Mathematics, Optimization, Statistics, and Machine Learning

Sunghee Yun

March 22, 2020

Contents

Part I

Mathematics

Chapter 1

Algebra

1.1 Permutation

Chapter 2

Calculus

2.1 Basics

Theorem 2.1 (L'Hôpital's rule) *Let $f : \mathbf{R} \to \mathbf{R}$ and $f : \mathbf{R} \to \mathbf{R}$ be differentiable on an open interval $I \subseteq \mathbf{R}$ except possibly at $c \in I$. If $\lim_{x\to c} f(x) = \lim_{x\to c} g(x) = 0$ or $\pm\infty$, $g'(x) = 0$ for all $x \in I\backslash\{c\}$, and $\lim_{x\to c} \frac{f'(x)}{g'(x)}$ exists, then*

$$\lim_{x\to c} \frac{f(x)}{g(x)} = \lim_{x\to c} \frac{f'(x)}{g'(x)}. \tag{2.1}$$

Definition 2.1 (Taylor polynomial) *Let $n \in \mathbf{Z}$ be a positive integer and let $f : \mathbf{R} \to \mathbf{R}$ be n times differentiable at $a \in \mathbf{R}$. The n-th order Taylor polynomial is defined by*

$$\begin{aligned} T_{f,n}(x) &= f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n \\ &= \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!}(x-a)^k \end{aligned} \tag{2.2}$$

Theorem 2.2 (Taylor's theorem) *Let $n \in \mathbf{Z}$ be a positive integer and let $f : \mathbf{R} \to \mathbf{R}$ be n times differentiable at $a \in \mathbf{R}$. Then there exists a function $h_n : \mathbf{R} \to \mathbf{R}$ such that*

$$f(x) = T_{f,n}(x) + h_n(x)(x-a)^n \tag{2.3}$$

and $\lim_{x\to a} h_n(x) = 0$. The remainder is called the Peano form of the remainder.

Theorem 2.3 (Taylor's theorem) *Let $n \in \mathbf{Z}$ be a positive integer, $a, b \in \mathbf{R}$, and $I_o = (a,b) \cup (b,a)$ and $I_c = [a,b] \cup [b,a]$. Let $f : \mathbf{R} \to \mathbf{R}$ be $n+1$ times differentiable on I_o and $f^{(n)}$ is continuous on I_c. Then for some $c \in I_o$,*

$$f(b) = T_{f,n}(b) + \frac{f^{(n+1)}(c)}{(n+1)!}(b-a)^{n+1}. \tag{2.4}$$

The remainder is called the Peano form of the remainder.

2.2 Multivariate functions

Definition 2.2 (Jacobian matrix) *Let $f : \mathbf{R}^n \to \mathbf{R}^m$ be a differentiable function, i.e., the partial derivative $\partial f_j(x)/\partial x_i$ exists for every $1 \le i \le n$ and $1 \le j \le m$. Then the Jacobian matrix of f at x is defined by the function $D_f : \mathbf{R}^n \to \mathbf{R}^{m\times n}$ such that*

$$D_f(x) = \begin{bmatrix} \partial f_1(x)/\partial x_1 & \partial f_1(x)/\partial x_2 & \cdots & \partial f_1(x)/\partial x_n \\ \partial f_2(x)/\partial x_1 & \partial f_2(x)/\partial x_2 & \cdots & \partial f_2(x)/\partial x_n \\ \vdots & \vdots & \ddots & \vdots \\ \partial f_m(x)/\partial x_1 & \partial f_m(x)/\partial x_2 & \cdots & \partial f_m(x)/\partial x_n \end{bmatrix} \in \mathbf{R}^{m\times n}. \tag{2.5}$$

2.3 Chain rule

Theorem 2.4 *Let $f : \mathbf{R} \to \mathbf{R}^n$ and $g : \mathbf{R}^n \to \mathbf{R}$ be differentiable. Then $h : \mathbf{R} \to \mathbf{R}$ such that $h(t) = g(f(t))$ is also differentiable and*

$$h'(t) = \sum_{i=1}^{n} f_i'(t) \frac{\partial g}{\partial x_i}(f(t)) = \nabla g(f(t))^T D_f(t)$$

for all $t \in \mathbf{dom}\, f$.

