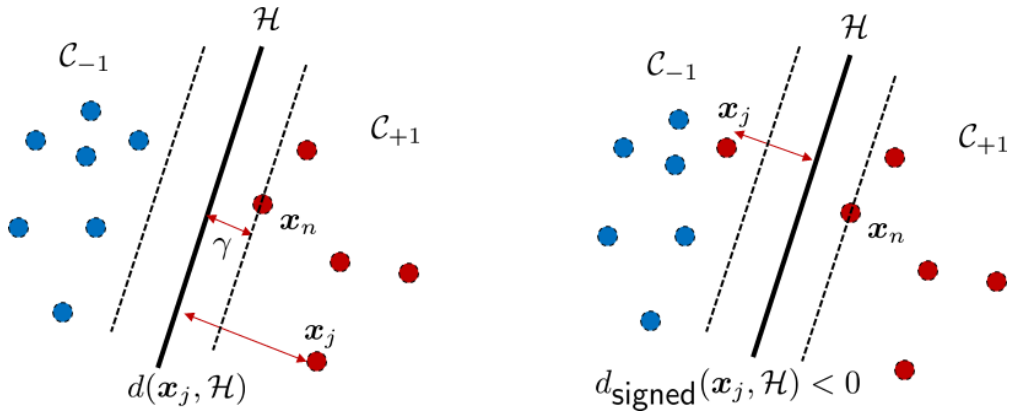


Figure 2.37: [Left] Definitions of the margin γ. Among all the data points $\{\boldsymbol{x}_j\}_{j=1}^n$, we pick the one that is closest to the decision boundary $\mathcal{H}$. Call this point $\boldsymbol{x}_n$. Then, the margin is defined as $d(\boldsymbol{x}_n, \mathcal{H})$. [Right] Definition of the signed distance. If a data point $\boldsymbol{x}_j$ lives on the wrong side of the decision boundary, its signed distance $d_{\text{signed}}(\boldsymbol{x}_j, \mathcal{H})$ is negative. The SVM constraint does not allow this happen by seeking the best decision boundary. If the two classes are linearly separable, then an optimal decision must exists.

SVM via Quadratic Optimization

We are now ready to formulate the SVM problem as an optimization. Let us assume that the two classes are linearly separable with a non-trivial gap in between. Under this assumption, we are guaranteed to have a non-trivial margin γ. Therefore, to build a successful SVM, we must require all the data points stay away from the decision boundary by at least a distance of γ:

$$y_j \left(\frac{\boldsymbol{w}^T \boldsymbol{x}_j + w_0}{\|\boldsymbol{w}\|_2} \right) \geq \gamma, \quad j = 1, \ldots, n.$$

If any of these conditions is violated, then there will be a misclassification.

As we discussed when we introduced the concept of margin, γ is a function of $\boldsymbol{\theta} = (\boldsymbol{w}, w_0)$ and so γ is $\gamma(\boldsymbol{\theta})$. By varying the optimization variable $\boldsymbol{\theta}$, we will change $\gamma(\boldsymbol{\theta})$. The optimal γ is obtained when it is **maximized**, leading to a maximum-margin classifier. Therefore, we consider the optimization

$$\begin{aligned} &\underset{\boldsymbol{w}, w_0}{\text{maximize}} \;\; \gamma \\ &\text{subject to} \;\; y_j \left(\frac{\boldsymbol{w}^T \boldsymbol{x}_j + w_0}{\|\boldsymbol{w}\|_2} \right) \geq \gamma, \quad j = 1, \ldots, n. \end{aligned} \tag{2.110}$$

A good solution to this optimization should achieve two goals: (i) We need to maximize the margin γ; (ii) We need to satisfy all the constraints. Figure 2.38 shows a few good and bad examples.

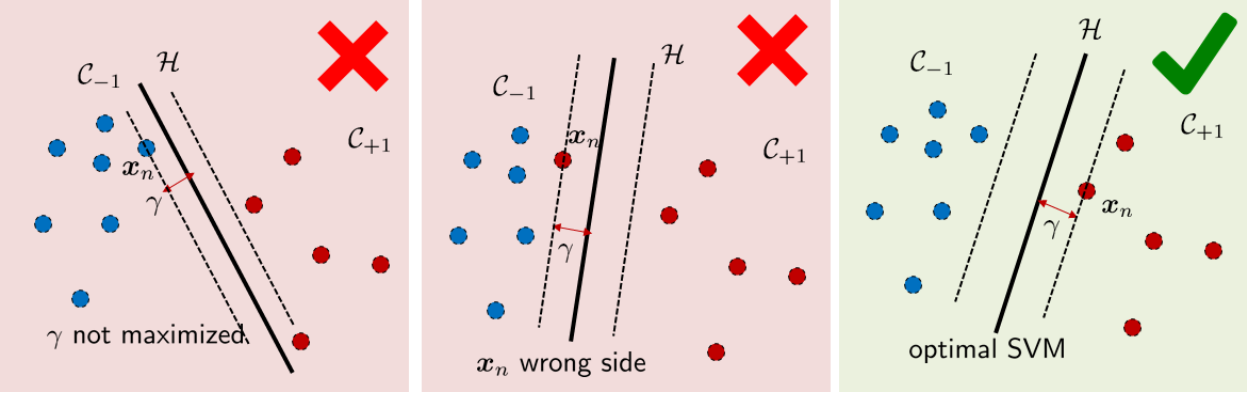


Figure 2.38: Optimal solution of SVM is determined when (1) the margin is maximized; (2) all the data points are correctly classified; (3) no data point can live inside the margin. [Left] The decision is not optimal because the margin is not maximized. [Middle] The decision is not optimal because $\boldsymbol{x}_n$ is misclassified. [Right] The decision is optimal.

