Part I

Optimization Basics

CHAPTER 1

LAGRANGE MULTIPLIERS

1 Optimization with Equality Constraints

1.1 Geometry Illustration

Suppose we want to find the maximum or minimum of a function $f(x, y)$ subject to a constraint $g(x, y) = 0$. Geometrically, this means we are optimizing $f(x, y)$ on the curve or surface defined by $g(x, y) = 0$. Fig. 1.1 illustrates this concept. The surface $f(x, y)$ represents the function to be optimized, while the red curve shows the constraint $g(x, y) = 0$. The green region is the projection of the constraint onto the ground plane.

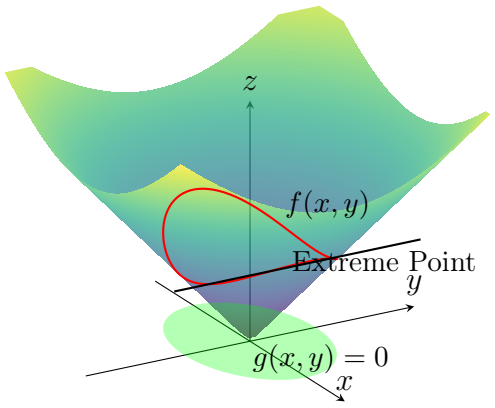


Figure 1.1: An optimization example: The green region represents the constraint $g(x, y) = 0$. Within the constrained region, when $f(x, y)$ reaches its extremum, the tangent to $f(x, y)$ becomes flat.

The method of Lagrange multipliers takes advantage of a special phenomenon: on the red curve, the tangent at the extremum is flat. This implies that at the extremum within the constrained region, we have the gradients of $f(x, y)$ and $g(x, y)$ are parallel. The solution is found by solving the following system of equations:

$$\frac{\partial f}{\partial x} + \lambda \frac{\partial g}{\partial x} = 0, \quad \frac{\partial f}{\partial y} + \lambda \frac{\partial g}{\partial y} = 0, \quad g(x, y) = 0.$$

This principle generalizes to higher-dimensional optimization problems with more constraints, where it remains valid. See the theorem for details.

The key to understanding lies in the fact that the tangent at the extremum of the constrained region is flat.

1.2 Mathematical Proof Under Equality Constraints

Theorem 1.1: Lagrange Multiplier

Assuming appropriate smoothness conditions, the minimum or maximum of $f(\mathbf{x})$ subject to the constraints

$$g_j(\mathbf{x}) = c_j, \quad 1 \leq j \leq p$$

that is not on the boundary of the region where $f(\mathbf{x})$ and $g_j(\mathbf{x})$ are defined can be found by introducing p new parameters $\lambda_1, \lambda_2, \ldots, \lambda_p$ and solving the system:

$$\frac{\partial}{\partial x_i}\left(f(\mathbf{x}) + \sum_{j=1}^{p} \lambda_j g_j(\mathbf{x})\right) = 0, \quad 1 \leq i \leq n$$

$$g_j(\mathbf{x}) = c_j, \quad 1 \leq j \leq p$$

Proof for Theorem.

By Taylor's Theorem:

$$f(\mathbf{x} + h\mathbf{y}) = f(\mathbf{x}) + h\mathbf{y} \cdot \nabla f(\mathbf{x}) + O(h^2), \tag{3.2}$$

where h is a scalar, $O(h^2)$ denotes terms that are bounded by h^2, and $\mathbf{x} \cdot \mathbf{y}$ is the dot product. Thus, equation (3.1) gives the vector direction in which $f(\mathbf{x})$ changes the most per unit change in $\mathbf{x}$, where the unit change is measured in terms of the length of the vector $\mathbf{x}$.

In particular, if $\mathbf{y} = \nabla f(\mathbf{x}_0) \neq 0$, then:

$$f(\mathbf{x}_0 - h\mathbf{y}) < f(\mathbf{x}_0) < f(\mathbf{x}_0 + h\mathbf{y}).$$

for sufficiently small values of h.

Now consider the problem of finding

$$\max f(\mathbf{x}) \quad \text{subject to} \quad g(\mathbf{x}) = c$$

for one constraint. If $\mathbf{x} = \mathbf{x}_1(t)$ is a path on the surface defined by $g(\mathbf{x}) = c$, then by the chain rule:

$$\frac{d}{dt} g\left(\mathbf{x}_1(0)\right) = \frac{d}{dt}\mathbf{x}_1(0) \cdot \nabla g\left(\mathbf{x}_1(0)\right) = 0.$$

This implies that $\nabla g\left(\mathbf{x}_1(0)\right)$ is orthogonal to the tangent vector $\frac{d}{dt}\mathbf{x}_1(0)$ for any path $\mathbf{x}_1(t)$ on the surface defined by $g(\mathbf{x}) = c$.

Conversely, if $\mathbf{x}_0$ is any point on the surface $g(\mathbf{x}) = c$ and $\mathbf{y}$ is any vector such that $\mathbf{y} \cdot \nabla g(\mathbf{x}_0) = 0$, then it follows from the Implicit Function Theorem that there exists a path $\mathbf{x}_1(t)$ on the surface $g(\mathbf{x}) = c$ such that $\mathbf{x}_1(0) = \mathbf{x}_0$ and $\frac{d}{dt}\mathbf{x}_1(0) = \mathbf{y}$. This result implies that the gradient vector $\nabla g(\mathbf{x}_0)$ is always orthogonal to the surface defined by $g(\mathbf{x}) = c$ at $\mathbf{x}_0$.

Now let $\mathbf{x}_0$ be a solution. We claim that:

$$\nabla f(\mathbf{x}_0) = \lambda \nabla g(\mathbf{x}_0),$$

for some scalar λ. First, we can always write:

$$\nabla f(\mathbf{x}_0) = c\nabla g(\mathbf{x}_0) + \mathbf{y},$$

where $\mathbf{y} \cdot \nabla g(\mathbf{x}_0) = 0$ (It's just expressing a vector as the sum of two perpendicular vectors). If $\mathbf{x}(t)$ is a path on the surface such that $\mathbf{x}(0) = \mathbf{x}_0$ and $\frac{d}{dt}\mathbf{x}(0) \cdot \nabla f(\mathbf{x}_0) \neq 0$, it follows with $\mathbf{y} = \frac{d}{dt}\mathbf{x}(0)$ that there exist values of $f(\mathbf{x})$ for $\mathbf{x} = \mathbf{x}(t)$ on the surface that are both larger and smaller than $f(\mathbf{x}_0)$.

Thus, if $\mathbf{x}_0$ is a maximum or minimum of $f(\mathbf{x})$ on the surface and:

$$\nabla f(\mathbf{x}_0) = c\nabla g(\mathbf{x}_0) + \mathbf{y}, \quad \text{where } \mathbf{y} \cdot \nabla g(\mathbf{x}_0) = 0,$$

then:

$$\mathbf{y} \cdot \nabla f(\mathbf{x}_0) = \mathbf{y} \cdot \nabla g(\mathbf{x}_0) + \mathbf{y} \cdot \mathbf{y} = \mathbf{y} \cdot \mathbf{y} = 0,$$

which implies $\mathbf{y} = 0$ (If $\mathbf{y} \neq 0$, there would be a direction (determined by $\mathbf{y}$) along which $f(\mathbf{x})$ could increase or decrease on the surface. This would contradict the assumption that $\mathbf{x}_0$ is a maximum or minimum.). Therefore:

$$\nabla f(\mathbf{x}_0) = c\nabla g(\mathbf{x}_0),$$

which completes the proof of Lagrange's Theorem for one constraint ($p = 1$).

Next, suppose that we want to solve:

$$\max f(\mathbf{x}) \quad \text{subject to} \quad g_1(\mathbf{x}) = c_1, \ldots, g_p(\mathbf{x}) = c_p$$

for p constraints. Let $\mathbf{x}_0$ be a solution. Recall that each vector $\nabla g_j(\mathbf{x}_0)$ is orthogonal to the surface $g_j(\mathbf{x}) = c_j$ at $\mathbf{x}_0$.

This means that $\nabla g_j(\mathbf{x}_0)$ represents the direction of the steepest change of $g_j(\mathbf{x})$, and any movement along the surface $g_j(\mathbf{x}) = c_j$ is tangent to the surface and thus orthogonal to $\nabla g_j(\mathbf{x}_0)$.

Let L be the linear space defined as:

$$L = \text{span}\{\nabla g_j(\mathbf{x}_0) : 1 \leq j \leq p\}.$$

This space L consists of all linear combinations of the gradients $\nabla g_j(\mathbf{x}_0)$. Geometrically, it captures all directions that are normal to the surfaces defined by the constraints $g_j(\mathbf{x}) = c_j$ at $\mathbf{x}_0$.

We claim that $\nabla f(\mathbf{x}_0) \in L$. In other words, the gradient of $f(\mathbf{x})$ at $\mathbf{x}_0$ lies entirely in the span of the gradients of the constraint functions. This is because if $\nabla f(\mathbf{x}_0)$ had

any component outside L, it would imply the existence of a direction along which $f(\mathbf{x})$ could increase or decrease while still satisfying all constraints. This would contradict the assumption that $\mathbf{x}_0$ is an optimum.

Thus, $\nabla f(\mathbf{x}_0)$ can be written as:

$$\nabla f(\mathbf{x}_0) = \sum_{j=1}^{p} \lambda_j \nabla g_j(\mathbf{x}_0),$$

where λ_j are scalars (Lagrange multipliers) that measure the relative influence of each constraint on the optimal solution. ■

2 General Cases with Inequality Constraints

Suppose we have a more general optimization problem with variables $\mathbf{x}$, p equality constraints, and l inequality constraints:

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g_i(\mathbf{x}) = 0, \quad i = 1, \ldots, p \\ & h_j(\mathbf{x}) \leq 0, \quad j = 1, \ldots, l \end{aligned} \tag{1.1}$$

Next, we relax the problem by removing the constraints and introducing penalty terms for each constraint. This leads to the following relaxed problem:

$$\min \quad f(\mathbf{x}) + \sum_{i=1}^{p} \lambda_i g_i(\mathbf{x}) + \sum_{j=1}^{l} \mu_j h_j(\mathbf{x}) \tag{1.2}$$

We now introduce a multiplier λ_i for each equality constraint and μ_j for each inequality constraint. These are called the KKT (Karush-Kuhn-Tucker) multipliers. If there are only equality constraints, we are left with the Lagrange multipliers (and the same results as before).

Much of the reasoning from the method of Lagrange multipliers still applies here. We can define the objective function as the Lagrangian $\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$, and find that the necessary conditions for optimality include $\nabla_{\mathbf{x}} \mathcal{L} = 0$ (stability) and $\nabla_{\boldsymbol{\lambda}} \mathcal{L} = 0$ (feasibility of the inequality constraints). However, it is not necessary to also have $\nabla_{\boldsymbol{\mu}} \mathcal{L} = 0$, since this is equivalent to enforcing $h_j(\mathbf{x}) = 0$, which provides only part of the feasible set. As a result, we cannot simply state that $\nabla_{\mathbf{x},\boldsymbol{\lambda},\boldsymbol{\mu}} \mathcal{L} = 0$.

Note that, in the previous section, we often switched between the forms $\nabla f = \lambda \nabla g$ and $\nabla f + \lambda \nabla g = 0$, since the absence of constraints on λ made them equivalent. In this case, however, the sign of μ matters, so we need to be more careful about the signs of the terms.

It is actually quite easy to understand. Imagine that if the extremum occurs when $h_j(\mathbf{x}) = 0$, it is equivalent to the Lagrange operator defined by the equality constraint. If it occurs when $h_j(\mathbf{x}) < 0$, then there is no need to consider it, so $\boldsymbol{\mu}_j$ is zero.

2.1 Simple Case Derivation

We start by considering the simplified two-dimensional system:

$$\begin{aligned} \min \quad & f(x,y) \\ \text{s.t.} \quad & h(x,y) \leq 0 \end{aligned}$$

which involves only inequality constraints. Again, we can imagine the level sets of f in the plane. The constraint $h(x,y) \leq 0$ defines a region within the plane. There are two possibilities for the optimal solution: either it lies completely within the feasible region, in which case $h(x,y) < 0$ and f is at an extremum, or it lies on the boundary, where $h(x,y) = 0$. in which case $h(x,y) = 0$ and the boundary of $h(x,y)$ is tangent to a contour of f.

If the optimum occurs where $h(x,y) < 0$, then the inequality constraint has no effect on the problem and can be ignored. Otherwise, $h(x,y) = 0$. In either case, the constraint $\mu h(x,y) = 0$ must be satisfied for some μ. Either $h(x,y) = 0$, in which case the value of μ does not matter, or $h(x,y) < 0$, in which case we must have $\mu = 0$. This is complementary slackness.

We do need to consider the sign of μ, however. Our goal is to be able to use this variable as part of the objective of the relaxed problem:

$$\min \quad f(x,y) + \mu h(x,y) \tag{1.3}$$

Recall that the gradient of a function is the direction of steepest ascent. Since we are dealing with the case in which the minimum of f lies outside the region defined by h, the gradient of f should point into the region. Otherwise, a local minimum would occur inside the region or on the other side of it, and in either case (x,y) would not be the minimum of the feasible set.

As for h, since the boundary of the region is defined by $h(x,y) = 0$ and the interior is defined by $h(x,y) < 0$, h must decrease as we move towards the interior, meaning that the gradient of h should point out of the region. These two directions are opposite, so $[\nabla f(x,y)]_k$ and $[\nabla h(x,y)]_k$ must have opposite signs for $k = 1, 2$, meaning that μ is always positive.

2.2 General KKT Conditions

Given a general constrained nonlinear optimization problem:

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g_i(\mathbf{x}) = 0, \quad i = 1, \ldots, p, \\ & h_j(\mathbf{x}) \leq 0, \quad j = 1, \ldots, \ell, \end{aligned} \tag{1.4}$$

The KKT conditions are a set of necessary conditions that any optimal solution $\mathbf{x} = (x_1, \ldots, x_n)$ must satisfy. Specifically, there must exist multipliers $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_p)$ and $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_\ell)$ such that the following hold.

- **Stationarity:** If minimizing,

$$\nabla_{\mathbf{x}} f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i \nabla_{\mathbf{x}} g_i(\mathbf{x}) + \sum_{j=1}^{\ell} \mu_j \nabla_{\mathbf{x}} h_j(\mathbf{x}) = 0$$

- **Primal Feasibility:**

$$g_i(\mathbf{x}) = 0, \quad i = 1, \ldots, p$$
$$h_j(\mathbf{x}) \leq 0, \quad j = 1, \ldots, \ell$$

- **Dual Feasibility:**

$$\mu_j \geq 0, \quad j = 1, \ldots, \ell$$

- **Complementary Slackness:**

$$\mu_j h_j(\mathbf{x}) = 0, \quad j = 1, \ldots, \ell$$

Theorem 2.1: Necessity of KKT Conditions

KKT condition is a necessary condition for optimization problems

Proof for Theorem.