Corollary 2.1 *Let $f : \mathbf{R}^n \to \mathbf{R}^m$ and $g : \mathbf{R}^m \to \mathbf{R}^p$ be differentiable. Then define a function $h : \mathbf{R}^n \to \mathbf{R}^p$ such that $h(x) = g(f(x))$ for all $x \in \mathbf{dom}\, f$. Then h is differentiable and*

$$Dh(x) = Dg(f(x))Df(x) \tag{2.6}$$

where $Df : \mathbf{R}^n \to \mathbf{R}^{m\times n}$, $Dg : \mathbf{R}^m \to \mathbf{R}^{p\times m}$, and $Df : \mathbf{R}^n \to \mathbf{R}^{p\times n}$ are the Jacobian matrix functions of f, g, and h respectively.

Corollary 2.2 *Let $f : \mathbf{R}^n \to \mathbf{R}^m$ and $g : \mathbf{R}^m \to \mathbf{R}$ be differentiable. Then define a function $h : \mathbf{R}^n \to \mathbf{R}$ such that $h(x) = g(f(x))$ for all $x \in \mathbf{dom}\, f$. Then h is differentiable and*

$$\nabla h(x) = Df(x)^T \nabla g(f(x)) \tag{2.7}$$

where $Df : \mathbf{R}^n \to \mathbf{R}^{m\times n}$, $Dg : \mathbf{R}^m \to \mathbf{R}^{p\times m}$, and $Df : \mathbf{R}^n \to \mathbf{R}^{p\times n}$ are the Jacobian matrix functions of f, g, and h respectively.

Corollary 2.3 *Let $f : \mathbf{R}^n \to \mathbf{R}$ be differentiable. Then for some $A \in \mathbf{R}^{n\times m}$ and $b \in \mathbf{R}^n$, define $g : \mathbf{R}^m \to \mathbf{R}$ such that $g(y) = f(Ay + b)$. Then*

$$\nabla g(y) = A^T \nabla f(Ay + b). \tag{2.8}$$

Corollary 2.4 *Let $f : \mathbf{R}^n \to \mathbf{R}$ be twice differentiable. Then for some $A \in \mathbf{R}^{n\times m}$ and $b \in \mathbf{R}^n$, define $g : \mathbf{R}^m \to \mathbf{R}$ such that $g(y) = f(Ay + b)$. Then*

$$\nabla^2 g(y) = A^T \nabla^2 f(Ay + b)A. \tag{2.9}$$

2.4 Integration

Lemma 2.1 *Let $A \in \mathbf{R}^{n\times n}$ be a nonsingular matrix. Suppose that the following integral exists for some $C \subseteq \mathbf{R}^n$.*

$$\int_C f(x)dx \tag{2.10}$$

Chapter 3

Convex analysis

3.1 Convex function

A function $f : \mathbf{R}^n \to \mathbf{R}$ is a convex function if, for all $x, y \in \mathbf{dom}\, f$ and all $0 \le \lambda \le 1$,

$$f(\lambda x + (1-\lambda)y) \le \lambda f(x) + (1-\lambda) f(y). \tag{3.1}$$

Theorem 3.1 *Let $f : \mathbf{R}^n \to \mathbf{R}$. Then for some $x \in \mathbf{dom}\, f$ and $v \in \mathbf{R}^n$, define a function $g_{x,v}(t) : \mathbf{R} \to \mathbf{R}$ such that $g_{x,v}(t) = f(x+tv)$ with the domain, $\{t \in \mathbf{R} \mid x + tv \in \mathbf{dom}\, f\}$. Then f is a convex function iff $g_{x,v}$ is a convex function for any $x \in \mathbf{dom}\, f$ and any $v \in \mathbf{R}^n$.*