Unfortunately, Equation (2.110) is a difficult optimization – by difficult we meant that it is not directly solvable using an off-the-shelf numerical solver. There are two reasons. First, γ is a function of $(\boldsymbol{w}, w_0)$. If we vary $(\boldsymbol{w}, w_0)$, the margin will change. Second, there is a term $\|\boldsymbol{w}\|_2$ appearing in the denominator of the constraint. This makes the constraint nonlinear, and is difficult to handle. In the followings we introduce two tricks to help us simplify the problem. The first trick is a re-normalization, something uniquely allowable in classification. The second trick is an optimization trick by flipping the reciprocal.

Let us define $\widetilde{\gamma} \overset{\text{def}}{=} |\boldsymbol{w}^T \boldsymbol{x}_n + w_0|$, which is the smallest unsigned distance without the denominator $\|\boldsymbol{w}\|_2$. The relationship between γ and $\widetilde{\gamma}$ is given by

$$\gamma \overset{\text{def}}{=} \frac{|\boldsymbol{w}^T \boldsymbol{x}_n + w_0|}{\|\boldsymbol{w}\|_2} = \frac{\widetilde{\gamma}}{\|\boldsymbol{w}\|_2}.$$

This gives us the first simplification of the optimization to

$$\begin{aligned} &\underset{\boldsymbol{w},w_0}{\text{maximize}} \quad \frac{\widetilde{\gamma}}{\|\boldsymbol{w}\|_2} \\ &\text{subject to} \quad y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0) \geq \widetilde{\gamma}, \quad j = 1, \ldots, n. \end{aligned} \tag{2.111}$$

Our next goal is to eliminate $\widetilde{\gamma}$. Consider a normalization of $\boldsymbol{\theta}$ so that the discriminant function $g_{\boldsymbol{\theta}}$ evaluated at $\boldsymbol{x}_n$ is $g_{\boldsymbol{\theta}}(\boldsymbol{x}_n) = 1$. To do so we pick a constant α such that $\alpha = 1/\widetilde{\gamma}$, and multiply $\boldsymbol{\theta}$ with α to yield $\alpha\boldsymbol{\theta}$. This gives us

$$\boldsymbol{w} \longleftarrow \alpha\boldsymbol{w} \quad \text{and} \quad w_0 \longleftarrow \alpha w_0.$$

Since $g_{\boldsymbol{\theta}}$ is linear, it holds that

$$g_{\alpha\boldsymbol{\theta}}(\boldsymbol{x}) = (\alpha\boldsymbol{w})^T\boldsymbol{x} + (\alpha w_0) = \alpha g_{\boldsymbol{\theta}}(\boldsymbol{x}), \tag{2.112}$$

which is just a scaled version of the original discriminant function. Therefore, the decision is not changed no matter what α we use.

Exercise. To convince yourself that the discriminant function is unchanged, consider the data

$$\boldsymbol{x}_1 = \begin{bmatrix} 0 \\ 2 \\ 2 \end{bmatrix}, \boldsymbol{x}_2 = \begin{bmatrix} 0 \\ 2 \\ 0 \end{bmatrix}, \boldsymbol{y} = \begin{bmatrix} -1 \\ -1 \\ +1 \end{bmatrix}, \boldsymbol{w} = \begin{bmatrix} 1.2 \\ -3.2 \end{bmatrix} \text{ and } w_0 = -0.5.$$

Compute $\alpha = 1/\widetilde{\gamma}$. Show that $(\boldsymbol{w}, w_0)$ satisfies $y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0) \geq \widetilde{\gamma}$, and that $(\alpha\boldsymbol{w}, \alpha w_0)$ satisfies $y_j((\alpha\boldsymbol{w})^T\boldsymbol{x}_j + \alpha w_0) \geq 1$.

Since we chose to force $g_{\boldsymbol{\theta}}(\boldsymbol{x}_n) = 1$, we can show that

$$\widetilde{\gamma} = |\boldsymbol{w}^T\boldsymbol{x}_n + w_0| = |g_{\boldsymbol{\theta}}(\boldsymbol{x}_n)| = 1.$$

Hence, the objective function and the constraint of the optimization Equation (2.111) become

$$\begin{aligned} &\underset{\boldsymbol{w},w_0}{\text{maximize}} \quad \frac{1}{\|\boldsymbol{w}\|_2} \\ &\text{subject to} \quad y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0) \geq 1, \quad j = 1, \ldots, n. \end{aligned} \tag{2.113}$$

This optimization problem is significantly more "civilized" than the original maximum-margin problem, because the variable-dependent quantity $\gamma(\boldsymbol{\theta})$ is eliminated. All is left is a reciprocal of an ℓ_2-norm, and a set of linear constraints.

The second (easy) trick we use is to replace $1/\|\boldsymbol{w}\|_2$ by its reciprocal $\|\boldsymbol{w}\|_2$, and flip the maximization to a minimization:

$$\underset{\boldsymbol{w},\boldsymbol{w}_0}{\text{maximize}} \quad \frac{1}{\|\boldsymbol{w}\|_2} \quad \equiv \quad \underset{\boldsymbol{w},w_0}{\text{minimize}} \quad \|\boldsymbol{w}\|_2.$$

This gives us

$$\begin{aligned} &\underset{\boldsymbol{w},w_0}{\text{minimize}} \;\; \|\boldsymbol{w}\|_2 \\ &\text{subject to} \;\; y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0) \geq 1, \quad j = 1,\ldots,n. \end{aligned} \tag{2.114}$$

However, the ℓ_2-norm $\|\boldsymbol{w}\|_2$ is not differentiable unless we take the square $\|\boldsymbol{x}\|_2^2$. In this context, the squaring is allowable because $(\cdot)^2$ is a monotonically increasing function (hence preserve the minimizer of the objective function.) Finally, by adding a scalar constant $1/2$ to the objective function we obtain the SVM formulation.