Now we prove the necessity of KKT conditions. Clearly, if $\mathbf{x}^*$ is an optimal solution, then it must be a local minimum. Consider the following set:

$$\tilde{\Omega} \triangleq \{\mathbf{x} \mid g_i(\mathbf{x}) = 0 \text{ for all } i,\ h_j(\mathbf{x}) = 0 \text{ for all } j \in J(\mathbf{x}^*),\ h_j(\mathbf{x}) < 0 \text{ for all } j \notin J(\mathbf{x}^*)\}.$$

It is a subset of the feasible set Ω, and thus $\mathbf{x}^*$ must be a local minimum on $\tilde{\Omega}$. If we assume that h_j is continuous for all j, then there exists $\epsilon > 0$ such that for all $\mathbf{x} \in \mathcal{B}(\mathbf{x}^*, \epsilon)$, $h_j(\mathbf{x}) < 0$ for all j. So locally, we have:

$$\tilde{\Omega} \cap \mathcal{B}(\mathbf{x}^*, \epsilon) = \{\mathbf{x} \mid g_i(\mathbf{x}) = 0 \text{ for all } i,\ h_j(\mathbf{x}) = 0 \text{ for all } j \in J(\mathbf{x}^*)\}.$$

Hence, $\mathbf{x}^*$ should be a local minimum on this set. There are only equality constraints, so the Lagrange condition applies. Therefore, there exist KKT multipliers $\lambda_1^*, \ldots, \lambda_m^*$ and $\mu_1^*, \ldots, \mu_\ell^*$ such that:

$$\nabla f(\mathbf{x}^*) + \sum_{i=1}^{p} \lambda_i^* \nabla g_i(\mathbf{x}^*) + \sum_{j=1}^{\ell} \mu_j^* \nabla h_j(\mathbf{x}^*) = 0,$$

and

$$\mu_j^* h_j(\mathbf{x}^*) = 0 \quad \text{for all } j = 1, \ldots, \ell.$$

The remaining part is to show that $\mu_j^* \geq 0$. We prove this by contradiction. Assume there exists an active $k \in J(\mathbf{x}^*)$ and $\mu_k^* < 0$. Then, we consider the set containing all other active constraints:

$$\tilde{\Omega} = \{\mathbf{x} \mid g_i(\mathbf{x}) = 0,\ i = 1, \ldots, p;\ h_j(\mathbf{x}) = 0,\ j \neq k,\ j \in J(\mathbf{x}^*)\}.$$

If $\mathbf{x}^*$ is regular, $T = T_{\mathbf{x}^*}\tilde{\Omega}$ is a linear space, where:

$$T = \ker \begin{pmatrix} \nabla g_i & 1 \leq i \leq p \\ \nabla h_j & k \neq j \in J(\mathbf{x}^*) \end{pmatrix}.$$

By regularity of $\mathbf{x}^*$, $\nabla h_k(\mathbf{x}^*) \notin \text{span}\{\nabla g_i(\mathbf{x}^*), \nabla h_j(\mathbf{x}^*)\}$ where $i = 1, 2, \ldots, p$ and $j \in J(\mathbf{x}^*)$, $j \neq k$. So there exists $\mathbf{v} \in T$ such that $\nabla h_k(\mathbf{x}^*)^\top \mathbf{v} \neq 0$, otherwise the above fact does not hold. Without loss of generality, assume $\nabla h_k(\mathbf{x}^*)^\top \mathbf{v} < 0$.

Now we consider the Lagrange condition:

$$\nabla f(\mathbf{x}^*) + \sum_{i=1}^{p} \lambda_i^* \nabla g_i(\mathbf{x}^*) + \sum_{j \in J(\mathbf{x}^*)} \mu_j^* \nabla h_j(\mathbf{x}^*) = 0.$$

Multiplying by $\mathbf{v}$, we have:

$$\left(\nabla f(\mathbf{x}^*) + \sum_{i=1}^{p} \lambda_i^* \nabla g_i(\mathbf{x}^*) + \sum_{j \in J(\mathbf{x}^*)} \mu_j^* \nabla h_j(\mathbf{x}^*) \right)^\top \mathbf{v} = 0.$$

Note that $\nabla g_i(\mathbf{x}^*)^\top \mathbf{v} = 0$ and $\nabla h_k(\mathbf{x}^*)^\top \mathbf{v} = 0$ if $j \neq k$. Then,

$$\nabla f(\mathbf{x}^*)^\top \mathbf{v} + \mu_k^* \nabla h_k(\mathbf{x}^*)^\top \mathbf{v} = 0 \implies \nabla f(\mathbf{x}^*)^\top \mathbf{v} < 0.$$

Since $\mathbf{v} \in T$, then there exists $\gamma : (-\epsilon, \epsilon) \to \tilde{\Omega}$ such that $\gamma(0) = \mathbf{x}^*$ and $\gamma'(0) = \mathbf{v}$. Then,

$$\begin{cases} f'(\gamma(t))\big|_{t=0} = \nabla f(\gamma(0))^\top \gamma'(0) = \nabla f(\mathbf{x}^*)^\top \mathbf{v} < 0 \\ h_k'(\gamma(t))\big|_{t=0} = \nabla h_k(\gamma(0))^\top \gamma'(0) = \nabla h_k(\mathbf{x}^*)^\top \mathbf{v} < 0 \end{cases}$$

which leads to

$$\begin{cases} \exists \epsilon_0 > 0,\ \forall 0 < \epsilon \leq \epsilon_0,\ f(\gamma(\epsilon)) < f(\gamma(0)) = f(\mathbf{x}^*) \\ \exists \delta_0 > 0,\ \forall 0 < \delta \leq \delta_0,\ h_k(\gamma(\delta)) < h_k(\gamma(0)) = h_k(\mathbf{x}^*) \\ \exists \xi_0 > 0,\ \forall 0 < \xi \leq \xi_0,\ h_j(\gamma(\xi)) \leq 0 \,\text{for any } j \notin J(\mathbf{x}^*) \end{cases}$$

Now we obtain that for $\mathbf{x}' \in \gamma(\min\{\epsilon_0, \delta_0, \xi_0\})$.

$$\begin{cases} h_k(\mathbf{x}') < h_k(\mathbf{x}^*) \leq 0, \\ f(\mathbf{x}') < f(\mathbf{x}^*), \\ \mathbf{x}' \in \tilde{\Omega}, \\ h_j(\mathbf{x}') \leq 0 \quad \text{for any } j \notin J(\mathbf{x}^*), \end{cases}$$

which contradicts the fact that $\mathbf{x}^*$ is optimal. Thus, we conclude $\mu_j^* \geq 0$. ∎

Theorem 2.2: Necessity of KKT Conditions

KKT condition is a sufficient condition for convex optimization problems

Proof for Theorem.

It suffices to show that for any feasible $\mathbf{x}$,

$$\nabla f(\mathbf{x}^*)^\top (\mathbf{x} - \mathbf{x}^*) \geq 0$$

since

$$f(\mathbf{x}) \geq f(\mathbf{x}^*) + \nabla f(\mathbf{x}^*)^\top (\mathbf{x} - \mathbf{x}^*).$$

By the KKT condition,

$$\nabla f(\mathbf{x}^*) = \sum_{i=1}^{p} -\lambda_i^* \nabla g_i(\mathbf{x}^*) + \sum_{j=1}^{\ell} -\mu_j^* \nabla h_j(\mathbf{x}^*).$$

We claim that

$$\nabla g_i(\mathbf{x}^*)^\top (\mathbf{x} - \mathbf{x}^*) = 0 \quad \text{for all } i$$

and

$$\nabla h_j(\mathbf{x}^*)^\top (\mathbf{x} - \mathbf{x}^*) \leq 0 \quad \text{for all } j.$$

Note that:

$$\begin{cases} \forall i, g_i \text{ is affine, so } g_i(\mathbf{x}) = g_i(\mathbf{x}^*) = 0 \implies \nabla g_i(\mathbf{x}^*)^\top (\mathbf{x} - \mathbf{x}^*) = 0; \\ \forall j \notin J(\mathbf{x}^*), \mu_j^* = 0; \\ \forall j \in J(\mathbf{x}^*), h_j(\mathbf{x}^*) = 0,\ h_j(\mathbf{x}) \leq 0 \implies \nabla h_j(\mathbf{x}^*)^\top (\mathbf{x} - \mathbf{x}^*) \leq h_j(\mathbf{x}) - h_j(\mathbf{x}^*) \leq 0. \end{cases}$$

Hence, we conclude that

$$\nabla f(\mathbf{x}^*)^\top (\mathbf{x} - \mathbf{x}^*) \geq 0.$$

■

2.3 Lagrangian Dual Problem

In many cases, the original problem in Equation (1.2) is challenging:

1. **Too many constraints**: Obviously, the more constraints there are, the harder the problem becomes to solve. In the original problem, there are a total of $k+1$ constraints, which makes it quite complicated.
2. **Unclear convexity of the original problem**: We cannot directly apply convex optimization methods to solve the original problem.

We can construct a generalized Lagrange function $\mathcal{L}$ for the original problem:

$$\begin{aligned} \mathcal{L} &: \mathbb{R}^n \times \mathbb{R}^p \times \mathbb{R}^\ell \to \mathbb{R} \\ \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) &= f(\mathbf{x}) + \sum_{i=1}^{p} \lambda_i g_i(\mathbf{x}) + \sum_{j=1}^{\ell} \mu_j h_j(\mathbf{x}), \end{aligned} \tag{1.5}$$

where $\mathbf{x} \in \mathbb{R}^n$, $\boldsymbol{\lambda} \in \mathbb{R}^p$, and $\boldsymbol{\mu} \in \mathbb{R}^\ell$. Here, $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are referred to as the Lagrange multiplier vectors. With this generalized Lagrange function, we can now proceed to define additional auxiliary functions g as needed.

We can define a Lagrange dual function $g(\boldsymbol{\lambda}, \boldsymbol{\mu})$ based on $\mathcal{L}$:

$$\begin{aligned} \mathcal{G}(\boldsymbol{\lambda}, \boldsymbol{\mu}) &= \inf_{\mathbf{x} \in \mathcal{D}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \\ \mathcal{G}(\boldsymbol{\lambda}, \boldsymbol{\mu}) &= \inf_{\mathbf{x} \in \mathcal{D}} \left(f(\mathbf{x}) + \sum_{i=1}^{p} \lambda_i g_i(\mathbf{x}) + \sum_{j=1}^{\ell} \mu_j h_j(\mathbf{x}) \right), \\ \lambda_i &\geq 0, \quad \forall i. \end{aligned} \tag{1.6}$$

Theorem 2.3: Concavity of Lagrange Dual Function

The function in Equation (1.6) is concave

Proof for Theorem.

We first have:

$$\mathcal{G}(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\mathbf{x} \in \mathcal{D}} \{\mathcal{L}(\mathbf{x}_1, \boldsymbol{\lambda}, \boldsymbol{\mu}),\, \mathcal{L}(\mathbf{x}_2, \boldsymbol{\lambda}, \boldsymbol{\mu}),\, \ldots,\, \mathcal{L}(\mathbf{x}_n, \boldsymbol{\lambda}, \boldsymbol{\mu})\},\, n \to \infty$$

For simplicity:

$$\gamma = (\boldsymbol{\lambda}, \boldsymbol{\mu}), \quad g(\gamma) = g(\boldsymbol{\lambda}, \boldsymbol{\mu})$$

The proof is as follows:

$$\begin{aligned} &\mathcal{G}(\theta\gamma_1 + (1-\theta)\gamma_2) \\ &= \inf\{\mathcal{L}(\mathbf{x}_1, \theta\gamma_1 + (1-\theta)\gamma_2),\, \mathcal{L}(\mathbf{x}_2, \theta\gamma_1 + (1-\theta)\gamma_2),\, \ldots,\, \mathcal{L}(\mathbf{x}_n, \theta\gamma_1 + (1-\theta)\gamma_2)\} \\ &= \inf\{\theta\mathcal{L}(\mathbf{x}_1, \gamma_1) + (1-\theta)\mathcal{L}(\mathbf{x}_1, \gamma_2),\, \ldots,\, \theta\mathcal{L}(\mathbf{x}_n, \gamma_1) + (1-\theta)\mathcal{L}(\mathbf{x}_n, \gamma_2)\} \\ &\geq \theta \inf\{\mathcal{L}(\mathbf{x}_1, \gamma_1), \mathcal{L}(\mathbf{x}_2, \gamma_1),\, \ldots,\, \mathcal{L}(\mathbf{x}_n, \gamma_1)\} \\ &+(1-\theta) \inf\{\mathcal{L}(\mathbf{x}_1, \gamma_2), \mathcal{L}(\mathbf{x}_2, \gamma_2),\, \ldots,\, \mathcal{L}(\mathbf{x}_n, \gamma_2)\} \\ &= \theta\mathcal{G}(\gamma_1) + (1-\theta)\mathcal{G}(\gamma_2). \end{aligned}$$

- **Second Line:** Notice that in $\mathcal{L}(\mathbf{x}_i, \gamma)$, the value of $\mathbf{x}_i$ is already fixed. Hence, $f(\mathbf{x})$, $g_i(\mathbf{x})$, and $h_i(\mathbf{x})$ are constants. We denote them as r, $\mathbf{p}$, and $\mathbf{q}$, respectively. Thus:

  $$\mathcal{L}(\mathbf{x}_i, \gamma) = \sum_{i=1}^{p} \lambda_i p_i + \sum_{j=1}^{\ell} \mu_j q_j + r.$$

  Clearly, $\mathcal{L}$ is an affine function, and we know that affine functions are both convex and concave. Therefore, the following holds:

  $$\mathcal{L}(\mathbf{x}_n, \theta\gamma_1 + (1-\theta)\gamma_2) \geq \theta\mathcal{L}(\mathbf{x}_n, \gamma_1) + (1-\theta)\mathcal{L}(\mathbf{x}_n, \gamma_2).$$

  After reorganizing, we arrive at the second line of the proof.

- **Third Line:** From the second line to the third line, we applied a simple mathematical principle:

  $$\min\{a_i + b_i\} \geq \min\{a\} + \min\{b\}.$$

  This property ensures that splitting terms and taking their individual infimum does not decrease the overall value. Thus, the inequality from the second line transitions to the third line smoothly.

■

Theorem 2.4: Lower bound property

If $\boldsymbol{\lambda} \geq 0$, then $\mathcal{G}(\boldsymbol{\lambda}, \boldsymbol{\nu}) \leq f(\mathbf{x}^*)$.

Proof for Theorem.

Suppose $\tilde{\mathbf{x}}$ is feasible and $\boldsymbol{\lambda} \geq 0$. Then,

$$f(\tilde{\mathbf{x}}) \geq \mathcal{L}(\tilde{\mathbf{x}}, \boldsymbol{\lambda}, \boldsymbol{\nu}) \geq \inf_{\mathbf{x} \in \mathcal{D}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = \mathcal{G}(\boldsymbol{\lambda}, \boldsymbol{\nu}).$$

This is because for feasible $\tilde{\mathbf{x}}$, we know:

$$g_i(\tilde{\mathbf{x}}) \leq 0, \quad h_i(\tilde{\mathbf{x}}) = 0.$$

Thus:

$$\lambda_i g_i(\tilde{\mathbf{x}}) \leq 0, \quad \mu_i h_i(\tilde{\mathbf{x}}) = 0, \quad f(\tilde{\mathbf{x}}) \geq \mathcal{L}(\tilde{\mathbf{x}}, \boldsymbol{\lambda}, \boldsymbol{\nu})$$

Now choose the minimizer of $f(\tilde{\mathbf{x}})$ over all feasible $\tilde{\mathbf{x}}$ to get:

$$f(\mathbf{x}^*) \geq \mathcal{G}(\boldsymbol{\lambda}, \boldsymbol{\nu}).$$

1. Our goal is to find the optimal solution $f(\mathbf{x}^*)$.
2. Sometimes, $f(\mathbf{x}^*)$ cannot be solved directly. In such cases, we hope to find a value that approximates $f(\mathbf{x}^*)$ as closely as possible.
3. **Since we already know that $\mathcal{G}$ provides a lower bound, what is the maximum value $\mathcal{G}$ can achieve to approximate $f(\mathbf{x}^*)$?**
4. **The answer is:**

$$\max \mathcal{G}(\boldsymbol{\lambda}, \boldsymbol{\mu}) \quad \textbf{s.t. } \boldsymbol{\lambda} \geq 0.$$

Part II

Shallow Model

CHAPTER 1

PERCEPTRON

1 The Structure

The perceptron (Figure 1.2) is an algorithm for learning a binary classifier called a threshold function: a function that maps its input $\mathbf{x}$ (a real-valued vector) to an output value $f(\mathbf{x})$ (a single binary value):

$$f(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + b) \tag{1.1}$$

where $\sigma(z)$ is a function that outputs $+1$ if $z \geq 0$ and -1 otherwise, $\mathbf{w}$ is a vector of real-valued weights, $\mathbf{w}^T\mathbf{x}$ is the dot product

$$\mathbf{w}^T\mathbf{x} = \sum_{i=1}^{m} w_i x_i, \tag{1.2}$$

where m is the number of inputs to the perceptron, and b is the bias. The bias shifts the decision boundary away from the origin and does not depend on any input value.