Proof: Suppose that f is a convex function. Then for any $x \in \mathbf{dom}\, f$ and $v \in \mathbf{R}^n$, for any $s, t \in \{t \in \mathbf{R} \mid x + tv \in \mathbf{dom}\, f\}$ and any $\lambda \in \mathbf{R}$ such that $0 \le \lambda \le 1$,

$$\begin{array}{l} g_{x,v}(\lambda s + (1-\lambda)t) = f(x + (\lambda s + (1-\lambda)t)v) \\ \quad = \quad f(\lambda(x+sv) + (1-\lambda)(x+tv)) \\ \quad \le \quad \lambda f(x+sv) + (1-\lambda) f(x+tv) = \lambda g_{x,v}(s) + (1-\lambda) g_{x,v}(t). \end{array}$$

Therefore $g_{x,v}$ is a convex function.

Now assume that $g_{x,v}(t) : \mathbf{R} \to \mathbf{R}$ is a convex function for any $x \in \mathbf{dom}\, f$ and $v \in \mathbf{R}^n$. Then for any $x, y \in \mathbf{dom}\, f$ and any $\lambda \in \mathbf{R}$ such that $0 \le \lambda \le 1$,

$$\begin{array}{l} f((1-\lambda)x + \lambda y) = f(x + \lambda(y-x)) \\ \quad = \quad g_{x,y-x}(\lambda) = g_{x,y-x}((1-\lambda)\cdot 0 + \lambda \cdot 1) \le (1-\lambda) g_{x,y-x}(0) + \lambda g_{x,y-x}(1) \\ \quad = \quad (1-\lambda) f(x) + \lambda f(y), \end{array}$$

thus, f is a convex function.

3.1.1 First order condition

Theorem 3.2 *If a function $f : \mathbf{R} \to \mathbf{R}$ is differentiable, it is a convex function iff, for all $x, y \in$ $\mathbf{dom}\, f$,*

$$f(y) \ge f(x) + f'(x)(y-x). \tag{3.2}$$

Proof: Suppose that f is a convex function. Then assume that $y > x$. Let $h \in \mathbf{R}$ be a positive number such that $h < y - x$. Then the definition of convexity implies that

$$f(x+h) \le (1-\lambda) f(x) + \lambda f(y)$$

where $\lambda = h/(y-x)$ since

$$(1-\lambda)x + \lambda y = x + \lambda(y-x) = x + h.$$

Thus

$$f(x+h) - f(x) \le \lambda(f(y) - f(x)) = \frac{h}{y-x}(f(y) - f(x)),$$

which implies

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \le \frac{f(y) - f(x)}{y - x}.$$

Therefore

$$f(y) - f(x) \geq f'(x)(y - x),$$

hence (3.2) is true when $y > x$.

We can prove (3.2) is true when $y < x$ using the very same method. Assume that $x > y$. Let $h \in \mathbf{R}$ be a positive number such that $h < x - y$. Then the definition of convexity implies that

$$f(x - h) \leq (1 - \lambda)f(x) + \lambda f(y)$$

where $\lambda = h/(x - y)$ since

$$(1 - \lambda)x + \lambda y = x + \lambda(y - x) = x - h.$$

Thus

$$f(x) - f(x - h) \geq \lambda(f(x) - f(y)) = \frac{h}{x - y}(f(x) - f(y)),$$

which implies

$$f'(x) = \lim_{h \to 0} \frac{f(x) - f(x - h)}{h} \geq \frac{f(x) - f(y)}{x - y} = \frac{f(y) - f(x)}{y - x}.$$

Therefore

$$f(y) - f(x) \geq f'(x)(y - x),$$

hence (3.2) is true when $y < x$. It is obvise that (3.2) is true when $y = x$. Hence we have just proved that if $f : \mathbf{R} \to \mathbf{R}$ is a convex function, then (3.2) holds for any $x, y \in \mathbf{dom}\, f$.