Theorem 14 (Defining SVM through Optimization)**.** *A **Support Vector Machine** defines a linear discriminant function $g_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0$ by solving the optimization*

$$\begin{aligned} &\underset{\boldsymbol{w},w_0}{\textit{minimize}} \quad \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ &\textit{subject to} \quad y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0) \geq 1, \quad j = 1,\ldots,n, \end{aligned} \tag{2.115}$$

which is a quadratic programming with linear constraints.

The resulting problem of is called a **quadratic programming** with linear inequality constraints. It is not difficult to prove that the problem is convex. As a result, we can use convex solvers such as `CVX` to solve the SVM optimization.

Convex Duality of SVM

Why is a support vector machine called "support vector" machine? We will answer this question by inspecting the convex duality of the optimization. By doing so we can show a few remarkable properties of SVM when handling high-dimensional data.

Consider Lagrangian of the constrained optimization in Equation (2.115):

$$\mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\lambda}) \stackrel{\text{def}}{=} \frac{1}{2}\|\boldsymbol{w}\|_2^2 - \sum_{j=1}^{n} \lambda_j \left[y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0) - 1\right],$$

where $\boldsymbol{\lambda} \geq 0$ is the Lagrange multiplier. The primal optimality criteria gives us

$$\nabla_{\boldsymbol{w}}\mathcal{L} = \boldsymbol{w} - \sum_{j=1}^{n} \lambda_j y_j \boldsymbol{x}_j = \boldsymbol{0}, \qquad \text{and} \qquad \nabla_{w_0}\mathcal{L} = \sum_{j=1}^{n} \lambda_j y_j = 0.$$

The first equality implies that the optimal weight $\boldsymbol{w}^*$ satisfies

$$\boldsymbol{w}^* = \sum_{j=1}^{n} \lambda_j y_j \boldsymbol{x}_j. \tag{2.116}$$

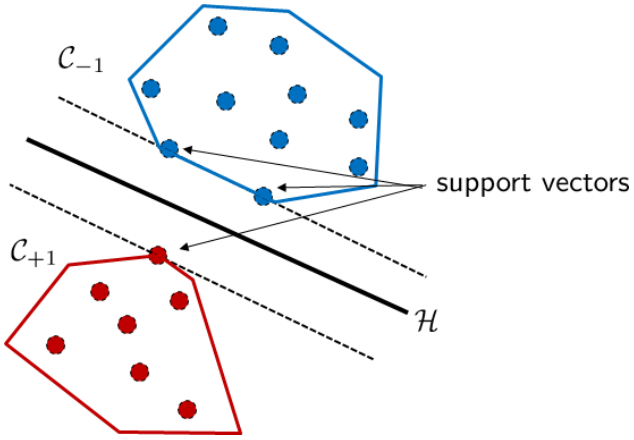


Figure 2.39: Support vector machine is defined using a few support vectors $\boldsymbol{x}_j \in \mathcal{V}$ so that $\boldsymbol{w}^* = \sum_{j\in\mathcal{V}} \lambda_j^* y_j \boldsymbol{x}_j$, where $\lambda_j^* > 0$ is the Lagrange multiplier.

This result is interesting in the sense that the optimal $\boldsymbol{w}^*$ is a linear combination of the vectors $\boldsymbol{x}_j$ in $\mathcal{D}$. But do we really need all $\boldsymbol{x}_j$ in $\mathcal{D}$ to define $\boldsymbol{w}^*$? Recall that in order to solve the optimization, besides finding a stationary point of the Lagrangian (via derivative), we also need to satisfy the complementarity condition at the optimal point. The complementarity condition states that

$$\lambda_j^* \left[y_j(\boldsymbol{w}^{*T}\boldsymbol{x}_j + w_0^*) - 1 \right] = 0, \quad \text{for} \quad j = 1, \ldots, n. \tag{2.117}$$

(Of course we have not yet determined w_0^*, but let us keep going.) Since $\lambda_j^* \geq 0$, this implies that if $\lambda_j^* > 0$ then $y_j(\boldsymbol{w}^{*T}\boldsymbol{x}_j + w_0^*) = 1$. Therefore, we can define the **support vector set**:

$$\mathcal{V} \stackrel{\text{def}}{=} \{j \mid \lambda_j^* > 0\}. \tag{2.118}$$

Because of the definition of $\mathcal{V}$, it holds that for any $j \in \mathcal{V}$, we must have $\boldsymbol{x}_j$ staying on the decision boundary. As a result, the optimal weight vector is

$$\boldsymbol{w}^* = \sum_{j\in\mathcal{V}} \lambda_j^* y_j \boldsymbol{x}_j, \tag{2.119}$$

which is a linear combination of a significantly smaller set of vectors $\boldsymbol{x}_j$. This explains the name "support vector machine", because the optimal weight $\boldsymbol{w}^*$ is defined by the linear combination of the vectors $\boldsymbol{x}_j$ where $j \in \mathcal{V}$.

How about the optimal bias w_0^*? If we look at the condition $\nabla_{w_0}\mathcal{L} = \sum_{j=1}^n \lambda_j y_j = 0$, we observe that since w_0 is a constant term, it is eliminated by the derivative. However, we can still determine w_0^* by noting that for any $\boldsymbol{x}^+$ in $\mathcal{C}_{+1}$ and $\boldsymbol{x}^-$ in $\mathcal{C}_{-1}$ we have

$$\boldsymbol{w}^T\boldsymbol{x}^+ + w_0 = +1, \quad \text{and} \quad \boldsymbol{w}^T\boldsymbol{x}^- + w_0 = -1.$$

Summing the two equations gives $\boldsymbol{w}^T(\boldsymbol{x}^+ + \boldsymbol{x}^-) + 2w_0 = 0$, which suggests that

$$w_0^* = -\frac{(\boldsymbol{x}^+ + \boldsymbol{x}^-)^T\boldsymbol{w}^*}{2}. \tag{2.120}$$

The existence of $\boldsymbol{x}^+$ and $\boldsymbol{x}^-$ is guaranteed because there must be at least one point on either side of the decision boundary.