Equivalently, since

$$\mathbf{w}^T\mathbf{x} + b = (\mathbf{w}, b) \cdot (\mathbf{x}, 1), \tag{1.3}$$

we can add the bias term b as another weight $\hat{\mathbf{w}}$ and add a coordinate 1 to each input $\hat{\mathbf{x}}$, and then write it as a linear classifier that passes through the origin:

$$f(\mathbf{x}) = \sigma(\hat{\mathbf{w}}^T\hat{\mathbf{x}}) \tag{1.4}$$

The binary value of $f(\mathbf{x})$ (0 or 1) is used to perform binary classification on $\mathbf{x}$ as either a positive or a negative instance. Spatially, the bias shifts the position (though not the orientation) of the planar decision boundary.

2 XOR Problem

Theorem 2.1: Linear Separable

If 3 points are collinear and the middle point has a different label than the other two, then these 3 points cannot be linearly separable.

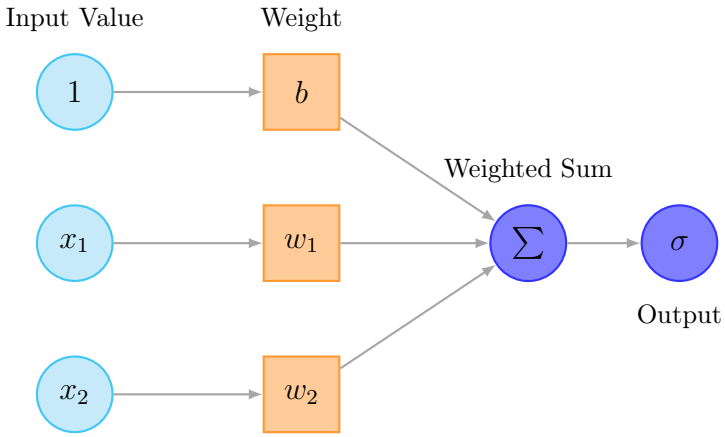


Figure 1.1: The perceptron structure

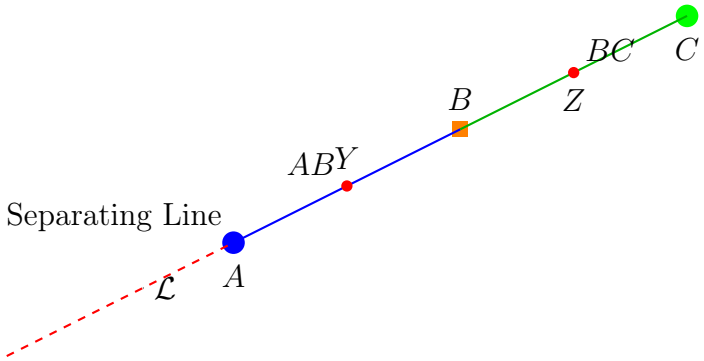


Figure 1.2: The linear separable problem

Proof for Theorem.

Let us label the points as point A, B, and C. A and C have the same label, and B has a different label. They are all collinear with line $\mathcal{L}$.

Assume a contradiction: a line can linearly separate A, B, and C. This means a line must cross between segment AB and segment BC to linearly separate these three points (by the definition of linear separability). Let us label the point where the line crosses segment AB as point Y, and the point where the line crosses segment BC as point Z.

However, since segments AB and BC are collinear with line $\mathcal{L}$, points Y and Z also fall on line $\mathcal{L}$. Since only one unique line can pass through two points, it must be that the only line that passes through segments AB and BC (and therefore separates points A, B, and C) is line $\mathcal{L}$.

However, line $\mathcal{L}$ cannot linearly separate A, B, and C, since line $\mathcal{L}$ also passes through them. Therefore, no line exists that can separate A, B, and C (See Figure 1.2 for illustration). ■

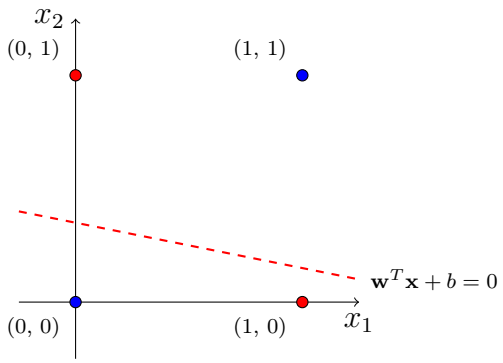


Figure 1.3: The perceptron can not solve XOR problems.

Theorem 2.2: Perceptron can not solve XOR problem (Fig. 1.3)

The XOR problem involves the following input-output pairs:

x_1	x_2	$y = \text{XOR}(x_1, x_2)$
0	0	0
0	1	1
1	0	1
1	1	0

Here, y represents the XOR output. The task is to find $\mathbf{w}$ and b such that the perceptron correctly classifies all four input-output pairs.

Proof for Theorem.

A perceptron can solve a problem if and only if the data points are **linearly separable**. This means there exists a hyperplane:

$$\mathbf{w}^T\mathbf{x} + b = 0$$

that separates the two classes $y = 0$ and $y = 1$.

Assume, for contradiction, that there exists a perceptron with weights $\mathbf{w}$ and bias b that solves the XOR problem. Then the perceptron must satisfy the following inequalities:

1. $b < 0$ (for $(x_1, x_2) = (0, 0)$, $y = 0$),
2. $w_2 + b > 0$ (for $(x_1, x_2) = (0, 1)$, $y = 1$),
3. $w_1 + b > 0$ (for $(x_1, x_2) = (1, 0)$, $y = 1$),
4. $w_1 + w_2 + b < 0$ (for $(x_1, x_2) = (1, 1)$, $y = 0$).

Adding inequalities 2 and 3:

$$(w_2 + b) + (w_1 + b) > 0 \implies w_1 + w_2 + 2b > 0.$$

But inequality 4 states:

$$w_1 + w_2 + b < 0.$$

This leads to a contradiction, because $w_1 + w_2 + 2b > 0$ cannot simultaneously satisfy $w_1 + w_2 + b < 0$ with $b < 0$. ∎

3 Learning Algorithm

Algorithm 3.1: Perceptron Learning Algorithm

Input: Training set $\{(\hat{\mathbf{x}}_i, y_i)\}_{i=1}^n$, learning rate η
Initialize: $\hat{\mathbf{w}}_1 = 0$
for t = 1 to T **do**
  for each training sample $(\hat{\mathbf{x}}_i, y_i)$ **do**
    if the sample is misclassified **then**
      $\hat{\mathbf{w}}_{t+1} = \hat{\mathbf{w}}_t + \eta y_i \hat{\mathbf{x}}_i$
    else
      leave $\hat{\mathbf{w}}_{t+1}$ unchanged
    end if
  end for
end for
Output: Weight vector $\hat{\mathbf{w}}_T$

Fig. 1.4 illustrates the learning algorithm of perceptron.

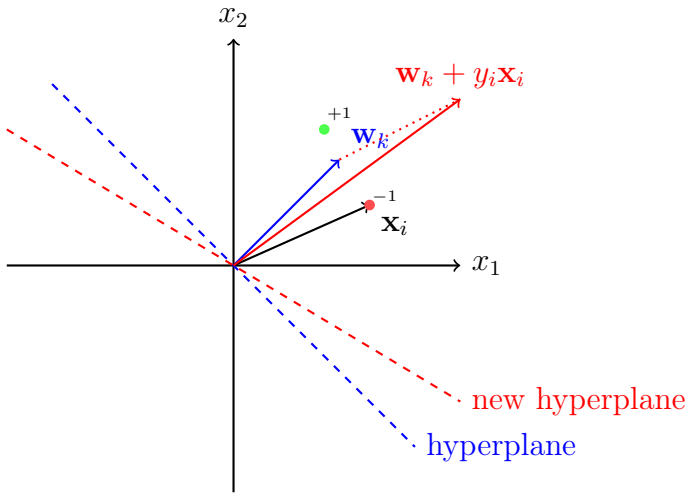


Figure 1.4: This figure illustrates the perceptron update process. The blue dashed line represents the initial hyperplane, which is perpendicular to the weight vector $\mathbf{w}_k$ (blue arrow). The red dashed line represents the new hyperplane after the weight vector is updated to $\mathbf{w}_k + y_i\mathbf{x}_i$ (red arrow), which rotates the hyperplane according to the position of $\mathbf{x}_i$. A misclassified sample $\mathbf{x}_i$ (red point) causes this update, while a correctly classified sample (green point) remains unaffected.

Theorem 3.1: Bound on the number of errors in the Perceptron algorithm in Algorithm 3.1

Assume that there exists some parameter vector $\mathbf{w}^*$ such that $\|\mathbf{w}^*\| = 1$, and some $\gamma > 0$ such that for all $t = 1, \ldots, n$:

$$y_t(\mathbf{x}_t^\top \mathbf{w}^*) \geq \gamma$$

Assume in addition that for all $t = 1, \ldots, n$, $\|\mathbf{x}_t\| \leq R$. Under these assumptions, the Perceptron algorithm makes at most:

$$\frac{R^2}{\gamma^2}$$

errors.

Definition of an error: An error occurs whenever we have $f(\mathbf{x_t}) \neq y_t$ for some (j, t) pair in the algorithm.

Proof for Theorem.

To show that the number of errors is bounded, we define $\mathbf{w}_k$ as the weight vector after the k-th error. Initially, we have $\mathbf{w}_1 = 0$. When the algorithm makes an error on an example $(\mathbf{x}_t, y_t)$, the update rule is:

$$\mathbf{w}_{k+1} = \mathbf{w}_k + y_t \mathbf{x}_t$$

We then calculate the inner product $\mathbf{w}_k^\top \mathbf{w}^*$, where $\mathbf{w}^*$ is the optimal separating hyperplane. Since the Perceptron algorithm updates the weight vector in the direction of the misclassified example, we have:

$$\mathbf{w}_{k+1}^\top \mathbf{w}^* = (\mathbf{w}_k + y_t \mathbf{x}_t)^\top \mathbf{w}^* = \mathbf{w}_k^\top \mathbf{w}^* + y_t(\mathbf{x}_t^\top \mathbf{w}^*) \geq \mathbf{w}_k^\top \mathbf{w}^* + \gamma$$

where γ is the margin. By induction, after k updates, we have:

$$\mathbf{w}_k^\top \mathbf{w}^* \geq k\gamma$$

Next, we bound the squared norm of $\mathbf{w}_k$. From the update rule, we have:

$$\|\mathbf{w}_{k+1}\|^2 = \|\mathbf{w}_k\|^2 + 2y_t(\mathbf{w}_k^\top \mathbf{x}_t) + \|\mathbf{x}_t\|^2$$

Assume $\|\mathbf{x}_t\| \leq R$, we obtain:

$$\|\mathbf{w}_{k+1}\|^2 = \|\mathbf{w}_k + y_t \mathbf{x}_t\|^2 = \|\mathbf{w}_k\|^2 + y_t^2 \|\mathbf{x}_t\|^2 + 2y_t(\mathbf{w}_k^\top \mathbf{x}_t) \leq \|\mathbf{w}_k\|^2 + R^2$$

The inequality follows because: 1) $y_t^2 \|\mathbf{x}_t\|^2 = \|\mathbf{x}_t\|^2 \leq R^2$, as assumed, and because $y_t^2 = 1$. 2) $y_t(\mathbf{w}_k^\top \mathbf{x}_t) \leq 0$, since the parameter vector $\mathbf{w}_k$ gave an error on the t-th example.

It follows by induction on k (recall that $\|\mathbf{w}_1\|^2 = 0$), that:

$$\|\mathbf{w}_{k+1}\|^2 \leq kR^2$$

Combining the lower and upper bounds, we have:

$$k^2\gamma^2 \leq \|\mathbf{w}_{k+1}\|^2 \leq kR^2$$

from which it follows that:

$$k \leq \frac{R^2}{\gamma^2}.$$

CHAPTER 2

SUPPORT VECTOR MACHINE

1 Linear SVM

To understand SVM, it is helpful to compare it with the Perceptron algorithm. A key distinction between the two lies in their stopping conditions. The Perceptron algorithm terminates once it successfully classifies all the training data points correctly. In contrast, the SVM algorithm concludes after identifying the optimal hyperplane that maximizes the margin, defined as the maximum distance between the data points of the two classes. By maximizing this margin, SVM offers greater robustness, enabling more confident classification of future data points (test datasets).

As the same as perceptron, the decision function of SVM is:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + b), \tag{2.1}$$

where $\mathbf{x} \in \mathbb{R}^m$ is the input vector, $\mathbf{w} \in \mathbb{R}^m$ is a real-valued weight vector, $b \in \mathbb{R}$ is the bias term, and $\text{sign}(z)$ returns $+1$ if $z \geq 0$ and -1 otherwise.

The margin is another core concept in SVM. It refers to the distance between a data point and the hyperplane. When there are many sample points, the margin is defined as the minimum distance between the sample points and the hyperplane.

Fig. 2.1 illustrates the definition of the margin, which is the shortest distance from the hyperplane to the closest data points. However, it is evident that the hyperplane in this example is suboptimal, as it is very close to the positive samples while being far from the negative samples. The goal of SVM is to identify the most appropriate hyperplane to separate the data, with one side containing positive samples and the other containing negative samples. Intuitively, the most appropriate hyperplane should lie "in the middle" between the positive and negative samples. In other words, it aims to find the hyperplane that maximizes the margin. Such a hyperplane is located precisely at the center of the positive and negative samples, because if the hyperplane is too close to any sample, the margin will shrink. Therefore, to ensure the margin is maximized, the hyperplane must remain sufficiently far from all positive and negative samples.

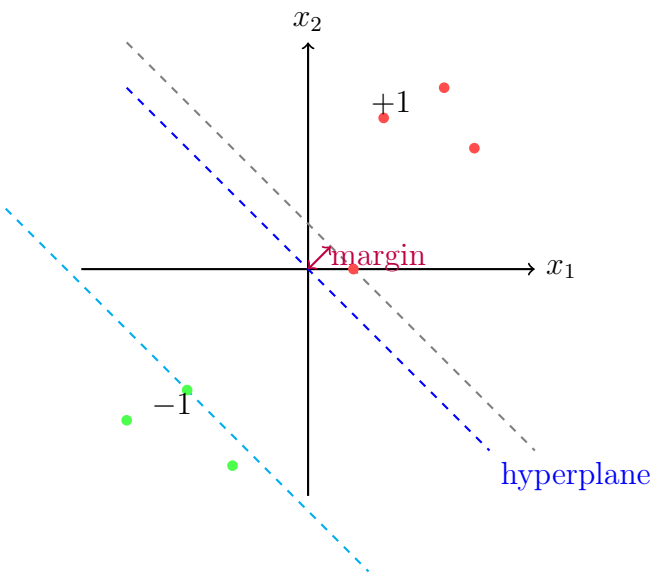


Figure 2.1: This figure illustrates the concept of margin and hyperplane in SVM. The blue dashed line represents the hyperplane ($\mathbf{w}^T\mathbf{x} + b = 0$), separating the two classes. The gray dashed lines represent the margin boundaries ($\mathbf{w}^T\mathbf{x} + b = \pm 1$). The purple double arrow shows the margin, which is the shortest distance from the hyperplane to the closest data points (support vectors).