Now we prove the converse. Suppose that (3.2) holds for any $x, y \in \mathbf{dom}\, f$. Now let $x, y \in \mathbf{dom}\, f$ and $\lambda \in \mathbf{R}$ suc that $0 \leq \lambda \leq 1$. Let $z = \lambda x + (1 - \lambda)y$. Then (3.2) implies that

$$f(x) - f(z) \geq f'(z)(x - z) = (1 - \lambda)f'(z)(x - y) \tag{3.3}$$

and

$$f(y) - f(z) \geq f'(z)(y - z) = \lambda f'(z)(y - x) \tag{3.4}$$

If we multiply λ on both sides of (3.3), multiply $1 - \lambda$ on both sides of (3.4), and add both sides, we have

$$\lambda(f(x) - f(z)) + (1 - \lambda)(f(y) - f(z)) \geq \lambda f(x) + (1 - \lambda)f(y) - f(z) \geq 0,$$

hence

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y).$$

Therefore f is a convex function.

Corollary 3.1 *Let $f : \mathbf{R} \to \mathbf{R}$ be differentiable. Then f is a convex function iff the derivative of f is a nondecreasing function.*

Proof: Suppose that f is a convex function. Let $x, y \in \mathbf{dom}\, f$ such that $x < y$. Then Theorem 3.2 implies

$$f(y) \geq f(x) + f'(x)(y - x)$$

and

$$f(x) \geq f(y) + f'(y)(x - y),$$

thus

$$f'(x) \leq \frac{f(y) - f(x)}{y - x} = \frac{f(x) - f(y)}{x - y} \leq f'(y)$$

since $y > x$. Therefore f' is a nondecreasing function.

Now we prove the converse. Suppose that f' is a nondecreasing function. Then if $y > x$, the mean value theorem implies that there exists some $z \in (x, y)$ such that

$$\frac{f(y) - f(x)}{y - x} = f'(z).$$

Since f' is nondecreasing, we have

$$f'(x) \leq \frac{f(y) - f(x)}{y - x} \leq f'(y),$$

thus

$$f(y) \geq f(x) + f'(x)(y - x) \tag{3.5}$$

and

$$f(y) \leq f(x) + f'(y)(y - x). \tag{3.6}$$

Therefore (3.5) implies that f satisfies (3.2). Now if $x > y$, (3.6) implies that

$$f(x) \leq f(y) + f'(x)(x - y) \Leftrightarrow f(y) \geq f(x) + f'(x)(y - x)$$

which again implies that f satisfies (3.2). Therefore (3.2) implies that f is a convex function.

Corollary 3.2 *If a function $f : \mathbf{R}^n \to \mathbf{R}$ is differentiable, it is a convex function iff, for all $x, y \in \mathbf{dom}\, f$,*

$$f(y) \geq f(x) + \nabla f(x)^T (y - x). \tag{3.7}$$

Proof: Suppose that f is a convex function. Let $x, y \in \mathbf{dom}\, f$. If we let $g_{x,y-x} : \mathbf{R} \to \mathbf{R}$ be a function such that $g_{x,y-x}(t) = f(x + t(y - x))$, Theorem 3.1 implies $g_{x,y-x}$ is a convex function. Therefore Theorem 3.2 together with Corollary 2.3 implies

$$f(y) = g_{x,y-x}(1) \geq g_{x,y-x}(0) + g'_{x,y-x}(0)(1 - 0) = f(x) + \nabla f(x)^T (y - x)$$

for any $x, y \in \mathbf{dom}\, f$.

Now suppose that (3.7) holds for any $x, y \in \mathbf{dom}\, f$. Then Corollary 2.3 implies that, for any $r, s \in \mathbf{R}$ and $v \in \mathbf{R}^n$ such that $r, s \in \{t \in \mathbf{R} \mid x + tv \in \mathbf{dom}\, f\}$,

$$g_{x,v}(r) = f(x + rv) \geq f(x + sv) + (r - s)\nabla f(x + sv)^T v = g_{x,v}(s) + g'_{x,v}(s)(r - s).$$

Thus Theorem 3.2 implies $g_{x,v}$ is a convex function for any $x \in \mathbf{dom}\, f$ and $v \in \mathbf{R}^n$. Therefore by Theorem 3.1, f is a convex function.