Let us continue with our discussion of the convex dual of the SVM. Recall that the convex dual of a convex optimization is given by

$$\begin{aligned} &\underset{\boldsymbol{\lambda}}{\text{maximize}} \;\; \mathcal{L}(\boldsymbol{w}^*, w_0^*, \boldsymbol{\lambda}) \\ &\text{subject to} \;\; \boldsymbol{\lambda} \geq 0. \end{aligned} \tag{2.121}$$

Therefore, to determine the convex dual, we substitute the optimal $(\boldsymbol{w}^*, w_0^*)$ into the Lagrangian. Then, we can show that

$$\begin{aligned} \mathcal{L}(\boldsymbol{w}^*, w_0^*, \boldsymbol{\lambda}) &= \frac{1}{2}\left\|\sum_{j=1}^n \lambda_j y_j \boldsymbol{x}_j\right\|_2^2 - \sum_{j=1}^n \lambda_j \left[y_j\left(\left(\sum_{i=1}^n \lambda_i y_i \boldsymbol{x}_i\right)^T \boldsymbol{x}_j + w_0\right) - 1\right] \\ &= \frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n \lambda_i\lambda_j y_i y_j \boldsymbol{x}_i^T\boldsymbol{x}_j - \sum_{i=1}^n\sum_{j=1}^n \lambda_i\lambda_j y_i y_j \boldsymbol{x}_i^T\boldsymbol{x}_j - \underbrace{\left(\sum_{j=1}^n \lambda_j y_j\right)}_{=0} w_0 + \sum_{j=1}^n \lambda_j \\ &= -\frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n \lambda_i\lambda_j y_i y_j \boldsymbol{x}_i^T\boldsymbol{x}_j + \sum_{j=1}^n \lambda_j, \end{aligned}$$

where $\sum_{j=1}^n \lambda_j y_j = 0$ because $\nabla_{w_0}\mathcal{L} = 0$. As a result, the **dual problem** of SVM is determined as follows.

Theorem 15 (Dual of SVM)**.** *The convex dual problem of SVM is given by*

$$\begin{aligned} &\underset{\boldsymbol{\lambda}\geq 0}{\textit{maximize}} \;\; -\frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n \lambda_i\lambda_j y_i y_j \boldsymbol{x}_i^T\boldsymbol{x}_j + \sum_{j=1}^n \lambda_j \\ &\textit{subject to} \;\; \sum_{j=1}^n \lambda_j y_j = 0. \end{aligned} \tag{2.122}$$

The dual problem is again a quadratic programming with linear constraint, and so any standard convex solver is able to find the solution. However, the complexity does not seem to be advantageous. What we are able to get from the dual problem is the Lagrange multiplier $\boldsymbol{\lambda}^*$, which gives us critical information about the "support vectors".

How about the hypothesis function? It turns out the linear discriminant function $g_{\boldsymbol{\theta}}(\boldsymbol{x})$ is also defined by the support vectors:

$$g_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{w}^{*T}\boldsymbol{x} + w_0^* = \sum_{j\in\mathcal{V}} \lambda_j^* y_j \boldsymbol{x}_j^T\boldsymbol{x} + w_0^*. \tag{2.123}$$

Therefore, for any given data point $\boldsymbol{x}$, the testing procedure is simplified to computing the inner products $\boldsymbol{x}_j^T\boldsymbol{x}$. The hypothesis function is $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \text{sign}(g_{\boldsymbol{\theta}}(\boldsymbol{x}))$.

Soft-Margin SVM

The SVM defined in Equation (2.115) is called the **hard margin** SVM. Hard margin SVM is guaranteed to correctly separate the two classes if the classes are **linearly separable** (See the discussion on separating hyperplane theorem). However, if the two classes are not linearly separable, the constraint of the SVM will never be satisfied and so the optimization in Equation (2.115) is infeasible.

Soft margin SVM is a relaxation of Equation (2.115). The key idea is to introduce a **slack variable** $\xi_j \geq 0$ such that the constraint becomes

$$y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0) \geq 1 - \xi_j, \quad \text{and} \quad \xi_j \geq 0, \tag{2.124}$$

for $j = 1, \ldots, n$. Geometrically, $1 - \xi_j$ means that we allow samples to live in a margin up to $1 - \xi_j$. If for whatever reason we allow ξ_j grows beyond 1, then the input $\boldsymbol{x}_j$ will be misclassified, and soft margin SVM allows this to happen.

However, if ξ_j can be made arbitrarily large, then we will be tolerating too many misclassifications. Therefore, the magnitude of the margin ξ_j must be controlled. The way soft-margin SVM does is to introduce a penalty function $\rho(\boldsymbol{\xi})$ and redefine the cost function as

$$\begin{aligned} \underset{\boldsymbol{w},w_0,\boldsymbol{\xi}}{\text{minimize}} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\rho(\boldsymbol{\xi}) \\ \text{subject to} \quad & y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0) \geq 1 - \xi_j, \\ & \xi_j \geq 0, \quad \text{for} \quad j = 1, \ldots, n, \end{aligned} \tag{2.125}$$

where $C > 0$ is the penalty parameter. The job of the penalty function $C\rho(\boldsymbol{\xi})$ is to control the margin $\boldsymbol{\xi}$. Thus, $C\rho(\boldsymbol{\xi})$ should be large if the margin is large, and $C\rho(\boldsymbol{\xi})$ should be small when the margin is small. The strength of the penalty function is controlled by C. In the extreme case where $C = 0$, the margin can be made arbitrarily large because the constraint $y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0) \geq 1 - \xi_j$ and $\xi_j \geq 0$ have no control over $\boldsymbol{\xi}$.