1.1 The objective function

To calculate the margin of the hyperplane $\mathbf{w}^T\mathbf{x} + b = 0$, we can introduce two additional hyperplanes, similar to the gray dashed lines that are perpendicular to the purple margin line in Fig. 2.1: $\mathbf{w}^T\mathbf{x} + b = \delta$ and $\mathbf{w}^T\mathbf{x} + b = -\delta$, where $\delta > 0$. Note that no data points can lie between these two hyperplanes. Therefore, they must satisfy a constraint: for positive samples ($y_i = 1$), $\mathbf{w}^T\mathbf{x}_i + b \geq \delta$; and for negative samples ($y_i = -1$), $\mathbf{w}^T\mathbf{x}_i + b \leq -\delta$.

As a result, calculating the margin of the original hyperplane is equivalent to determining the distance between these two hyperplanes. Using the sign of the labels, the constraint for the two hyperplanes can be unified as:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq \delta, \quad \forall i = 1, 2, \cdots, n \tag{2.2}$$

Suppose the two hyperplanes are defined as $H_0 : \mathbf{w}^T\mathbf{x} + b = -\delta$ and $H_1 : \mathbf{w}^T\mathbf{x} + b = \delta$, where $\mathbf{x}_0$ is a point on H_0. The distance between the two hyperplanes is denoted as d.

It is already known that $\mathbf{w} \perp H_1$, so the direction of this vector should be the same as $\mathbf{w}$, and its length is exactly the distance d we are trying to compute. Therefore, the vector $\overline{\mathbf{x}_0\mathbf{x}_1}$ is given by:

$$\overline{\mathbf{x}_0\mathbf{x}_1} = \frac{d\mathbf{w}}{\|\mathbf{w}\|}, \tag{2.3}$$

which implies:

$$\mathbf{x}_1 = \mathbf{x}_0 + \frac{d\mathbf{w}}{\|\mathbf{w}\|}. \tag{2.4}$$

Now, substitute $\mathbf{x}_1$ into the equation for H_1 ($\mathbf{w}^T\mathbf{x} + b = \delta$):

$$\mathbf{w}^T\mathbf{x}_1 + b = \mathbf{w}^T\left(\mathbf{x}_0 + \frac{d\mathbf{w}}{\|\mathbf{w}\|}\right) + b = \mathbf{w}^T\mathbf{x}_0 + \frac{d\mathbf{w}^T\mathbf{w}}{\|\mathbf{w}\|} + b = \delta \tag{2.5}$$

Since $\mathbf{x}_0$ is a point on H_0, it satisfies $\mathbf{w}^T\mathbf{x}_0 + b = -\delta$. Substituting this into equation (2), we get:

$$\mathbf{w}^T\mathbf{x}_1 + b = -\delta + \frac{d\|\mathbf{w}\|^2}{\|\mathbf{w}\|} = -\delta + d\|\mathbf{w}\| = \delta. \tag{2.6}$$

From this, we derive the expression for d:

$$d = \frac{2\delta}{\|\mathbf{w}\|}. \tag{2.7}$$

Since δ is a positive constant, to maximize m, we must minimize $\|\mathbf{w}\|$. Considering the constraint for the two hyperplanes (1), we can now derive the objective function for SVM:

$$\min_{\mathbf{w},b} \|\mathbf{w}\| \quad \text{s.t.} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq \delta, \quad i = 1, \ldots, n. \tag{2.8}$$

To find the optimal hyperplane, we need to determine the $\mathbf{w}$ and b that minimize this function. Note that since we have n samples, there are n constraints in total.

1.2 Lagrangian Duality

In the previous article, we derived the objective function of the SVM:

$$\min_{\mathbf{w},b} \|\mathbf{w}\| \quad \text{s.t.} \quad y_i(\mathbf{w}^\top\mathbf{x}_i + b) \geq \delta, \quad i = 1, \ldots, n \tag{2.9}$$

Since in the solving process, the constant δ in the constraint does not affect the result, we simply replace δ with 1 for simplicity. Furthermore, for convenience in solving the optimization problem later, we replace $\|\mathbf{w}\|$ in the original objective function with $\frac{1}{2}\|\mathbf{w}\|^2$. Optimizing the latter is equivalent to optimizing the former, and the final result remains unchanged.

Thus, we obtain the most common objective function for SVM:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{s.t.} \quad y_i(\mathbf{w}^\top\mathbf{x}_i + b) \geq 1, \quad i = 1, \ldots, n \tag{2.10}$$

The SVM primal problem satisfies the KKT conditions described in Section 2.2 of Chapter 1 in Part I. Since the SVM primal problem satisfies all these conditions, we can directly apply the Lagrangian multiplier method. First, we write the Lagrangian function:

$$\begin{aligned} \min \quad & \mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\mathbf{w}^\top\mathbf{x}_i + b) - 1\right) \\ \text{s.t.} \quad & \alpha_i \geq 0, \quad \forall i = 1, \ldots, n \end{aligned} \tag{2.11}$$

Then, by setting $\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 0$ and $\frac{\partial \mathcal{L}}{\partial b} = 0$, we obtain the following system of equations:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = 0 \tag{2.12}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{2.13}$$

In cases where the constraints are equality constraints, we would also derive additional equations from $\frac{\partial \mathcal{L}}{\partial \alpha} = 0$, which could then be used to solve for $\mathbf{w}$ and b.

However, since the current constraints are inequality constraints, the result from $\frac{\partial \mathcal{L}}{\partial \alpha}$ leads to a set of inequalities:

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1 \geq 0, \quad i = 1, 2, \ldots, n. \tag{2.14}$$

These inequalities make it impossible to directly solve for $\mathbf{w}$ and b.

From Equation (2.12), we solve for:

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \tag{2.15}$$

Substituting Equation (2.15) into Equation (2.11), we obtain:

$$\mathcal{G}(\boldsymbol{\alpha}, b) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) - b \sum_{i=1}^{n} \alpha_i y_i \tag{2.16}$$

From Equation (2.13), it is already established that:

$$\sum_{i=1}^{n} \alpha_i y_i = 0, \tag{2.17}$$

so Equation (2.16) simplifies to:

$$\mathcal{G}(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j). \tag{2.18}$$

Equation (2.18) is the final version of the dual form. Hence, we obtain the Lagrangian dual problem for SVM:

$$\max_{\boldsymbol{\alpha}} \quad \mathcal{G}(\boldsymbol{\alpha}) \quad \text{s.t.} \quad \sum_{i=1}^{n} \alpha_i y_i = 0, \quad \alpha_i \geq 0, \quad i = 1, \ldots, n. \tag{2.19}$$

After solving for $\boldsymbol{\alpha}$, we can compute $\mathbf{w}$ using Equation (2.12), and subsequently determine b based on the margin of the hyperplane.

1.3 Soft-margin SVM

In datasets with many outliers, we allow a small number of data points to fall within the margin. The key is the introduction of slack variables ξ_i. Previously, the margin was defined by the hard constraint:

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \quad y_i \in \{-1, 1\}. \tag{2.20}$$

In the new formulation, we soften the constraint to:

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \tag{2.21}$$

where ξ_i are the slack variables. This new constraint permits a functional margin that is less than 1 and introduces a penalty of cost $C\xi_i$ for any data point that: - Falls within the margin on the correct side of the separating hyperplane (i.e., $0 < \xi_i \leq 1$). - Lies on the wrong side of the separating hyperplane (i.e., $\xi_i > 1$).

This soft-margin formulation expresses a preference for correctly classifying the training data while relaxing the constraints to allow for non-separable data. The penalty is proportional to the amount by which a given example is misclassified. The parameter C controls the trade-off between maximizing the margin and minimizing classification errors.

We aim to minimize the sum total of the penalties ξ_i over all i, which serves as an upper bound on the training classification errors. The new well-posed optimization problem (using ℓ_1-regularization) becomes:

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i, \\ \text{s.t.} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1, \ldots, n. \end{aligned} \tag{2.22}$$

We take the Lagrangian in the usual manner:

$$\mathcal{L}(\mathbf{w}, \boldsymbol{\xi}, b, \boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w}^\top \mathbf{w} + C\sum_{i=1}^{n} \xi_i + \sum_{i=1}^{n} \alpha_i \left[1 - \xi_i - y_i(\mathbf{x}_i^\top \mathbf{w} + b)\right]. \tag{2.23}$$

1. Differentiating $\mathcal{L}$ with respect to $\mathbf{w}$:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = 0. \tag{2.24}$$

2. Differentiating $\mathcal{L}$ with respect to b:

$$\frac{\partial \mathcal{L}}{\partial b} = \sum_{i=1}^{n} \alpha_i y_i = 0. \tag{2.25}$$

3. Differentiating $\mathcal{L}$ with respect to ξ_i:

$$\frac{\partial \mathcal{L}}{\partial \xi_i} = C - \alpha_i - \beta_i = 0 \quad \Rightarrow \quad \alpha_i \leq C \text{ and } \beta_i \geq 0. \tag{2.26}$$

By substituting $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$ into the objective function and eliminating ξ_i, we obtain the dual form:

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \left\{ \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \right\}, \\ \text{s.t.} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C, \quad i = 1, 2, \ldots, n. \end{aligned} \tag{2.27}$$

2 Nonlinear SVM

The previously described SVM is still a linear classifier, which cannot address the issue of non-linearly separable problems inherent in the Perceptron. As shown in Fig. 2.2, to overcome the limitations of linear classifiers, SVM introduces the concept of **feature mapping**, defined as:

$$\mathbf{x} \mapsto \boldsymbol{\phi}(\mathbf{x}), \tag{2.28}$$

where $\boldsymbol{\phi}(\mathbf{x})$ is a mapping from the input space to a higher-dimensional feature space. In this transformed space, the data that was originally non-linearly separable may become linearly separable.

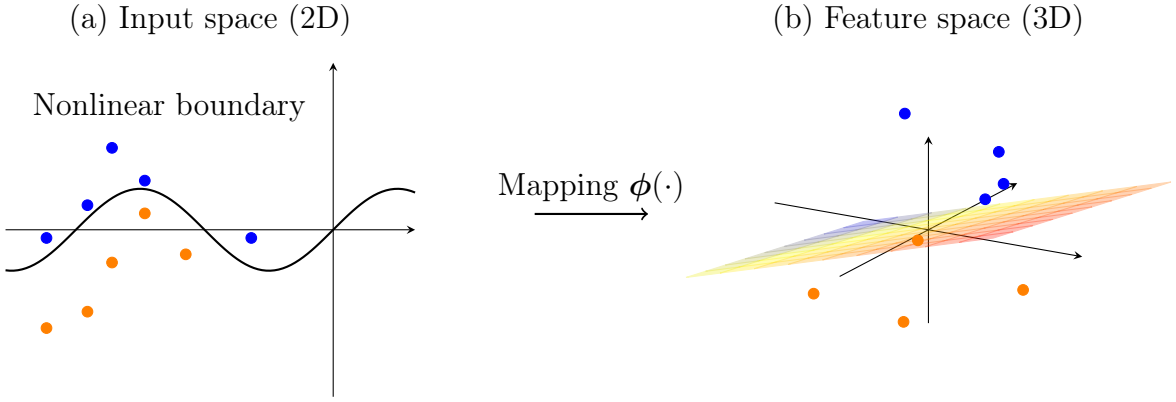


Figure 2.2: SVM uses the kernel method to transform linearly inseparable problems into linearly separable ones in a higher-dimensional space.

To create a quadratic decision boundary, a feature mapping $\phi(\mathbf{x})$ is required to transform the input data into a higher-dimensional space. For example, a possible feature mapping is given as:

$$\boldsymbol{\phi}(\mathbf{x}) = \left(1, \sqrt{2}x_1, \sqrt{2}x_2, \ldots, \sqrt{2}x_d, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \ldots, \sqrt{2}x_{d-1}x_d, x_1^2, \ldots, x_d^2\right), \tag{2.29}$$

where pairwise terms are defined for $i < j$. The dimensionality of the transformed space is approximately $O(d^2)$, which becomes problematic for large d. This is due to the rapid growth in the number of features, making computations inefficient or even infeasible.

Linear algorithms, such as SVM, are fundamentally based on the computation of inner products (objective function in Equation (2.18). A key question arises: *What if we could compute the inner product in the feature space without explicitly computing the feature mapping* $\boldsymbol{\phi}(\mathbf{x})$?

For example, consider the feature mapping:

$$\boldsymbol{\phi}(\mathbf{x}) = \left(1, \sqrt{2}x_1, \sqrt{2}x_2, \ldots, \sqrt{2}x_d, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \ldots, \sqrt{2}x_{d-1}x_d, x_1^2, \ldots, x_d^2\right). \tag{2.30}$$

The inner product of two feature-mapped vectors $\boldsymbol{\phi}(\mathbf{x})$ and $\boldsymbol{\phi}(\mathbf{y})$ is given by:

$$\langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{y}) \rangle = 1 + \sum_{i=1}^{d} 2x_iy_i + \sum_{i=1}^{d}\sum_{j=1}^{d} x_ix_jy_iy_j = (1 + \langle \mathbf{x}, \mathbf{y} \rangle)^2. \tag{2.31}$$

This computation shows that the inner product in the high-dimensional feature space can be expressed as a simple function of the inner product in the original input space. Let us denote the inner product in the feature space by:

$$K(\mathbf{x}, \mathbf{y}) \triangleq \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{y}) \rangle. \tag{2.32}$$

In this case, the kernel $K(\mathbf{x}, \mathbf{y})$ is called the **polynomial kernel**, and it is defined as:

$$K(\mathbf{x}, \mathbf{y}) = (1 + \langle \mathbf{x}, \mathbf{y} \rangle)^2 . \tag{2.33}$$

The key advantage of using kernels is that we can compute $K(\mathbf{x}, \mathbf{y})$ in $O(d)$ memory and computation time, without ever explicitly computing $\boldsymbol{\phi}(\mathbf{x})$ in the high-dimensional feature space. This makes the kernel trick both computationally efficient and memory-efficient.

If, instead of using the original input $\mathbf{x}$, we first map it to a feature space $\phi(\mathbf{x})$, the dual optimization problem remains similar. The only difference is that the inner product in the input space $\mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}$ is replaced by the inner product in the feature space $\boldsymbol{\phi}(\mathbf{x}^{(i)}) \cdot \boldsymbol{\phi}(\mathbf{x}^{(j)})$. This inner product can be computed using the kernel function $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$, avoiding the explicit computation of $\boldsymbol{\phi}(\mathbf{x})$.

Thus, the final dual optimization problem becomes:

$$\begin{aligned} \max_{\alpha_i} \quad & \left\{ \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \right\}, \\ \text{s.t.} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C, \quad i = 1, 2, \ldots, n. \end{aligned} \tag{2.34}$$

where α_i are the Lagrange multipliers, y_i represents the class label ($+1$ or -1), and $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ is the kernel function.

From the final optimization form, we can observe that the objective function is independent of $\boldsymbol{\phi}()$ and relies only on the kernel function in Equation (2.34). This raises an important question: how do we choose the kernel function? This is where Mercer's Theorem comes into play.

1. **Mercer's Theorem (1909):**
   Any "reasonable" kernel function corresponds to a mapping into some feature space.

2. **Definition of "Reasonable":**
   A kernel function $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ is considered reasonable if it produces a Gram matrix $K_{ij} = K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ that is **positive semidefinite**.