3.1.2 Second order condition

Theorem 3.3 *If a function $f : \mathbf{R} \to \mathbf{R}$ is twice differentiable, it is a convex function iff, for all $x \in \mathbf{dom}\, f$,*

$$f''(x) \geq 0. \tag{3.8}$$

Proof: Suppose that f is a convex function. Then Corollary 3.1 implies that f' is a nondecreasing function, hence

$$f''(x) = \lim_{h \to 0} \frac{f'(x+h) - f'(x)}{h} = \lim_{h \to 0^+} \frac{f'(x+h) - f'(x)}{h} \geq 0.$$

Now if $f''(x) \geq 0$ for all $x \in \mathbf{dom}\, f$, the mean value theorem implies that f' is a nondecreasing function.

Theorem 3.4 *If a function $f : \mathbf{R}^n \to \mathbf{R}$ is twice differentiable, it is a convex function iff, for all $x \in \mathbf{dom}\, f$,*

$$\nabla^2 f(x) \succeq 0. \tag{3.9}$$

Proof: Suppose that f is a convex function. Then Theorem 3.1 implies that for any $x \in \mathbf{dom}\, f$ and any $v \in \mathbf{R}^n$, the function $g_{x,v} : \mathbf{R} \to \mathbf{R}$ such that $g_{x,v}(t) = f(x + tv)$ is a convex function in $\{t \in \mathbf{R} \mid x + tv \in \mathbf{dom}\, f\}$. Then Theorem 3.3 together with Corollary 2.4 implies that

$$v^T \nabla^2 f(x) v = g''_{x,v}(0) \geq 0.$$

Therefore $\nabla^2 f(x) \succeq 0$ for any $x \in \mathbf{dom}\, f$.

Now if $\nabla^2 f(x) \succeq 0$ for all $x \in \mathbf{dom}\, f$, then Corollary 2.4 implies that $g''_{x,v}(t) = v^T \nabla^2 f(x + tv) v \geq 0$ for any $x \in \mathbf{dom}\, f$ and any $v \in \mathbf{R}^n$. Then Theorem 3.3 implies $g_{x,v}$ is a convex function for any $x \in \mathbf{dom}\, f$ and any $v \in \mathbf{R}^n$, hence by Theorem 3.1, f is a convex function.

Chapter 4

Linear Algebra

4.1 Vector space

Cauchy–Schwarz inequality: for $a, b \in \mathbf{C}^n$,

$$|a^H b| \leq \|a\|_2 \|b\|_2. \tag{4.1}$$

The generalized form: for $f : [0,1] \to \mathbf{C}$ and $g : [0,1] \to \mathbf{C}$,

$$\left| \int_0^1 \overline{f(t)} g(t)\, dt \right| \leq \left(\int_0^1 |f(t)|^2 dt \right)^{1/2} \left(\int_0^1 |g(t)|^2 dt \right)^{1/2}. \tag{4.2}$$

Hölder's inequality: for $a, b \in \mathbf{C}^n$, $p > 1$, and $q > 1$ such that $1/p + 1/q = 1$,

$$|a^H b| \leq \|a\|_p \|b\|_q. \tag{4.3}$$

The generalized form: for $f : [0,1] \to \mathbf{C}$, $g : [0,1] \to \mathbf{C}$, $p > 1$, and $q > 1$ such that $1/p + 1/q = 1$,

$$\left| \int_0^1 \overline{f(t)} g(t)\, dt \right| \leq \left(\int_0^1 |f(t)|^p dt \right)^{1/p} \left(\int_0^1 |g(t)|^q dt \right)^{1/q}. \tag{4.4}$$

When $b = \mathbf{1}$, the Cauchy-Schwarz inequality implies

$$\|a\|_1 \leq n \|a\|_2 \tag{4.5}$$

4.2 Determinant

4.2.1 Definition

Let $A \in \mathbf{R}^{n \times n}$. Then the determinant of A is defined by

$$\det(A) = \sum_{\sigma \in \Pi(n)} \mathbf{sign}(\sigma) \prod_{i=1}^n A_{i,\sigma(i)} \tag{4.6}$$

where $\Pi(n)$ is the symmetric group of permutations of $\{1, \ldots, n\}$.