The typical choice of the penalty function $\rho(\boldsymbol{\xi})$ is $\rho(\boldsymbol{\xi}) = \|\boldsymbol{\xi}\|^2$. Such penalty term controls the overall energy of $\boldsymbol{\xi}$, and is differentiable. The overall optimization is given by the definition below.

Definition 7 (Soft-Margin SVM $\|\boldsymbol{\xi}\|^2$)**.** *The* ***soft-margin SVM*** *for $\rho(\boldsymbol{\xi}) = \|\boldsymbol{\xi}\|^2$ is defined as the optimization*

$$\begin{aligned} \underset{\boldsymbol{w},w_0,\boldsymbol{\xi}}{\textit{minimize}} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{j=1}^{n}\xi_j^2 \\ \textit{subject to} \quad & y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0) \geq 1 - \xi_j, \\ & \xi_j \geq 0, \quad \textit{for} \quad j = 1, \ldots, n. \end{aligned} \tag{2.126}$$

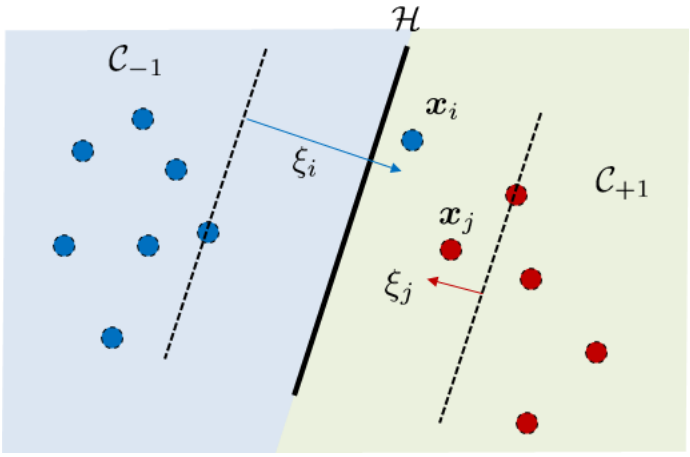


Figure 2.40: Soft-margin SVM. By introducing the slack variable ξ_j, we allow misclassifications. However, $\boldsymbol{\xi}_j$'s are controlled by the penalty term $C\rho(\boldsymbol{\xi})$ in Equation (2.125) so that $\boldsymbol{\xi}$ cannot be made arbitrarily large.

Note that this problem remains a quadratic programming with linear constraints. Therefore, the complexity of the problem stays the same as the original SVM problem. The exercise below shows that we can eliminate the non-negativity constraint.

Exercise 6.1. Consider the optimization problem shown in Equation (2.126). Show that the non-negativity constraint $\xi_j \geq 0$ is not necessary. That is, show that Equation (2.126) is equivalent to

$$\begin{aligned} \underset{\boldsymbol{w},w_0,\boldsymbol{\xi}}{\text{minimize}} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{j=1}^{n}\xi_j^2 \\ \text{subject to} \quad & y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0) \geq 1 - \xi_j, \quad \text{for} \quad j = 1,\ldots,n. \end{aligned}$$

Hint: Show that $\xi_j < 0$ for some j can never be a solution.

Besides $\rho(\boldsymbol{\xi}) = \|\boldsymbol{\xi}\|^2$, we can also consider penalty functions such as $\rho(\boldsymbol{\xi}) = \|\boldsymbol{\xi}\|_1$, then the objective function becomes $\frac{1}{2}\|\boldsymbol{w}\|^2 + C\|\boldsymbol{\xi}\|_1$. However, since $\boldsymbol{\xi} \geq 0$, the one-norm is simplified to $\|\boldsymbol{\xi}\|_1 = \sum_{j=1}^{n}\xi_j$. As a result, the optimization becomes the following.

Definition 8 (Soft-Margin SVM $\|\boldsymbol{\xi}\|_1$). *The **soft-margin SVM** for $\rho(\boldsymbol{\xi}) = \|\boldsymbol{\xi}\|_1$ is defined as the optimization*

$$\begin{aligned} \textit{minimize}_{\boldsymbol{w},w_0,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{j=1}^{n}\xi_j \\ \textit{subject to} \quad & y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0) \geq 1 - \xi_j, \\ & \xi_j \geq 0, \quad \textit{for} \quad j = 1,\ldots,n. \end{aligned} \tag{2.127}$$

Connection with Perceptron Algorithm

How is SVM related to the perceptron algorithm? Let us try to derive the training loss function of SVM and see if it provides any insight.

In the soft-margin SVM, note that the conditions $\xi_j \geq 0$ and $y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0) \geq 1 - \xi_j$ imply that

$$\xi_j \geq 0, \qquad \text{and} \qquad \xi_j \geq 1 - y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0).$$

The combination of the two can be compactly written as

$$\xi_j \geq \max\big\{0,\ \ 1 - y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0)\big\} \stackrel{\text{def}}{=} \big[1 - y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0)\big]_+.$$

Going back to the objective function, we note that the ℓ_1-norm soft SVM cost function is

$$J(\boldsymbol{\theta}, \boldsymbol{\xi}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{j=1}^n \xi_j,$$

which always increases as ξ_j increases. Therefore, at the optimal $\boldsymbol{\xi} = \boldsymbol{\xi}^*$, we must have ξ_j achieving its lower bound $\big[1 - y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0)\big]_+$:

$$J(\boldsymbol{\theta}, \boldsymbol{\xi}) \geq J(\boldsymbol{\theta}, \boldsymbol{\xi}^*) = \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{j=1}^n \big[1 - y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0)\big]_+.$$

Consequently, the optimization can be written as

$$\underset{\boldsymbol{w}, w_0}{\text{minimize}} \quad \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{j=1}^n \big[1 - y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0)\big]_+, \tag{2.128}$$

where we can call the first term $\|\boldsymbol{w}\|_2^2$ as the **regularization** term, as the second term $\sum_{j=1}^n \big[1 - y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0)\big]_+$, as the **data fidelity** term. By moving terms around, we can further write it as

$$\underset{\boldsymbol{w}, w_0}{\text{minimize}} \quad \sum_{j=1}^n \big[1 - y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0)\big]_+ + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2, \tag{2.129}$$

where $\lambda = 1/C$ is a fixed constant.