3. **Implications of Positive Semidefiniteness:**

   - Kernel functions that are positive semidefinite ensure that the data is implicitly mapped to a high-dimensional feature space, where SVM optimization can take place.
   - Complex kernel functions can be constructed as long as the resulting Gram matrix remains positive semidefinite.

4. **Combining Kernels:**
   Simple kernels can be combined to create more complex kernels while preserving positive semidefiniteness. For instance:

   - Additive combinations: $K_1 + K_2$
   - Multiplicative combinations: $K_1 \cdot K_2$
   - Scaling by positive constants: cK_1, where $c > 0$.

The equivalent definition of Mercer's Theorem is particularly useful for constructing kernel functions. However, testing whether a function qualifies as a valid kernel is challenging because it requires verifying the Gram matrix's positive semidefiniteness for arbitrary inputs $\mathbf{x}_i, \mathbf{x}_j, \ldots$. The difficulty lies in the word "arbitrary." As a result, in practical applications, pre-existing, well-established kernel functions are typically used.

Commonly Used Kernel Functions

1. **Linear Kernel:**
$$K(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top \mathbf{y} \tag{2.35}$$
   The linear kernel is the simplest of all kernel functions. It maps the input data without any transformation.

2. **Polynomial Kernel:**
$$K(\mathbf{x}, \mathbf{y}) = (a\mathbf{x}^\top \mathbf{y} + c)^p \tag{2.36}$$
   where a, c, and p are real-valued parameters. The linear kernel is a special case of the polynomial kernel when $p = 1$.

3. **Gaussian Kernel (RBF Kernel):**
$$K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\sigma^2}\right) \tag{2.37}$$
   The Gaussian kernel maps input data into an infinite-dimensional feature space. It is based on the idea that data points that are non-linearly separable in a low-dimensional space may become linearly separable in a higher-dimensional space. In fact, **in infinite-dimensional space, data is guaranteed to be linearly separable**. This makes the Gaussian kernel a must-try in non-linear SVM problems.

4. **Power-Law Kernel:**
$$K(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^{-d}, \quad d > 0 \tag{2.38}$$
   This kernel measures similarity based on the inverse power of the distance.

5. **Laplacian Kernel:**
$$K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|}{\sigma}\right) \tag{2.39}$$
   Similar to the Gaussian kernel, the Laplacian kernel is based on the exponential function but uses the absolute distance rather than the squared distance.

These kernels provide a range of options for different types of data and decision boundaries. Selecting the appropriate kernel function depends on the characteristics of the problem and the structure of the data.

3 Training Support Vector Machines

Similar to the challenges encountered in deep learning, solving Equation (2.34) becomes extremely difficult when the training set is large. The quadratic form in Equation (2.34) involves a matrix with a number of elements equal to the square of the number of training examples.

Sequential Minimal Optimization (SMO) is a simple algorithm that can quickly solve the SVM problem without any extra matrix storage. At every step, SMO chooses two Lagrange multipliers to jointly optimize, finds the optimal values for these multipliers, and updates the SVM to reflect the new optimal values. The advantage of SMO lies in the fact that solving for two Lagrange multipliers can be done analytically.

Without loss of generality, we will assume that two elements, α_1 and α_2, have been chosen for updating to improve the objective value. In order to compute the new values for these two parameters, one can observe that, in order not to violate the linear constraint:

$$\sum_{i=1}^{n} y_i \alpha_i = 0, \tag{2.40}$$

the new values of the multipliers must lie on a line:

$$y_1 \alpha_1^{(\text{old})} + y_2 \alpha_2^{(\text{old})} = \text{constant} = y_1 \alpha_1 + y_2 \alpha_2, \tag{2.41}$$

in the (α_1, α_2) space, and within the box defined by $0 \leq \alpha_1, \alpha_2 \leq C$, as shown in Fig. 2.3.

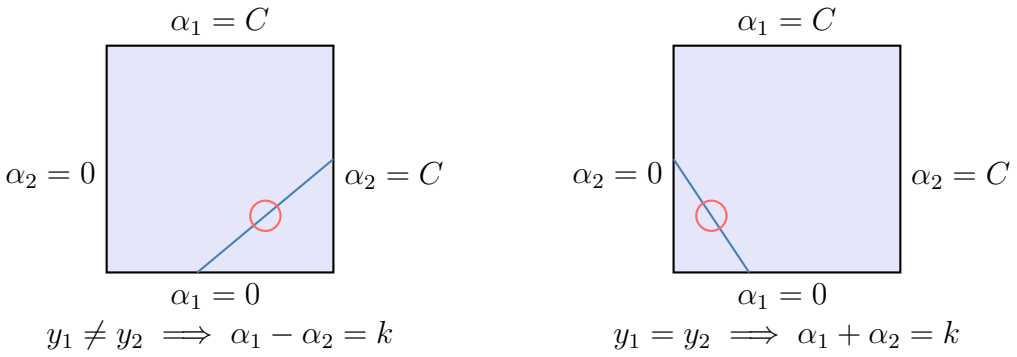


Figure 2.3: The two Lagrange multipliers must fulfill all of the constraints of the full problem. The inequality constraints cause the Lagrange multipliers to lie in the box. The linear equality constraint causes them to lie on a diagonal line.

Without loss of generality, the algorithm first computes $\alpha_2^{(\text{new})}$ and successively uses it to obtain $\alpha_1^{(\text{new})}$. The box constraint $0 \leq \alpha_1, \alpha_2 \leq C$, together with the linear equality constraint, provides a more restrictive constraint on the feasible values for $\alpha_2^{(\text{new})}$:

$$U \leq \alpha_2^{(\text{new})} \leq V, \tag{2.42}$$

where U and V are defined as follows:

$$\begin{aligned}
&\text{if } y_1 \neq y_2 \\
&\begin{cases} U = \max\{0, \alpha_2^{(\text{old})} - \alpha_1^{(\text{old})}\}, \\ V = \min\{C, C - \alpha_1^{(\text{old})} + \alpha_2^{(\text{old})}\}, \end{cases} \\
&\text{if } y_1 = y_2 \\
&\begin{cases} U = \max\{0, \alpha_1^{(\text{old})} + \alpha_2^{(\text{old})} - C\}, \\ V = \min\{C, \alpha_1^{(\text{old})} + \alpha_2^{(\text{old})}\}. \end{cases}
\end{aligned} \tag{2.43}$$

Now, let's begin deriving the optimization process for SMO. For representation simplicity, let's define the following symbols for each element of matrix K:

$$K(\mathbf{x}_i, \mathbf{x}_j) \equiv K_{ij} \tag{2.44}$$

$$f(\mathbf{x}_i) \equiv \sum_{j=1}^{\ell} \alpha_j y_j K_{ji} + b \tag{2.45}$$

$$v_i \equiv \sum_{j=3}^{\ell} \alpha_j y_j K_{ij} = f(\mathbf{x}_i) - \sum_{j=1}^{2} \alpha_j y_j K_{ij} - b, \quad i = 1, 2, \cdots, n \tag{2.46}$$

$$E_i \equiv f(\mathbf{x}_i) - y_i = \left(\sum_{j=1}^{\ell} \alpha_j y_j K_{ji} + b \right) - y_i, \quad i = 1, 2, \cdots, n \tag{2.47}$$

Consider the objective as a function of α_1 and α_2:

$$\begin{aligned}
\mathcal{G}(\alpha_1, \alpha_2) = \alpha_1 + \alpha_2 &- \frac{1}{2} K_{11} \alpha_1^2 - \frac{1}{2} K_{22} \alpha_2^2 - \alpha_1 \alpha_2 y_1 y_2 K_{12} \\
&- y_1 \alpha_1 \sum_{j=3}^{n} y_j \alpha_j K_{1j} - y_2 \alpha_2 \sum_{j=3}^{n} y_j \alpha_j K_{2j} + \sum_{i=3}^{n} \alpha_i \\
&- \frac{1}{2} \sum_{i=3}^{n} \sum_{j=3}^{n} \alpha_i \alpha_j y_i y_j K_{ij}.
\end{aligned} \tag{2.48}$$

Substitute Equations (2.45),(2.46) and (2.47) into (2.48) yields

$$\mathcal{G}(\alpha_1, \alpha_2) = \alpha_1 + \alpha_2 - \frac{1}{2} K_{11} \alpha_1^2 - \frac{1}{2} K_{22} \alpha_2^2 - \alpha_1 \alpha_2 y_1 y_2 K_{12} - y_1 \alpha_1 v_1 - y_2 \alpha_2 v_2 + \text{constant} \tag{2.49}$$

Note also that the constraint

$$\sum_{i=1}^{n} \alpha_i^{(\text{old})} y_i = \sum_{i=1}^{n} \alpha_i y_i = 0, \tag{2.50}$$

implies the condition

$$\alpha_1 + s\alpha_2 = \text{constant} = \alpha_1^{(\text{old})} + s\alpha_2^{(\text{old})} = \gamma, \tag{2.51}$$

where $s = y_1 y_2$ ($y_i y_i = 1$). The above equation demonstrates how $\alpha_1^{(\text{new})}$ is computed from $\alpha_2^{(\text{new})}$:

$$\alpha_1 = \gamma - s\alpha_2. \tag{2.52}$$

Eliminating α_1 in (20), we have the objective function as α_2:

$$\begin{aligned}
\mathcal{G}(\alpha_2) &= \gamma - s\alpha_2 + \alpha_2 - \frac{1}{2}K_{11}(\gamma - s\alpha_2)^2 - \frac{1}{2}K_{22}\alpha_2^2 - sK_{12}(\gamma - s\alpha_2)\alpha_2 \\
&\quad - y_1(\gamma - s\alpha_2)v_1 - y_2\alpha_2 v_2 + \text{constant}. \\
&= -\frac{1}{2}K_{11}(\gamma^2 - 2\gamma s\alpha_2 + s^2\alpha_2^2) - \frac{1}{2}K_{22}\alpha_2^2 + s^2K_{12}\alpha_2^2 \\
&\quad + (1 - s - sK_{12}\gamma)\alpha_2 - y_1v_1y_2v_2 + \text{constant}. \\
&= \frac{1}{2}(2K_{12} - K_{11} - K_{22})\alpha_2^2 + (1 - s + K_{11}s\gamma - K_{12}s\gamma + y_2v_1 - y_2v_2)\alpha_2 + \text{constant}.
\end{aligned} \tag{2.53}$$

The stationary points satisfy

$$\frac{\mathrm{d}\mathcal{G}(\alpha_2)}{\mathrm{d}\alpha_2} = (2K_{12} - K_{11} - K_{22})\alpha_2 + (1 - s + K_{11}s\gamma - K_{12}s\gamma + y_2v_1 - y_2v_2) = 0. \tag{2.54}$$

This yields

$$\alpha_2^{(\text{new,unc})}(K_{11} + K_{22} - 2K_{12}) = 1 - s + K_{11}s\gamma - K_{12}s\gamma + y_2v_1 - y_2v_2, \tag{2.55}$$

$$\alpha_2^{(\text{new,unc})} = \frac{1 - s + (K_{11} - K_{12})s\gamma + y_2(v_1 - v_2)}{K_{11} + K_{22} - 2K_{12}}. \tag{2.56}$$

Multiplying Equation (2.56) by y_2 and with $\kappa = K_{11} + K_{22} - 2K_{12}$, it is easy to see

$$\alpha_2^{(\text{new,unc})}\kappa y_2 = y_2 - y_1 + (K_{11} - K_{12})y_1\gamma + v_1 - v_2, \tag{2.57}$$

$$= y_2 - y_1 + (K_{11} - K_{12})y_1\gamma + \left(f(x_1) - \sum_{j=1}^{2} y_j\alpha_j K_{1j}\right) \tag{2.58}$$

and

$$y_1\gamma = y_1(\alpha_1 + s\alpha_2) = y_1\alpha_1 + y_2\alpha_2 \tag{2.59}$$

Since

$$\sum_{j=1}^{2} y_j\alpha_j K_{2j} - \sum_{j=1}^{2} y_j\alpha_j K_{1j} = y_1\alpha_1K_{21} + y_2\alpha_2K_{22} - y_1\alpha_1K_{11} + y_2\alpha_2K_{12}, \tag{2.60}$$

Substitute Equation. (2.59) and (2.60) into (2.56), we can find

$$\begin{aligned}
\alpha_2^{(\text{new,unc})}\kappa y_2 &= y_2 - y_1 + (K_{11} - K_{12})(\alpha_1y_1 + \alpha_2y_2) + y_1\alpha_1K_{21} \\
&\quad + y_2\alpha_2K_{22} - y_1\alpha_1K_{11} + y_2\alpha_2K_{12} + f(\mathbf{x}_1) - f(\mathbf{x}_2) \\
&= y_2 - y_1 + f(\mathbf{x}_1) - f(\mathbf{x}_2) + y_2\alpha_2K_{11} \\
&\quad - y_2\alpha_2K_{12} + y_2\alpha_2K_{22} - y_2\alpha_2K_{12} \\
&= y_2\alpha_2\kappa + \big(f(\mathbf{x}_1) - y_1\big) - \big(f(\mathbf{x}_2) - y_2\big).
\end{aligned} \tag{2.61}$$

So we have

$$\alpha_2^{(\text{new})} = \alpha_2^{(\text{old})} + y_2 \frac{E_2^{(\text{old})} - E_1^{(\text{old})}}{\kappa}, \tag{2.62}$$

where

$$E_i \equiv f(\mathbf{x}_i) - y_i = \left(\sum_{j=1}^{\ell} \alpha_j y_j K_{ij} + b \right) - y_i, \quad i = 1, 2, \cdots, n \tag{2.63}$$

where $f(\mathbf{x})$ denotes the current hypothesis determined by the value of α and b at a particular stage of learning, and E_i is the difference between the function output and the target classification on the training point $\mathbf{x}_1$ or $\mathbf{x}_2$. Note that E_i can be large if a point is correctly classified.

Finally, we must clip $\alpha_2^{(\text{new,unc})}$ if necessary to ensure it remains in the interval $[U, V]$.

4 SVM vs Perceptron

Figure 2.4 shows the decision boundaries generated by an SVM and a perceptron on a linearly separable dataset. The SVM boundary maximizes the margin between the two classes, while the perceptron simply finds a separating hyperplane.

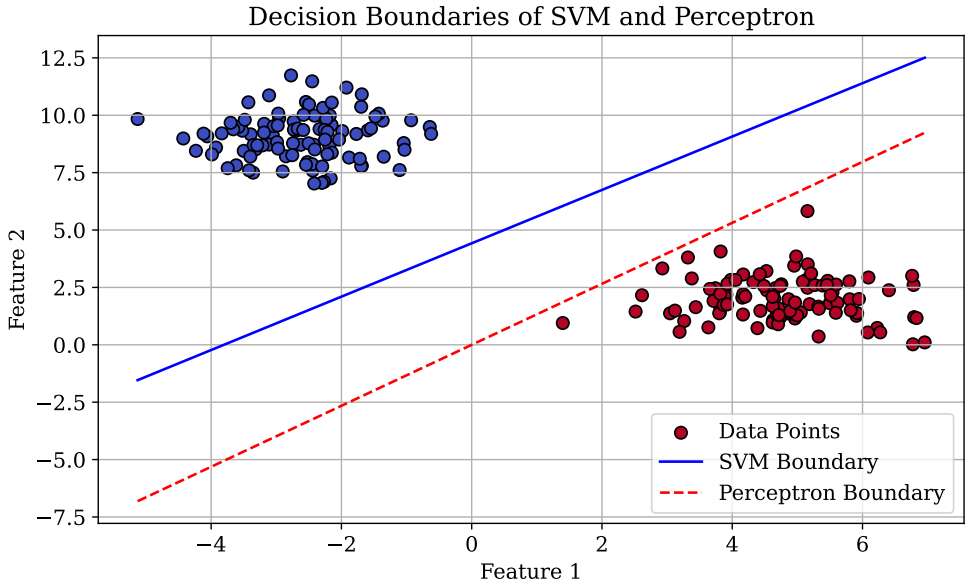


Figure 2.4: Decision boundaries of SVM (solid blue line) and Perceptron (dashed red line) on a linearly separable dataset.