Characteristic polynomial For $A \in \mathbf{R}^{n \times n}$, the characteristic of A is defined by the following (nth order) polynomial.

$$p(x) = \det(xI_n - A). \tag{4.7}$$

Note that even though $xI_n - A \notin \mathbf{R}^{n \times n}$, *i.e.*, rather $xI_n - A \in \mathbf{R}(1)^{n \times n}$ where $\mathbf{R}(k)$ denotes the set of all k-th order polynomial with real coefficients, the determinant can be defined in the same way that that for real matrix is defined, *e.g.*, using (4.6).

4.2.2 Properties

multiplicative property of determinants For $A, B \in \mathbf{R}^{n\times n}$,

$$\det(AB) = \det(A)\det(B). \tag{4.8}$$

Proof: By (4.6),

$$\begin{aligned}
\det(AB) &= \sum_{\sigma\in\Pi(n)} \mathbf{sign}(\sigma)\prod_{i=1}^{n}(AB)_{i,\sigma(i)} \\
&= \sum_{\sigma\in\Pi(n)} \mathbf{sign}(\sigma)\prod_{i=1}^{n}\sum_{j=1}^{n} A_{i,j}B_{j,\sigma(i)} \\
&= \sum_{\sigma\in\Pi(n)} \mathbf{sign}(\sigma)\sum_{1\le j_1\le n,\ldots,1\le j_n\le n}\prod_{i=1}^{n} A_{i,j_i}B_{j_i,\sigma(i)} \\
&= \sum_{1\le j_1\le n,\ldots,1\le j_n\le n}\sum_{\sigma\in\Pi(n)} \mathbf{sign}(\sigma)\prod_{i=1}^{n} A_{i,j_i}B_{j_i,\sigma(i)} \\
&= \sum_{1\le j_1\le n,\ldots,1\le j_n\le n}\prod_{i=1}^{n} A_{i,j_i}\left(\sum_{\sigma\in\Pi(n)} \mathbf{sign}(\sigma)\prod_{k=1}^{n} B_{j_k,\sigma(k)}\right) \\
&= \sum_{\tau\in\Pi(n)}\prod_{i=1}^{n} A_{i,\tau(i)}\left(\sum_{\sigma\in\Pi(n)} \mathbf{sign}(\sigma)\prod_{k=1}^{n} B_{\tau(k),\sigma(k)}\right) \\
&= \sum_{\tau\in\Pi(n)}\prod_{i=1}^{n} A_{i,\tau(i)}\left(\sum_{\sigma\in\Pi(n)} \mathbf{sign}(\sigma)\prod_{k=1}^{n} B_{k,(\sigma\circ\tau^{-1})(k)}\right) \\
&= \sum_{\tau\in\Pi(n)}\prod_{i=1}^{n} A_{i,\tau(i)}\left(\sum_{\sigma\in\Pi(n)} \mathbf{sign}(\sigma\circ\tau^{-1})\mathbf{sign}(\tau^{-1})\prod_{k=1}^{n} B_{k,(\sigma\circ\tau^{-1})(k)}\right) \\
&= \sum_{\tau\in\Pi(n)}\prod_{i=1}^{n} A_{i,\tau(i)}\left(\sum_{\sigma\in\Pi(n)} \mathbf{sign}(\sigma\circ\tau^{-1})\mathbf{sign}(\tau)\prod_{k=1}^{n} B_{k,(\sigma\circ\tau^{-1})(k)}\right) \\
&= \sum_{\tau\in\Pi(n)}\mathbf{sign}(\tau)\prod_{i=1}^{n} A_{i,\tau(i)}\left(\sum_{\sigma\in\Pi(n)} \mathbf{sign}(\sigma\circ\tau^{-1})\prod_{k=1}^{n} B_{k,(\sigma\circ\tau^{-1})(k)}\right) \\
&= \det(B)\sum_{\tau\in\Pi(n)}\mathbf{sign}(\tau)\prod_{i=1}^{n} A_{i,\tau(i)} = \det(B)\det(A) \\
&= \det(A)\det(B).
\end{aligned}$$