Theorem 16 (Soft-Margin SVM $\|\boldsymbol{\xi}\|_1$)**.** *The* ***soft-margin SVM*** *for $\rho(\boldsymbol{\xi}) = \|\boldsymbol{\xi}\|_1$ is equivalent to*

$$\textit{minimize}_{\boldsymbol{w}, w_0} \quad \sum_{j=1}^n \big[1 - y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0)\big]_+ + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2, \tag{2.130}$$

for some parameter $\lambda > 0$.

What is so special about Equation (2.130)? If we let $\lambda \to 0$, then the cost function in Equation (2.129) can be written as

$$\begin{aligned} J(\boldsymbol{\theta}) &= \sum_{j=1}^{n} \max\left\{0, \;\; 1 - y_j(\boldsymbol{w}^T \boldsymbol{x}_j + w_0)\right\} \\ &= \sum_{j=1}^{n} \max\Big\{1 - y_j g_{\boldsymbol{\theta}}(\boldsymbol{x}_j), \;\; 0\Big\}, \end{aligned}$$

which is reminiscent to the loss function of the perceptron algorithm, with the only difference that we have $\max(1-s, 0)$ versus $\max(-s, 0)$. The presence of an off-shift 1 gives the margin. But beyond that, SVM can be considered as a **regularized** perceptron algorithm if we incorporate $\lambda\|\boldsymbol{w}\|_2^2$ by limiting the magnitude of $\boldsymbol{w}$.

2.7 Nonlinear Transformation

Now that we have seen a number of linear classification methods, it would be natural to ask if we can generalize the concept to nonlinear classification methods. However, simply extending the discriminant functions to high order polynomials does not seem to be a plausible direction, as it does not transfer the linearity structure we want. The approach we adopt here is to use a transformation so that the classification is linear in the transformed space.

Consider a situation shown in Figure 2.41. The discriminant function $g(\boldsymbol{x})$ is defined as the circle

$$g(\boldsymbol{x}) = 0.5 - (x_1^2 + x_2^2). \tag{2.131}$$

For any $\boldsymbol{x}$ such that $g(\boldsymbol{x}) > 0$, we declare $\boldsymbol{x}$ belongs to class $+1$, and for any $\boldsymbol{x}$ such that $g(\boldsymbol{x}) < 0$ we declare $\boldsymbol{x}$ belongs to class -1. The hypothesis function is thus

$$h(\boldsymbol{x}) = \text{sign}(g(\boldsymbol{x})) = \text{sign}\left(0.5 - x_1^2 - x_2^2\right). \tag{2.132}$$

For the data we show in Figure 2.41, this hypothesis function guarantees that we can separate the data perfectly.

Let us try to rewrite $h(\boldsymbol{x})$. Define $z_0 = 1$, $z_1 = x_1^2$ and $z_2 = x_2^2$. Then, $h(\boldsymbol{x})$ becomes

$$\begin{aligned} h(\boldsymbol{x}) &= \text{sign}\left(0.5 - x_1^2 - x_2^2\right) \\ &= \text{sign}\left(\begin{bmatrix} 0.5 & -1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ x_1^2 \\ x_2^2 \end{bmatrix}\right) = \text{sign}\left(\begin{bmatrix} \widetilde{w}_0 & \widetilde{w}_1 & \widetilde{w}_2 \end{bmatrix} \begin{bmatrix} z_0 \\ z_1 \\ z_2 \end{bmatrix}\right). \end{aligned} \tag{2.133}$$

Therefore, we are able to rewrite a nonlinear discriminant function into a linear form by using a transformation

$$\boldsymbol{z} = \Phi(\boldsymbol{x}) = \begin{bmatrix} 1 \\ x_1^2 \\ x_2^2 \end{bmatrix}. \tag{2.134}$$

The example in Figure 2.41 shows two spaces. In the **input space** $\mathcal{X}$, the input vector $\boldsymbol{x} = [1,\ x_1,\ x_2]^T$ is classified using a nonlinear hypothesis $h(\boldsymbol{x}) = \text{sign}\,(0.5 - x_1^2 - x_2^2)$. In the transformed space, also called the **feature space** $\mathcal{Z}$, the transformed vector $\boldsymbol{z} = [1,\ x_1^2,\ x_2^2]^T$ is classified using a linear hypothesis. If we denote $\widetilde{\boldsymbol{w}} = [\widetilde{w}_0,\ \widetilde{w}_1,\ \widetilde{w}_2]^T$, then the hypothesis function is given by

$$h(\boldsymbol{x}) = \text{sign}\left(\widetilde{\boldsymbol{w}}^T \Phi(\boldsymbol{x})\right) = \widetilde{h}(\Phi(\boldsymbol{x})), \tag{2.135}$$

where h is a nonlinear hypothesis function, and $\widetilde{h}$ is a linear hypothesis function.