Figure 2.5 shows the decision boundary of an SVM with an RBF kernel applied to non-linearly separable data. The decision boundary effectively separates the two classes. This demonstrates that SVMs can be used to solve non-linearly separable problems.

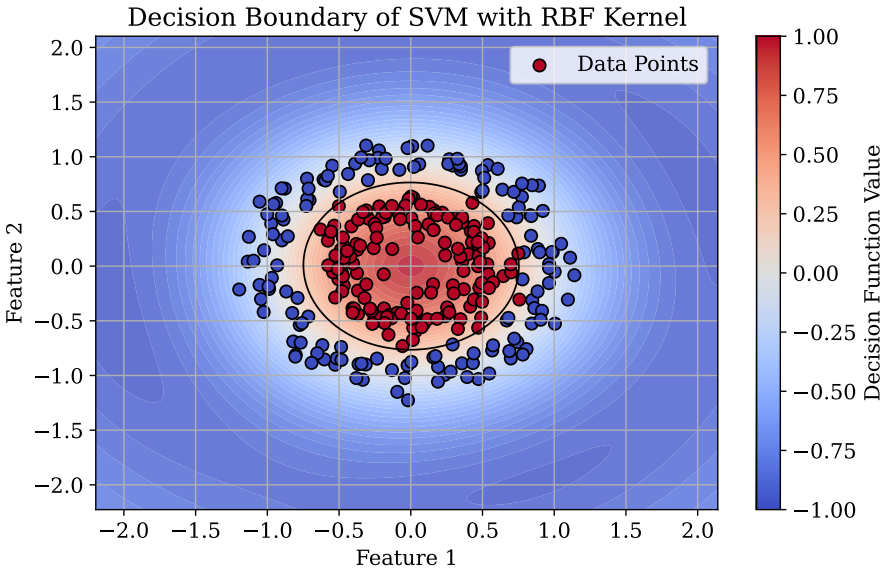


Figure 2.5: Decision boundary of SVM with RBF kernel. The black solid line represents the decision boundary, while the background colors indicate different classification regions.

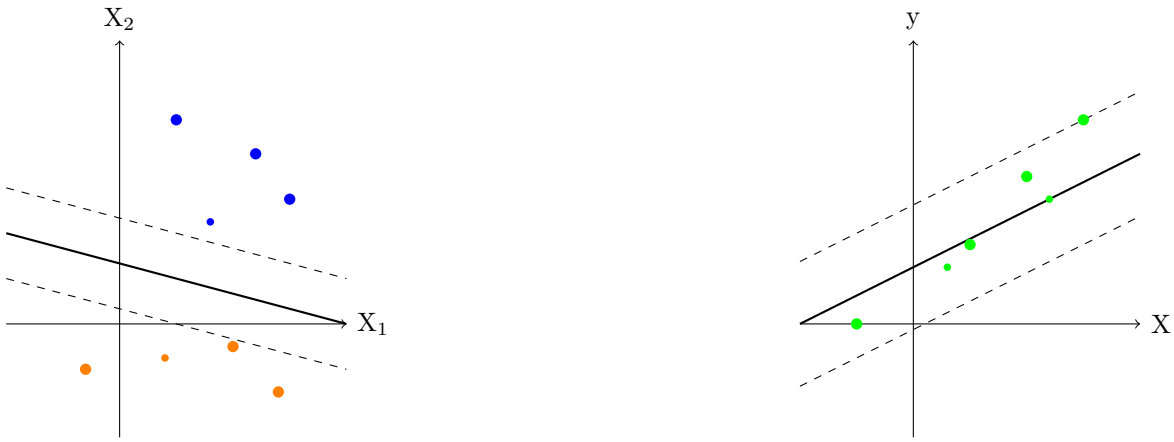
5 Support Vector Regression

Support Vector Machines can also be applied to solve regression problems. As shown in Fig. 2.6, unlike classification, our goal in regression is to predict vectors that stay as close as possible to points $(\mathbf{x}_i, y_i)$ for $i = 1, 2, \cdots, n$.

Similar to the aforementioned classification problem, the regression problem leads to the following optimization formulation.

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi},\boldsymbol{\xi}^*} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*), \\ \text{s.t.} \quad & y_i - (\mathbf{w}^\top \mathbf{x}_i + b) \leq \epsilon + \xi_i, \\ & (\mathbf{w}^\top \mathbf{x}_i + b) - y_i \leq \epsilon + \xi_i^*, \\ & \xi_i, \xi_i^* \geq 0, \quad i = 1, \ldots, n. \end{aligned} \tag{2.64}$$

We proceed to construct the Lagrangian by combining the objective function with the



Classification problem using SVM

Regression problem using SVR

Figure 2.6: Comparison of Support Vector Classification and Support Vector Regression (SVR). The left panel illustrates the classification problem solved using SVM, where the decision boundary maximizes the margin between two classes. The right panel shows the regression problem solved using SVR, where the model fits a regression line within an ε-insensitive tube.

corresponding constraints, introducing a set of dual variables.

$$\begin{aligned}\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\xi}^*, \boldsymbol{\alpha}, \boldsymbol{\alpha}^*, \boldsymbol{\eta}, \boldsymbol{\eta}^*) := \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \\ & - \sum_{i=1}^{n}\eta_i\xi_i - \sum_{i=1}^{n}\eta_i^*\xi_i^* \\ & - \sum_{i=1}^{n}\alpha_i(\varepsilon + \xi_i - y_i + \mathbf{w}^\top\mathbf{x}_i + b) \\ & - \sum_{i=1}^{n}\alpha_i^*(\varepsilon + \xi_i^* + y_i - \mathbf{w}^\top\mathbf{x}_i - b), \\ \text{s.t.} \quad \xi_i \geq 0, \quad \xi_i^* \geq 0, \quad \alpha_i \geq 0, \quad & \alpha_i^* \geq 0, \quad \eta_i \geq 0, \quad \eta_i^* \geq 0, \quad i = 1, \ldots, n.\end{aligned} \tag{2.65}$$

It follows from the saddle point condition that the partial derivatives of $\mathcal{L}$ with respect to the primal variables ($\mathbf{w}$, b, ξ_i, ξ_i^*) must vanish for optimality:

$$\frac{\partial\mathcal{L}}{\partial b} = \sum_{i=1}^{n}(\alpha_i^* - \alpha_i) = 0, \tag{2.66}$$

$$\frac{\partial\mathcal{L}}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\mathbf{x}_i = 0, \tag{2.67}$$

$$\frac{\partial\mathcal{L}}{\partial \xi_i} = C - \alpha_i - \eta_i = 0, \quad \frac{\partial\mathcal{L}}{\partial \xi_i^*} = C - \alpha_i^* - \eta_i^* = 0, \quad i = 1, \ldots, n. \tag{2.68}$$

Substituting Equation (2.66), (2.67), and (2.68) into (2.64), and adding kernel term yields the dual optimization problem.

$$\begin{aligned} \max_{\alpha_i, \alpha_i^*} \quad & \left\{ -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) K(\mathbf{x}_i, \mathbf{x}_j) - \epsilon \sum_{i=1}^{n} (\alpha_i + \alpha_i^*) + \sum_{i=1}^{n} y_i (\alpha_i - \alpha_i^*) \right\}, \\ \text{s.t.} \quad & \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) = 0, \quad 0 \leq \alpha_i, \alpha_i^* \leq C, \quad i = 1, 2, \ldots, n. \end{aligned} \tag{2.69}$$

An example of SVR-fitted curves is shown in Fig. 2.7.

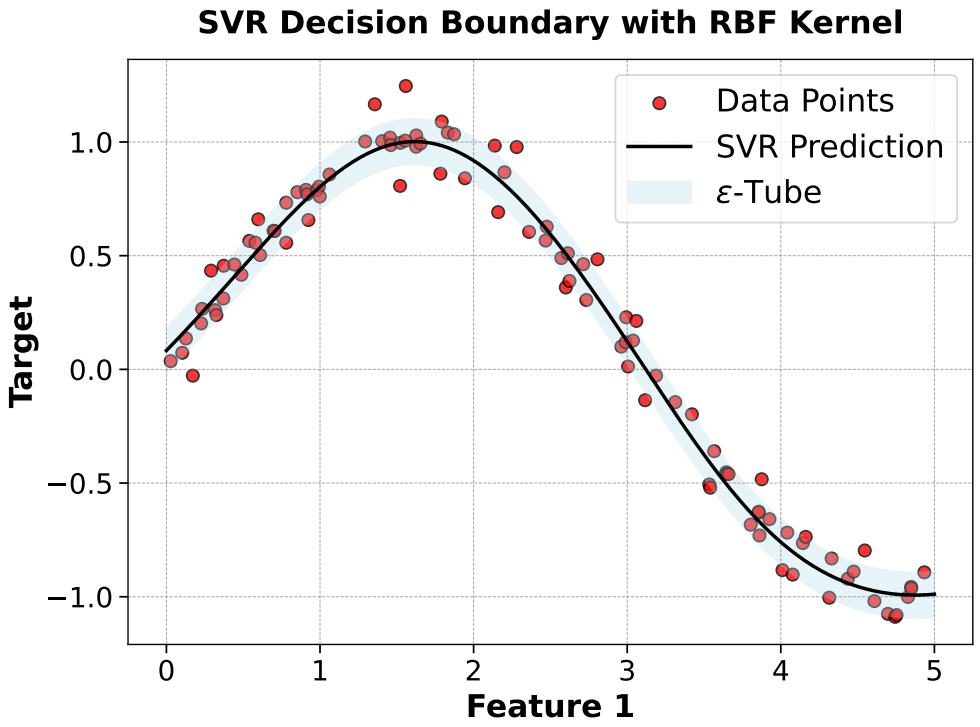


Figure 2.7: Decision boundary of SVR with RBF kernel.

CHAPTER 3

TREE-BASED MACHINE LEARNING

1 Classification Tree

Consider the classification problem. Fig. 3.1 illustrates a set of points in a plane, each marked with either a cross or a circle. The goal of a classifier is to partition the plane into smaller regions and assign a classification label—a cross or a circle—to each region. This partitioning should be performed in a way that minimizes the classification error for similar, unseen data.

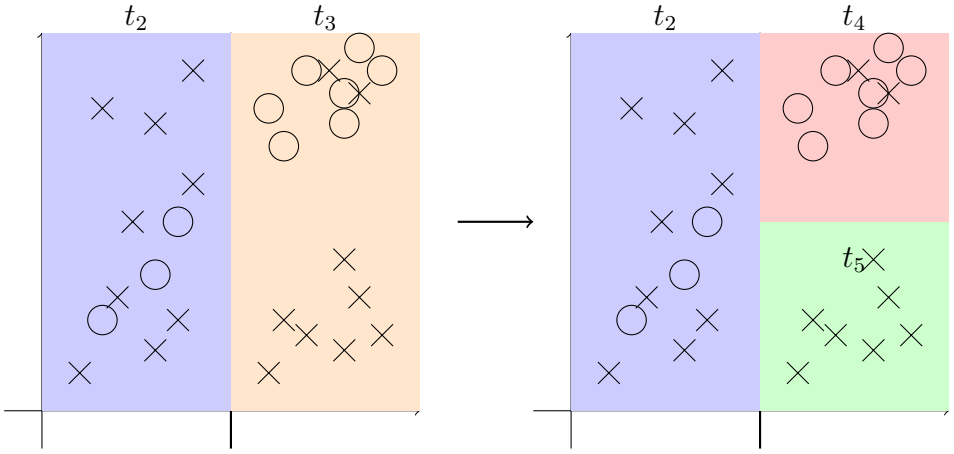


Figure 3.1: Illustration of a CART decision process. The first diagram (left) shows the split of t_2 and t_3 using a vertical line. The second diagram (right) illustrates the further split of t_3 into t_4 and t_5 based on horizontal and vertical lines. Different regions are shaded with distinct colors for clarity.

Now, consider the vertical line shown in Fig. 3.1. This line divides the plane into two regions: the left region, labeled t_2, and the right region, labeled t_3. The region t_3 is then further divided into two subregions, t_4 and t_5. Suppose we stop splitting here. The three regions corresponding to the terminal nodes t_2, t_4, and t_5 then constitute the partition of the entire plane. Our task now is to assign a class label to each of these partitioned regions. Upon inspection, the region t_2 contains more crosses than circles. Thus, by majority vote, the entire region t_2 is assigned the class label "cross." Similarly, the region t_5 is assigned the

class label "cross." On the other hand, the region t_4 contains more circles than crosses, so it is assigned the class label "circle." In this way, every point in the plane is assigned a class label.

In the future, given any point p, which is not necessarily one of the data points we have been dealing with, its class label will be determined by majority vote within the region it belongs to. For example, if p is in region t_4, it will be assigned the label "circle," which is the class label of region t_4. Similarly, if p is in region t_2, its label will be "cross." In this way, our tree functions as a classifier.

1.1 Mathematical Definition

The dataset is given as $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$, where $\mathbf{x}^{(i)} \in \mathcal{X}$ is a vector representing the input, and $y^{(i)} \in \mathcal{Y}$, where $\mathcal{Y} = \{1, \ldots, K\}$ is the set of K class labels. We define the following empirical quantities:

$$N = |\mathcal{D}| : \text{ the total number of data points,}$$

$$N_j = \left|\{(\mathbf{x}^{(i)}, y^{(i)}) \in \mathcal{D} : y^{(i)} = j\}\right| : \text{ the number of data points with class label } j \in \mathcal{Y}.$$

Let t denote a node, which is also identified with a subregion of $\mathcal{X}$ that it represents. At node t, we define the following:

$$\mathcal{D}(t) = \{(\mathbf{x}^{(i)}, y^{(i)}) \in \mathcal{D} : (\mathbf{x}^{(i)}, y^{(i)}) \in t\} : \text{ the set of data points in node } t,$$

$$N(t) = |\mathcal{D}(t)| : \text{ the number of data points in node } t,$$

$$N_j(t) = \left|\{(\mathbf{x}^{(i)}, y^{(i)}) \in \mathcal{D}(t) : y^{(i)} = j\}\right| : \text{ the number of data points in node } t \text{ with class label } j \in \mathcal{Y}.$$

If one randomly chooses a data point from $\mathcal{D}$, the probability of picking a data point with class label j would normally be N_j/N, assuming every point has an equal chance. However, in some cases, this assumption of "equal chance" may not be desirable. To account for this, we introduce $\boldsymbol{\pi} = (\pi(1), \ldots, \pi(K))$, the prior probability distribution of Y, which represents the probability of picking class label j.

Now, define:

$$p(j, t) = \pi(j)\frac{N_j(t)}{N_j}.$$

This is the probability of picking a data point that is in t with the class label j. (Note that if $\pi(j) = N_j/N$, then $p(j, t) = N_j(t)/N$.)

Define:

$$p(t) = \sum_j p(j, t). \tag{3.1}$$

This is the probability of picking a data point in the node t. We also define:

$$p(j|t) = \frac{p(j, t)}{p(t)}, \tag{3.2}$$

which is the conditional probability of picking a data point with the class label j when the node t is given. Note that if $\pi(j) = \frac{N_j}{N}$, then: $p(t) = \sum_j \frac{N_j(t)}{N} = \frac{N(t)}{N}$, and hence:

$p(j|t) = \frac{N_j(t)}{N(t)}$. As mentioned above, CART has a particular way of creating a tree. First, it always splits along a single feature (variable) of the input. Recall that the input vector $\mathbf{x}$ is of the form: $\mathbf{x} = (x_1, \ldots, x_d)$, where each x_i can be a numeric, categorical (nominal), or discrete ordinal variable. At each node, CART always chooses one of these variables and splits the node based on the values of that variable only.