Determinant in terms of minors The determinant of $A \in \mathbf{R}^{n\times n}$ can be expressed in terms of its minors, *i.e.*, for all $1 \leq k \leq n$,

$$\det(A) = \sum_{j=1}^{n} (-1)^{k+j} A_{k,j} \,\mathbf{minor}(A) = \sum_{i=1}^{n} (-1)^{i+k} A_{i,k} \,\mathbf{minor}(A) \tag{4.9}$$

where $\mathbf{minor}(A)_{ij}$ is the (i,j)-minor of A, *i.e.*, the determinant of the $(n-1)$-by-$(n-1)$ matrix that results from deleting row i and column j of A.

Upper and lower triangular matrices The determinant of an upper or lower triangular matrix is the product of the diagonal entries.

Matrix inverse formula For a square matrix $A \in \mathbf{R}^{n\times n}$, if $\det(A) \neq 0$, the inverse matrix is uniquely determined and it can be expressed as

$$A^{-1} = \frac{1}{\det(A)} \,\mathbf{adj}(A) \tag{4.10}$$

where $\mathbf{adj}(A) \in \mathbf{R}^{n\times n}$ denotes the adjugate of A which is defined by

$$\mathbf{adj}(A) = \begin{bmatrix} \mathbf{minor}(A)_{1,1} & -\mathbf{minor}(A)_{2,1} & \cdots & (-1)^{n+1} \mathbf{minor}(A)_{n,1} \\ -\mathbf{minor}(A)_{1,2} & \mathbf{minor}(A)_{2,2} & \cdots & (-1)^{n+1} \mathbf{minor}(A)_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ (-1)^{n+1} \mathbf{minor}(A)_{1,n} & (-1)^{n+2} \mathbf{minor}(A)_{2,n} & \cdots & \mathbf{minor}(A)_{n,n} \end{bmatrix}, \tag{4.11}$$

i.e.,

$$\mathbf{adj}(A)_{i,j} = (-1)^{i+j} \,\mathbf{minor}(A)_{j,i} \tag{4.12}$$

for all $1 \leq i, j \leq n$.

Singularity A square matrix is invertible if and only if its determinant is nonzero.

> *Proof*: Suppose that $A \in \mathbf{R}^{n\times n}$ is invertible. Then there exists $B \in \mathbf{R}^{n\times n}$ such that $AB = I_n$. The multiplicative property of determinants implies that $\det(A)\det(B) = 1$. Therefore $\det(A) \neq 0$. Conversely, if $\det(A) \neq 0$, there exists the inverse matrix of A given by (4.10). Therefore A is invertible, *i.e.*, nonsignular if and only if $\det(A) \neq 0$.

Characteristic polynomial and eigvenvalues For $A \in \mathbf{R}^{n\times n}$, λ is an eigenvalue of A if and only if it is a root for the characteristic polynomial of A, *i.e.*,

$$\det(\lambda I - A) = 0. \tag{4.13}$$

Cayley–Hamilton theorem For a square matrix $A \in \mathbf{R}^{n\times n}$, $p(A) = 0$ where p is the characteristic polynomial of A given by (4.7).

4.3 Eigenvalues

4.3.1 Basic definitions

Given a square matrix $A \in \mathbf{R}^{n\times n}$, if there exist $\lambda \in \mathbf{C}$ and nonzero $v \in \mathbf{C}^n$ such that

$$Av = \lambda v \tag{4.14}$$

then λ is called an eigenvalue of A and v is called an eigenvector associated with λ.