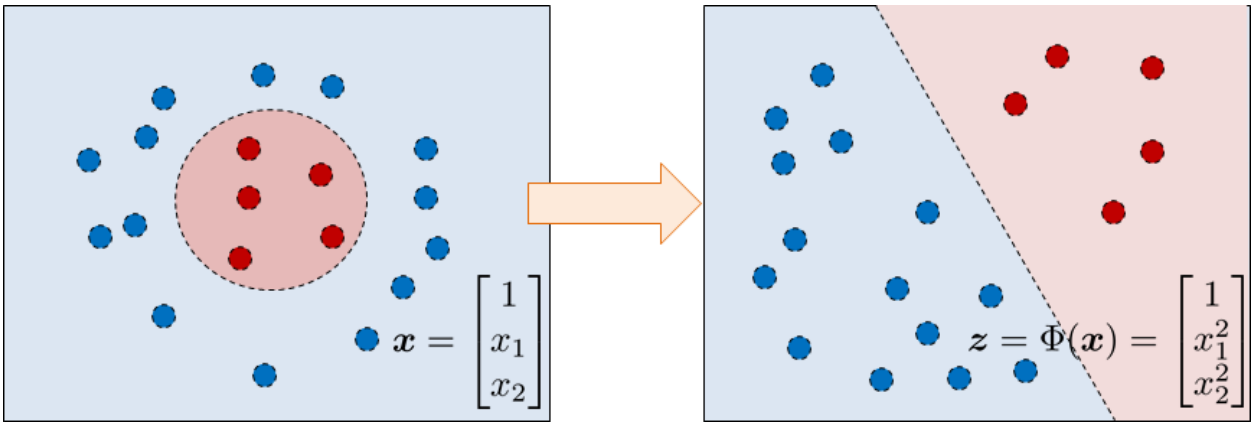


Figure 2.41: [Left] The input $\boldsymbol{x}$ is separable using a nonlinear hypothesis. [Right] By transforming $\boldsymbol{z} = \Phi(\boldsymbol{x})$, we are able to classify $\boldsymbol{z}$ using a linear classifier.

As the names suggest, the vector $\boldsymbol{x}$ is called the **input vector** whereas the transformed vector $\boldsymbol{z}$ is called the **feature vector**. The dimensionality of $\boldsymbol{z}$ does not need to be the same as $\boldsymbol{x}$, and typically it is smaller than that of $\boldsymbol{x}$.

Exercise 7.1. The transformation Φ does not need to be a one-to-one.

(i) Give an example Φ where a vector $\boldsymbol{z} \in \mathcal{Z}$ does not have a corresponding $\boldsymbol{x} \in \mathcal{X}$.

(ii) Give an example Φ where multiple vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2 \in \mathcal{X}$ are mapped to the same $\boldsymbol{z} \in \mathcal{Z}$.

So how do we construct the nonlinear transformation Φ? From a practical point of view there is really no one-fit-all recipe. However, in most of the cases we do not need to explicitly construct Φ but use something called a **kernel**. To see how this works let us use SVM as an example. Recall that the dual SVM is given by

$$\begin{aligned} \underset{\boldsymbol{\lambda} \geq 0}{\text{maximize}} \quad & -\frac{1}{2}\sum_{i=1}^n \sum_{j=1}^n \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j + \sum_{j=1}^n \lambda_j \\ \text{subject to} \quad & \sum_{j=1}^n \lambda_j y_j = 0. \end{aligned} \tag{2.136}$$

When we use nonlinear transformation, the objective function of the above dual SVM is modified to

$$\mathcal{L}(\boldsymbol{\lambda}) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\lambda_i\lambda_j y_i y_j \Phi(\boldsymbol{x}_i)^T\Phi(\boldsymbol{x}_j) + \sum_{j=1}^{n}\lambda_j \tag{2.137}$$

Therefore, effectively we are replacing $\boldsymbol{x}_i^T\boldsymbol{x}_j$ by $\Phi(\boldsymbol{x}_i)^T\Phi(\boldsymbol{x}_j)$. The inner product $\Phi(\boldsymbol{x}_i)^T\Phi(\boldsymbol{x}_j)$ is called a **kernel**

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \Phi(\boldsymbol{x}_i)^T\Phi(\boldsymbol{x}_j). \tag{2.138}$$

Our claim is that computing the kernel can be done without knowing Φ. Here is an example.

Consider a kernel defined by

$$K(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}^T\boldsymbol{v})^2, \qquad \boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^2.$$

If we write out $K(\boldsymbol{u}, \boldsymbol{v})$, we can show that

$$\begin{aligned} K(\boldsymbol{u}, \boldsymbol{v}) &= \left(\sum_{i=1}^{2} u_i v_i\right)^2 = \left(\sum_{i=1}^{2} u_i v_i\right)\left(\sum_{j=1}^{2} u_j v_j\right) \\ &= \sum_{i=1}^{2}\sum_{j=1}^{2}(u_i u_j)(v_i v_j) \\ &= \begin{bmatrix} u_1u_1 & u_1u_2 & u_2u_1 & u_2u_2 \end{bmatrix} \begin{bmatrix} v_1v_1 \\ v_1v_2 \\ v_2v_1 \\ v_2v_2 \end{bmatrix} = \Phi(\boldsymbol{u})^T\Phi(\boldsymbol{v}). \end{aligned}$$

Let us compare the computation required for computing $K(\boldsymbol{u}, \boldsymbol{v})$ versus $\Phi(\boldsymbol{u})^T\Phi(\boldsymbol{v})$. To compute $\Phi(\boldsymbol{u})^T\Phi(\boldsymbol{v})$, we need to perform pair-wise multiplications followed by an inner product. However, computing $K(\boldsymbol{u}, \boldsymbol{v})$ can be done by noting that $K(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}^T\boldsymbol{v})^2$, which only requires an inner product followed by a scalar square. Thus, as long as we have a good choice $K(\boldsymbol{u}, \boldsymbol{v})$, the effective computation can be done directly using $K(\boldsymbol{u}, \boldsymbol{v})$ without Φ.