Another particular way CART performs splitting is that it always creates two child nodes—hence the binary splitting—which results in a binary tree. Of course, general tree-based methods can split nodes into multiple (more than two) child nodes. However, for CART, the split is always binary. The reason for this is simplicity, as a single multiple split can equivalently be achieved by several successive binary splits.

1.2 Impurity and Splitting

Definition 1.1: Impurity Function

An impurity function ϕ is a function $\phi(p_1, \cdots, p_K)$ defined for $p_1, \cdots, p_K$ with $p_i \geq 0$ for all i and $p_1 + \cdots + p_K = 1$ such that:

1. $\phi(p_1, \cdots, p_K) \geq 0$,
2. $\phi\left(\frac{1}{K}, \cdots, \frac{1}{K}\right)$ is the maximum value of ϕ,
3. $\phi(p_1, \cdots, p_K)$ is symmetric with regard to $p_1, \cdots, p_K$,
4. $\phi(1, 0, \cdots, 0) = \phi(0, 1, \cdots, 0) = \phi(0, \cdots, 0, 1) = 0$.

The following are examples of impurity functions:

Example :

1. ***Entropy impurity***

$$\phi(p_1, \cdots, p_K) = -\sum_i p_i \log p_i,$$

where we use the convention $0 \log 0 = 0$***.***

2. ***Gini impurity***

$$\phi(p_1, \cdots, p_K) = \sum_i p_i(1 - p_i) = \sum_{i<j} p_i p_j.$$

Definition 1.2: Impurity of a Node

For a node t, its impurity (measure) $i(t)$ is defined as:

$$i(t) = \phi\big(p(1|t), \cdots, p(K|t)\big).$$

Given a node t, there are many possible ways of performing a binary split. A general principle is to choose the split that decreases the impurity the most. This is easy to understand because when the impurity of a region is zero, **it means that all data points in that region belong to the same class** ($\phi(1, 0, \cdots, 0) = \phi(0, 1, \cdots, 0) = \phi(0, \cdots, 0, 1) = 0$).

Definition 1.3: Decrease in Impurity

Let t be a node and let s be a split of t into two child nodes t_L and t_R. The decrease in impurity is defined as:

$$\Delta i(s, t) = i(t) - p_L i(t_L) - p_R i(t_R).$$

Definition 1.4: Terminal Nodes of a Tree

Let T be a tree. We denote by $\tilde{T}$ the set of terminal (leaf) nodes of T.

Definition 1.5: Impurity of a Node and a Tree

The impurity of a node t is defined as:

$$I(t) = i(t)p(t),$$

and the impurity of the tree T is defined as:

$$I(T) = \sum_{t \in \tilde{T}} I(t) = \sum_{t \in \tilde{T}} i(t)p(t).$$

Let us now examine how a split affects the tree impurity. The left side of Fig. 3.2 shows a tree T, where the terminal nodes are labeled as t and t'. The right side depicts a new tree T', which is obtained by splitting the terminal node t into two child nodes t_L and t_R, while the rest of the terminal nodes remain unchanged.

It is straightforward to observe that:

$$\begin{aligned} I(T) &= I(t) + \sum_{t' \in \tilde{T} \setminus \{t\}} I(t'), \\ I(T') &= I(t_L) + I(t_R) + \sum_{t' \in \tilde{T} \setminus \{t\}} I(t'), \end{aligned} \tag{3.3}$$

where $\tilde{T}$ denotes the set of terminal (leaf) nodes of tree T.

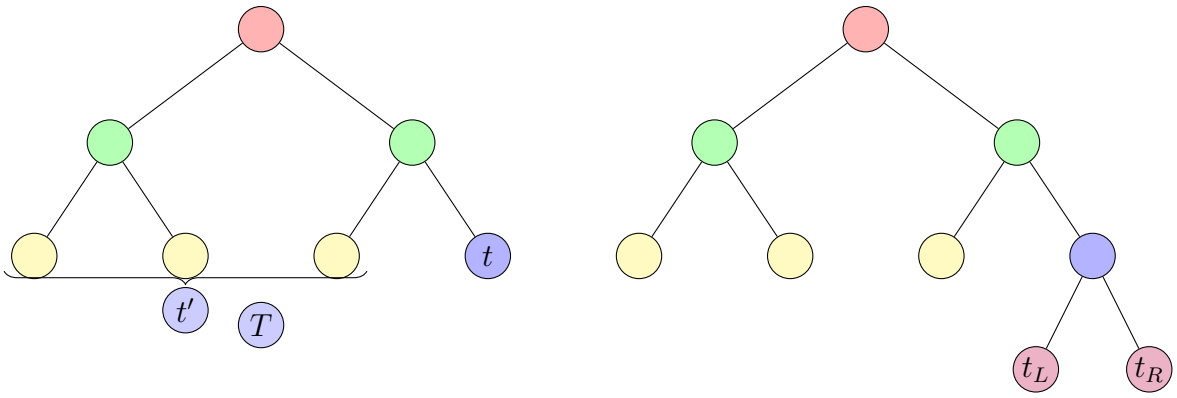


Figure 3.2: Tree Splitting: The left tree T shows terminal nodes t and t'. The right tree T' is formed by splitting t into child nodes t_L and t_R, while t' remains unchanged.

Thus, the change in the total impurity from T to T' is given by:

$$I(T) - I(T') = I(t) - I(t_L) - I(t_R), \tag{3.4}$$

which inspires the following definition.

Definition 1.6: The Total Impurity Change

Let t be a node and let s be a split of t into two child nodes t_L and t_R. Then the total impurity change due to s is defined as:

$$\Delta I(s,t) = I(t) - I(t_L) - I(t_R).$$

Note that:

$$\Delta I(s,t) = i(t)p(t) - i(t_L)p(t_L) - i(t_R)p(t_R) = \big\{i(t) - p_L i(t_L) - p_R i(t_R)\big\}p(t) = \Delta i(s,t)p(t).$$

Thus, the split s that maximizes $\Delta I(s,t)$ also maximizes $\Delta i(s,t)$. Therefore, we can use either node impurity measure to achieve the same result.

1.3 Determining Node Class Label

A tree-based classifier assigns class labels to subregions of $\mathcal{X}$, formed by terminal nodes, using a weighted majority vote guided by a cost function.

Definition 1.7: Cost of Misclassifying Class

Let $i, j \in \mathcal{Y}$ be labels. A cost $C(i|j)$ of misclassifying class j as class i is defined as a nonnegative number satisfying $C(j|j) = 0$ for all $j \in \mathcal{Y}$.

Now, suppose our classifier predicts the label of a node t as i, but the true label is unknown. In this case, the expected cost of declaring the label i is calculated as the weighted sum of

costs, where the weights are the probabilities $p(j|t)$. Mathematically, this is expressed as:

$$\sum_j C(i|j)p(j|t). \tag{3.5}$$

In particular, when $C(i|j) = 1 - \delta_{ij}$, where δ_{ij} is the Kronecker delta defined as:

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j, \end{cases} \tag{3.6}$$

the expected cost of misclassification is easily computed as:

$$\sum_j (1 - \delta_{ij})p(j|t) = 1 - p(i|t). \tag{3.7}$$

Definition 1.8: Class Label

Let t be a node. The class label $j^*(t)$ of t is defined as:

$$j^*(t) = \arg\min_i \sum_j C(i|j)p(j|t).$$

In case there is a tie, the tie is broken using some arbitrary rule.

In the case where $C(i|j) = 1 - \delta_{ij}$, it is easily seen from Equation (3.7) that:

$$\arg\min_i \sum_j C(i|j)p(j|t) = \arg\max_i p(i|t). \tag{3.8}$$

Assume further that $\pi(i) = \frac{N_i}{N}$. Then, we have $p(i|t) = \frac{N_i(t)}{N(t)}$. Thus, $j^*(t)$ is reduced to:

$$j^*(t) = \arg\max_i N_i(t). \tag{3.9}$$

Namely, in this case, the class label $j^*(t)$ of t is determined by the simple majority vote rule.

1.4 Misclassification Cost

A goal of the classification tree is to minimize misclassification. A simplistic way of looking at this is to count the number of misclassified cases and calculate the ratio. But a more general way is to utilize the misclassification cost defined above. For that, we need the following definitions.

Definition 1.9: Misclassification Cost

Define the misclassification cost $r(t)$ of a node t as:

$$r(t) = \min_i \sum_j C(i|j)p(j|t).$$

Let $R(t) = r(t)p(t)$. Using this, the misclassification cost $R(T)$ of a tree T is defined as:

$$R(T) = \sum_{t \in \tilde{T}} R(t) = \sum_{t \in \tilde{T}} r(t)p(t),$$

where $\tilde{T}$ denotes the set of terminal (leaf) nodes of T.

Proposition 1.10

Let a node t be split into t_L and t_R. Then:

$$R(t) \geq R(t_L) + R(t_R),$$

and equality holds if $j^*(t) = j^*(t_L) = j^*(t_R)$.

Proof. By Definition,

$$r(t) = \sum_j C(j^*(t)|j)p(j|t).$$

Thus,

$$\begin{aligned} R(t) = r(t)p(t) &= \sum_j C(j^*(t)|j)p(j|t)p(t) \\ &= \sum_j C(j^*(t)|j)p(j,t) \\ &= \sum_j C(j^*(t)|j)\big\{p(j,t_L) + p(j,t_R)\big\}. \end{aligned}$$

where the last equality is due to the fact that $N_j(t) = N_j(t_L) + N_j(t_R)$. Note that for any node t, we have:

$$R(t) = r(t)p(t) = \min_i \sum_j C(i|j)p(j,t).$$

Therefore, we have:

$$\begin{aligned} R(t) - R(t_L) - R(t_R) = &\sum_j C(j^*(t)|j)p(j,t_L) - \min_i \sum_j C(i|j)p(j,t_L) \\ &+ \sum_j C(j^*(t)|j)p(j,t_R) - \min_i \sum_j C(i|j)p(j,t_R) \geq 0. \end{aligned}$$

It is trivial to see that if $j^*(t) = j^*(t_L) = j^*(t_R)$, then the equality holds.

Through this conclusion, we can see that continuously dividing the space by CART, even randomly, can reduce the classification cost on the training set.

1.5 Weakest-link Cutting & Minimal Cost-complexity Pruning

We may eventually end up with pure terminal nodes that contain only a single class. In some cases, these terminal nodes may even consist of just one data point, which is a clear indication of overfitting. This raises the question: when and how should we stop splitting, and how should we prune back if the tree has been over-split? A simple rule of thumb is to stop splitting a node if the number of elements in the node falls below a preset threshold. However, the larger question remains: "*What is the right-sized tree, and how can we find a tree that generalizes well?*" This is the key issue we aim to address here. One way to mitigate this problem is to penalize the tree for having too many terminal nodes.

Definition 1.11: Regularization Cost

Let $\alpha \geq 0$. For any node t, not necessarily a terminal node, define:

$$R_\alpha(t) = R(t) + \alpha.$$

For a tree T, define:

$$R_\alpha(T) = R(T) + \alpha|\tilde{T}|,$$

where $|\tilde{T}|$ is the number of terminal elements in T.

If we use $R_\alpha(t)$ as a new measure for the (augmented) misclassification cost, a split does not necessarily decrease $R_\alpha(t)$, because it increases $\alpha|\tilde{T}|$.

Definition 1.12: Pruned Subtree

A pruned subtree T_1 of a tree T is a subtree obtained by the following process:

1. Take a node t of T,
2. Remove all descendants of t from T, but not t itself.

If T_1 is a pruned subtree of T, we write $T_1 \preceq T$. Note in particular that T_1 always contains the root node of T.
One should note the subtlety of this definition: a pruned subtree always has the same root node as the original tree, while a subtree can be any part of the original tree and may not necessarily contain the root node.

Pruning ensures that the totality of the subregions corresponding to the terminal nodes of the pruned subtree $T - T_{t_2}$ still forms a partition of X. This is because, even when the descendants of a node t_2 are removed, the node t_2 itself remains intact in the pruned subtree. By keeping t_2, the structural integrity of the tree is preserved, allowing the terminal nodes of the pruned subtree to continue representing mutually disjoint subregions of X, which is

critical for maintaining the classification or regression functionality of the tree.

Suppose we have completed all the splitting and grown a large tree, denoted by $T_{\max}$. Our goal is to find a subtree of $T_{\max}$ that minimizes R_α. However, there may be more than one such subtree. To resolve this ambiguity, we introduce the following definition.

Definition 1.13: Optimally Pruned Subtree

Given a tree T and $\alpha \geq 0$, the optimally pruned subtree $T(\alpha)$ of T is defined by the following conditions:

1. $T(\alpha) \leq T$,
2. $R_\alpha(T(\alpha)) \leq R_\alpha(T')$ for any pruned subtree $T' \preceq T$,
3. If a pruned subtree $T' \preceq T$ satisfies the condition $R_\alpha(T(\alpha)) = R_\alpha(T')$, then $T(\alpha) \preceq T'$.

Proposition 1.14

$T(\alpha)$ exists and is unique.

Proof. Let T be a given tree with root node t_1. We use induction on $n = |\tilde{T}|$, the number of terminal nodes. If $n = 1$, then $T = \{t_1\}$, and there is nothing to prove. Assume $n > 1$. In this case, t_1 must have child nodes t_L and t_R. Let T_L be the subtree of T with t_L as its root node, including all its descendants. Similarly, define T_R with t_R as its root. Thus, $T_L = T_{t_L}$ and $T_R = T_{t_R}$. These subtrees, known as the primary branches of T, are not pruned subtrees of T because they do not include the root node of the original tree.

Let T' be a nontrivial pruned subtree of T (i.e., a subtree with more than one node). Thus, T' must contain the nodes t_1, t_L, and t_R of T. Let $T'_L = T'_{t_L}$ and $T'_R = T'_{t_R}$ be the primary branches of T'. Clearly, T'_L is a pruned subtree of T_L with the same root node t_L. Similarly, T'_R is a pruned subtree of T_R with the same root node t_R. (Note that T'_L and T'_R are not pruned subtrees of T.). By the induction hypothesis, there exist optimally pruned subtrees $T_L(\alpha)$ and $T_R(\alpha)$ of T_L and T_R, respectively. There are two cases to consider.

The first case to consider is when:

$$R_\alpha(t_1) \leq R_\alpha(T_L(\alpha)) + R_\alpha(T_R(\alpha)).$$

By the definition of $T_L(\alpha)$, we have $R_\alpha(T_L(\alpha)) \leq R_\alpha(T'_L)$, and similarly, $R_\alpha(T_R(\alpha)) \leq R_\alpha(T'_R)$. Since $R_\alpha(T') = R_\alpha(T'_L) + R_\alpha(T'_R)$, combining this with the above equation, we obtain:

$$R_\alpha(t_1) \leq R_\alpha(T').$$

Since this inequality holds for any pruned subtree T' of T, we conclude that:

$$T(\alpha) = \{t_1\}.$$

The remaining case is when:

$$R_\alpha(t_1) > R_\alpha(T_L(\alpha)) + R_\alpha(T_R(\alpha)).$$

Now, create a new tree T_1 by joining $T_L(\alpha)$ and $T_R(\alpha)$ to t_1. Clearly, T_1 is a pruned subtree of T, and $T_L(\alpha)$ and $T_R(\alpha)$ are the primary branches of T_1. It follows that:

$$R_\alpha(T_1) = R_\alpha(T_L(\alpha)) + R_\alpha(T_R(\alpha)).$$

$T(\alpha)$, if it exists, cannot be $\{t_1\}$. Now, let T' be any nontrivial pruned subtree of T. By the definition of $T_L(\alpha)$, we have:

$$R_\alpha(T_L(\alpha)) \leq R_\alpha(T'_L),$$

and similarly:

$$R_\alpha(T_R(\alpha)) \leq R_\alpha(T'_R).$$

Since $R_\alpha(T') = R_\alpha(T'_L) + R_\alpha(T'_R)$, we must have:

$$R_\alpha(T_1) \leq R_\alpha(T').$$

Therefore, T_1 is the pruned subtree of T that has the smallest R_α-value among all pruned subtrees of T.

It remains to show that any strictly smaller pruned subtree of T_1 has a larger R_α-value. Let T' be a strictly smaller pruned subtree of T_1. Without loss of generality, we may assume $T'_L < T_L(\alpha)$ and $T'_R \leq T_R(\alpha)$.

By the definition of $T_L(\alpha)$, we have: $R_\alpha(T_L(\alpha)) < R_\alpha(T'_L)$, and similarly: $R_\alpha(T_R(\alpha)) \leq R_\alpha(T'_R)$. Using the same argument as above, we obtain: $R_\alpha(T_1) < R_\alpha(T')$.

Therefore, we can conclude that:

$$T_1 = T(\alpha).$$

The next step is to find $T(\alpha)$. However, the challenge lies in the fact that there are so many possibilities for α. Let us now explore how to determine such a "good" α. By the continuity argument, it is easily implied that for α sufficiently small:

$$R_\alpha(T_t) < R_\alpha(t). \tag{3.10}$$

At any node $t \in T(0)$, in that case, we would be better off not pruning. Note that:

$$\begin{aligned} R_\alpha(t) &= R(t) + \alpha, \\ R_\alpha(T_t) &= R(T_t) + \alpha|\tilde{T}_t|. \end{aligned} \tag{3.11}$$

Therefore, Equation (3.10) is equivalent to saying that:

$$\alpha < \frac{R(t) - R(T_t)}{|\tilde{T}_t| - 1}. \tag{3.12}$$

But as α gets larger, there appears a node (or nodes) at which Equation (3.10) is no longer valid. To find such node(s), let us define a function $g_1(t)$ of node t by:

$$g_1(t) = \begin{cases} \frac{R(t)-R(T_t)}{|\tilde{T}_t|-1}, & \text{if } t \notin \tilde{T}_1, \\ \infty, & \text{otherwise.} \end{cases} \tag{3.13}$$

Define

$$\alpha_2 = \min_{t \in T_1} g_1(t). \tag{3.14}$$

So for $\alpha < \alpha_2$, there is no node at which Equation (3.12) is violated. But for $\alpha = \alpha_2$, there is at least one node at which the inequality (3.12) becomes an equality. Such nodes are called weakest-link nodes. Suppose t_1^0 is such a node. Then,

$$g_1(t_1^0) = \alpha_2 = \min_{t \in T_1} g_1(t). \tag{3.15}$$

We then prune T_1 at t_1^0 to come up with a pruned subtree $T_1 - T_{t_1^0}$. Note that

$$R_{\alpha_2}(T_{t_1^0}) = R_{\alpha_2}(t_1^0). \tag{3.16}$$

Therefore,

$$R_{\alpha_2}(T_1) = R_{\alpha_2}(T_1 - T_{t_1^0}). \tag{3.17}$$

There may be more than one such weakest-link node. Suppose there are two such nodes. If one of them is an ancestor of the other, prune at the ancestor (then its descendants are all gone). If neither one is an ancestor of the other, prune at both. Keep doing this for all such weakest-link nodes. This way, we get a new subtree T_2, and it is not hard to see that

$$T_2 = T(\alpha_2). \tag{3.18}$$

Using T_2 in lieu of T_1, we can repeat the same process to come up with α_3 and $T_3 = T(\alpha_3)$. Repeat this process until we arrive at the subtree with the root node only. Namely, we get the following sequence of pruned subtrees:

$$T_1 \geq T_2 \geq \cdots \geq \{t_1\}, \tag{3.19}$$

where $T_k = T(\alpha_k)$ and $0 = \alpha_1 < \alpha_2 < \cdots$. We need the following proposition for cross-validation.

Proposition 1.15

i) If $\alpha_1 \leq \alpha_2$, then $T(\alpha_2) \leq T(\alpha_1)$. (ii) For α with $\alpha_k \leq \alpha < \alpha_{k+1}$, $T(\alpha_k) = T(\alpha)$.

Proof. Recall that $T(\alpha)$ is either $\{t_1\}$ or the join of $T_L(\alpha)$ and $T_R(\alpha)$ at t_1. Suppose $T(\alpha_1) = \{t_1\}$, i.e.,

$$R_\alpha(t_1) \leq R_\alpha(T_0),$$

for any pruned subtree T_0 of T. Then,

$$\begin{aligned} R_{\alpha_2}(t_1) &= R_{\alpha_1}(t_1) + \alpha_2 - \alpha_1 \\ &\leq R_{\alpha_1}(T_0) + \alpha_2 - \alpha_1 \\ &\leq R_{\alpha_1}(T_0) + (\alpha_2 - \alpha_1)|T_0| \\ &= R_{\alpha_2}(T_0), \end{aligned}$$

Therefore, we can easily conclude that $T(\alpha_2) = \{t_1\}$.

Assume now $T(\alpha_1)$ is nontrivial and is the join of $T_L(\alpha_1)$ and $T_R(\alpha_1)$ at t_1. By the induction hypothesis, we may assume that $T_L(\alpha_2) \leq T_L(\alpha_1)$ and $T_R(\alpha_2) \leq T_R(\alpha_1)$. Thus, if $T(\alpha_2)$ is the join of $T_L(\alpha_2)$ and $T_R(\alpha_2)$ at t_1, the proof of (i) is complete. On the other hand, if $T(\alpha_2) = \{t_1\}$, then again it is obvious that $\{t_1\}$ is a pruned subtree of anything.

Proof of (ii): Suppose the assertion of (ii) is false. Then (i) says that $T(\alpha) < T(\alpha_k)$. Therefore, the pruning of $T(\alpha_k)$ must have occurred at some node, say, at $t_0 \in T(\alpha_k)$. Let $T(\alpha_k)^{t_0}$ be the branch of $T(\alpha_k)$ at t_0. Namely, $T(\alpha_k)^{t_0}$ is the subtree of $T(\alpha_k)$ with t_0 as its root node together with all of its descendant nodes in $T(\alpha_k)$. Since the pruning has occurred at t_0, we must have

$$R_\alpha(t_0) \leq R_\alpha(T(\alpha_k)^{t_0}).$$

It is then not hard to see that $\alpha_{k+1} \leq \alpha$. This is a contradiction to the assumption of (ii).

From Fig. 3.3, we can clearly observe the differences in decision boundaries produced by the three classification methods we have studied so far.

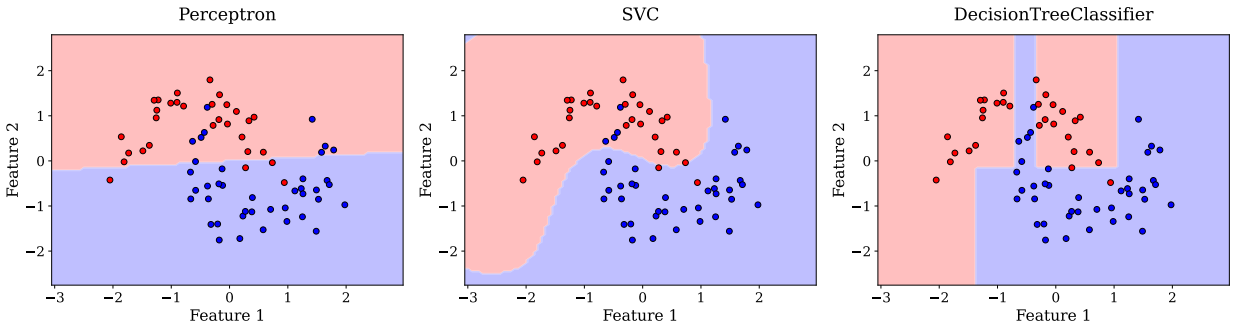


Figure 3.3: Decision boundaries of three classifiers on a non-linear dataset. The dataset consists of two interleaving moons with added noise, making it non-linearly separable. The three subplots show the decision boundaries created by the Perceptron (a linear classifier), RBF SVM (a kernel-based support vector machine with an RBF kernel), and CART (a decision tree classifier), respectively. The red and blue shaded regions represent the predicted classes, and the data points are marked by circles. Note how RBF SVM and CART can create non-linear boundaries, whereas Perceptron only learns a linear boundary.

2 Regression Tree

Now, let us look at the regression tree. Fig. 3.4 shows points in the xy-plane. The regression problem is to find a good function $y = f(x)$ whose graph lies close to the given data points in Fig. 3.4. The method of the CART regression tree is similar to that of the CART classification tree in that the entire region of the x-axis is partitioned into subregions, and the partitioning pattern is encoded in a tree data structure, called the regression tree. The only difference here is that, instead of taking the majority vote as was done for the classification tree, the average y-value of the data is used as the (constant) value of the regression function on each partitioned subregion.

The region is then partitioned into four subregions: t_4, t_5, t_6, and t_7. On each subregion, the average y-value, y_4, y_5, y_6, and y_7, is computed, respectively. In this way, the final

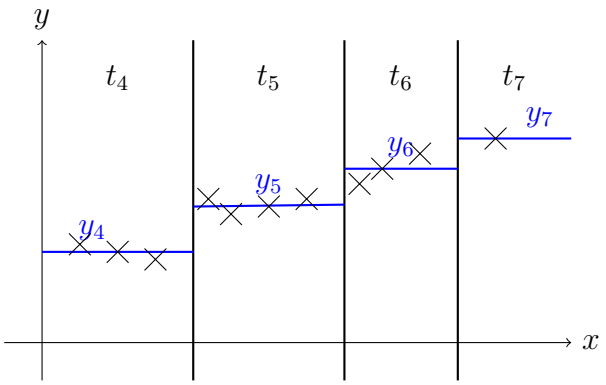


Figure 3.4: Illustration of a regression tree. The plane is divided into four regions (t_4, t_5, t_6, and t_7) using vertical split lines. Each region is assigned a predicted regression value (y_4, y_5, y_6, and y_7), shown as horizontal blue lines. The scatter points ($\times$) represent the data distribution.

regression function $y = f(x)$ becomes a piecewise constant function, with values y_i on t_i for $i = 4, 5, 6$, and 7.

2.1 Details of Regression Tree

We now consider the regression tree. Let the dataset be $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$, where $\mathbf{x}^{(i)} \in \mathcal{X}$ and $y^{(i)} \in \mathcal{Y}$. The goal of the regression tree algorithm is to construct a function $f : \mathbf{x}^{(i)} \to y^{(i)}$ such that the error

$$\sum_{i=1}^{N} \|f(\mathbf{x}^{(i)}) - y^{(i)}\|^2 \tag{3.20}$$

is minimized. This is achieved by constructing a tree and assigning a constant value to each subregion corresponding to the terminal nodes of the tree. Thus, the function f constructed in this manner is a piecewise constant function.

Recall the method used to construct a tree. Specifically, each node $t \in T$ corresponds to a subset of $\mathcal{X}$. At each node t, define the average y-value, $y(t)$, of the data at that node as follows:

$$\mathbf{y}(t) = \frac{1}{N(t)} \sum_{i:\mathbf{x}^{(i)} \in t} y^{(i)}, \tag{3.21}$$

where $N(t)$ denotes the number of data points in the subset associated with node t, and the sum is taken over all data points $\mathbf{x}^{(i)}$ that belong to the subset t.

We also define the (squared) error rate $r(t)$ of node t by:

$$r(t) = \frac{1}{N(t)} \sum_{i:\mathbf{x}^{(i)} \in t} (y^{(i)} - y(t))^2, \tag{3.22}$$

which is simply the variance of the data at node t and serves as an estimator of $\mathrm{Var}(y \mid \mathbf{x} \in t)$.

Next, we define the cost $R(t)$ of node t by:

$$R(t) = r(t)p(t), \tag{3.23}$$

where $p(t) = \frac{N(t)}{N}$ is the proportion of data points at node t relative to the total number of data points. Therefore, we have:

$$R(t) = \frac{1}{N} \sum_{i:\mathbf{x}^{(i)} \in t} (y^{(i)} - y(t))^2. \tag{3.24}$$

We now need the following proposition.

Proposition 2.1

Suppose a node t is split into t_L and t_R. Then,

$$R(t) = R(t_L) + R(t_R),$$

where the equality holds if and only if $y(t) = y(t_L) = y(t_R)$.

Proof.

$$\begin{aligned} R(t) &= \frac{1}{N} \sum_{i:\mathbf{x}^{(i)} \in t} (y^{(i)} - y(t))^2 \\ &= \frac{1}{N} \sum_{i:\mathbf{x}^{(i)} \in t_L} (y^{(i)} - y(t))^2 + \frac{1}{N} \sum_{i:\mathbf{x}^{(i)} \in t_R} (y^{(i)} - y(t))^2 \\ &\geq \frac{1}{N} \sum_{i:\mathbf{x}^{(i)} \in t_L} (y^{(i)} - y(t_L))^2 + \frac{1}{N} \sum_{i:\mathbf{x}^{(i)} \in t_R} (y^{(i)} - y(t_R))^2, \end{aligned}$$

where the last equality follows from the following Lemma. The statement on the equality also follows from the following Lemma.

Remark.

> We can observe that for regression problems, just like in classification, each split of the tree reduces the fitting error.

Lemma 2.2: Optimal Value of $y(t)$

For a given node t, the value $y(t)$ minimizes the following expression:

$$y(t) = \arg\min_a \sum_{i:\mathbf{x}^{(i)} \in t} (y^{(i)} - a)^2,$$

where the equality holds if and only if $a = y(t)$, for all $a \in \mathbb{R}$.

Proof.

$$\begin{aligned}\sum_{\mathbf{x}^{(i)}\in t}(y^{(i)}-a)^2 &= \sum_{\mathbf{x}^{(i)}\in t}\Big(y^{(i)}-y(t)+y(t)-a\Big)^2\\ &= \sum_{\mathbf{x}^{(i)}\in t}(y^{(i)}-y(t))^2 - 2(y(t)-a)\sum_{\mathbf{x}^{(i)}\in t}(y^{(i)}-y(t)) + N(t)(y(t)-a)^2\\ &= \sum_{\mathbf{x}^{(i)}\in t}(y^{(i)}-y(t))^2 + N(t)(y(t)-a)^2\\ &\geq \sum_{\mathbf{x}^{(i)}\in t}(y^{(i)}-y(t))^2\end{aligned}$$

The final step uses the fact that $N(t)(y(t)-a)^2 \geq 0$.

Clearly, equality holds if and only if $y(t)-a=0$, i.e., $a=y(t)$.

Let s be a split of a node t. Define the decrease $\Delta R(s,t)$ of the cost by s as

$$\Delta R(s,t) = R(t) - R(t_L) - R(t_R). \tag{3.25}$$

The splitting rule at t is s^* such that we take the split s^* among all possible candidate splits that decreases the cost the most. Namely,

$$\Delta R(s^*,t) = \max_s \Delta R(s,t). \tag{3.26}$$

In this manner, we can grow the regression tree to $T_{\max}$. As before, a simple rule of thumb is to stop splitting a node if the number of elements in the node is less than a predetermined threshold.