Exercise 7.2. Derive the nonlinear transform $\Phi(\boldsymbol{u})$ for

$$K(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}^T\boldsymbol{v} + c)^2,$$

such that $K(\boldsymbol{u}, \boldsymbol{v}) = \Phi(\boldsymbol{u})^T\Phi(\boldsymbol{v})$. You may assume that $\boldsymbol{u} \in \mathbb{R}^2$ for simplicity.

Are there other choices of the kernel? One of the more popular choices of the kernel is the **radial basis kernel**, and in particular the Gaussian kernel:

$$K(\boldsymbol{u}, \boldsymbol{v}) = \exp\left\{-\frac{\|\boldsymbol{u} - \boldsymbol{v}\|^2}{2\sigma^2}\right\}. \tag{2.139}$$

If we use the radial basis kernel, then the SVM model parameter $\{\boldsymbol{w}, w_0\}$ can be determined by first maximizing the Lagrange dual (using convex solvers)

$$\mathcal{L}(\boldsymbol{\lambda}) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\lambda_i\lambda_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) + \sum_{j=1}^{n}\lambda_j, \tag{2.140}$$

and then define the hypothesis function as

$$h(\boldsymbol{x}) = \widetilde{h}(\Phi(\boldsymbol{x})) = \text{sign}\left(\sum_{j\in\mathcal{V}}\lambda_j^* y_j K(\boldsymbol{x}_j, \boldsymbol{x})\right).$$

The advantage of adopting the transformation, besides allowing nonlinear hypothesis function, is the dimensionality reduction introduced by the kernel K. While the input vector $\boldsymbol{x}$ could be high-dimensional, a carefully chosen kernel K can effectively bring the dimensionality down. Practically, kernel options are available in most of the SVM packages, and practitioners can choose appropriate kernels based on domain knowledge.

The last question we need to solve is that if we are given a kernel $K(\boldsymbol{u}, \boldsymbol{v})$, how do we know K is a valid kernel? Consider a set of data points $\{\boldsymbol{x}_j\}_{j=1}^n$, we can construct a $n \times n$ matrix $\boldsymbol{K}$ such that

$$[\boldsymbol{K}]_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \Phi(\boldsymbol{x}_i)^T\Phi(\boldsymbol{x}_j). \tag{2.141}$$

Let $\boldsymbol{z}$ be an arbitrary vector. Then,

$$\begin{aligned}\boldsymbol{z}^T\boldsymbol{K}\boldsymbol{z} &= \sum_{i=1}^{n}\sum_{j=1}^{n} z_i K_{ij} z_j = \sum_{i=1}^{n}\sum_{j=1}^{n} z_i\Phi(\boldsymbol{x}_i)^T\Phi(\boldsymbol{x}_j)z_j \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} z_i\left(\sum_{k=1}^{n}[\Phi(\boldsymbol{x}_i)]_k[\Phi(\boldsymbol{x}_j)]_k\right)z_j \overset{(a)}{=} \sum_{k=1}^{n}\left(\sum_{i=1}^{n}[\Phi(\boldsymbol{x}_i)]_k z_i\right)^2 \geq 0,\end{aligned}$$

where $[\Phi(\boldsymbol{x}_i)]_k$ denotes the k-th element of the vector $\Phi(\boldsymbol{x}_i)$. The result shows that if K is a valid kernel, then the corresponding matrix $\boldsymbol{K}$ must be symmetric positive semi-definite. More generally, this result is necessary and sufficient for K to be a valid kernel. That is, K is a valid kernel if and only if $\boldsymbol{K}$ is symmetric positive semi-definite. Kernels satisfying this condition is called **Mercer Kernel**.

Exercise 7.3. Verify the last equality (a) in the proof above.

Reading Suggestions

Introduction and Bayesian

The first two sections of this chapter are largely based on the following material. This cover the multi-class classification, Bayesian decision rule, and the estimation methods such as maximum-likelihood and maximum-a-posteriori.

- Learning from Data, by Abu-Mostafa, Magdon-Ismail and Lin, AMLBook, 2012. (Ch.1)
- Duda, Hart and Stork's *Pattern Classification*, Wiley-Interscience; 2 edition, 2000. (Ch.2 and Ch.3)
- UCSD ECE 271A Course Note `http://www.svcl.ucsd.edu/courses/ece271A/ece271A.htm`.
- Purdue ECE 645 Course Note `https://engineering.purdue.edu/ChanGroup/ECE645.html`. (Handwritten Note 1-4)
- Stanford CS 229 Course Note `http://cs229.stanford.edu/notes/cs229-notes2.pdf`. (Note 2)

Linear and Logistic Regression

The section on linear and logistic regression is based on these material:

- Elements of Statistical Learning, by Hastie, Tibshirani and Friedman, Springer, 2 edition, 2009. (Ch. 3)
- Purdue ECE 695 Course Note `https://engineering.purdue.edu/ChanGroup/ECE695.html`. (Lecture 2)
- Stanford CS 229 Course Note `http://cs229.stanford.edu/notes/cs229-notes1.pdf`. (Note 1)

Perceptron Algorithm

The section on Perceptron algorithm is based on

- Learning from Data, by Abu-Mostafa, Magdon-Ismail and Lin, AMLBook, 2012. (Ch.1)
- Duda, Hart and Stork's *Pattern Classification*, Wiley-Interscience; 2 edition, 2000. (Ch.4)

Support Vector Machine and Nonlinear Transform

The section on SVM and nonlinear transformation is based on

- Stanford CS 229 Course Note `http://cs229.stanford.edu/notes/cs229-notes3.pdf` (Note 3)

- Purdue ECE 695 Course Note `https://engineering.purdue.edu/ChanGroup/ECE695.html`. (Lecture 2)
- Learning from Data, by Abu-Mostafa, Magdon-Ismail and Lin, AMLBook, 2012. (Ch.3)