If there exist n linearly independent eigenvectors, we have

$$A \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix} = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix} \mathbf{diag}(\lambda_1, \ldots, \lambda_n) \tag{4.15}$$

or

$$AV = V\Lambda \tag{4.16}$$

where

$$V = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix} \in \mathbf{C}^{n \times n} \tag{4.17}$$

and

$$\Lambda = \mathbf{diag}(\lambda_1, \ldots, \lambda_n) = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \in \mathbf{C}^{n \times n}. \tag{4.18}$$

In this case, A is said to be diagonalizable.

Since V is nonsingular, *i.e.*, invertible, we can rewrite (4.16) as

$$A = V\Lambda V^{-1} \Leftrightarrow V^{-1}AV = \Lambda. \tag{4.19}$$

4.3.2 Symmetric matrices

Given a symmetric matrix $A = A^T \in \mathbf{R}^{n \times n}$, all the eigenvalues are real and we can choose n real orthonormal eigenvectors, *i.e.*, we can find n eigenvectors $v_1, \ldots, v_n \in \mathbf{R}^n$ associated with n eigenvectors, $\lambda_1, \ldots, \lambda_n \in \mathbf{R}$ such that

$$\|v_i\| = 1 \tag{4.20}$$

for $i = 1, \ldots, n$ and

$$v_i^T v_j = 0 \tag{4.21}$$

for $1 \leq i \neq j \leq n$. Thus, all symmetric matrices are diagonalizable.

Now (4.19) becomes

$$A = V\Lambda V^T \Leftrightarrow V^T AV = \Lambda \tag{4.22}$$

since

$$V^T V = I_n \tag{4.23}$$

where $I_n \in \mathbf{R}^{n \times n}$ is the indentity matrix. We can rewrite (4.22) as

$$A = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix} \mathbf{diag}(\lambda_1, \ldots, \lambda_n) \begin{bmatrix} v_1^T \\ \vdots \\ v_n^T \end{bmatrix} = \sum_{i=1}^{n} \lambda_i v_i v_i^T. \tag{4.24}$$

4.4 Positive definiteness

- A symmetric matrix $A = A^T \in \mathbf{R}^{n\times n}$ is called positive semidefinite if for all $x \in \mathbf{R}^n$,

$$x^T A x \geq 0. \tag{4.25}$$

- A symmetric matrix $A = A^T \in \mathbf{R}^{n\times n}$ is called positive definite if for all nonzero $x \in \mathbf{R}^n$,

$$x^T A x > 0. \tag{4.26}$$

- The set of all the n-by-n positive semidefinite matrices is (sometimes) denoted by $\mathcal{S}^n_+$, *i.e.*,

$$\mathcal{S}^n_+ = \{A = A^T \in \mathbf{R}^{n\times n} \mid x^T A x \geq 0 \text{ for all } x \in \mathbf{R}^n\}. \tag{4.27}$$

- The set of all the n-by-n positive definite matrices is (sometimes) denoted by $\mathcal{S}^n_{++}$, *i.e.*,

$$\mathcal{S}^n_{++} = \{A = A^T \in \mathbf{R}^{n\times n} \mid x^T A x > 0 \text{ for all nonzero } x \in \mathbf{R}^n\}. \tag{4.28}$$

- $A = A^T \in \mathbf{R}^{n\times n}$ is positive semidefinite if and only if all the eigenvalues of A are nonnegative.
- $A = A^T \in \mathbf{R}^{n\times n}$ is positive definite if and only if all the eigenvalues of A are positive.

Proof: For symmetric $A = A^T$, there exist orthgonal $V \in \mathbf{R}^{n\times n}$ and diagonal $\Lambda \in \mathbf{R}^{n\times n}$ such that

$$A = V\Lambda V^T = \sum_{i=1}^n \lambda_i v_i v_i^T,$$

thus for any $x \in \mathbf{R}^n$,

$$x^T A x = x^T \left(\sum_{i=1}^n \lambda_i v_i v_i^T\right) x = \sum_{i=1}^n \lambda_i x^T v_i v_i^T x = \sum_{i=1}^n \lambda_i (v_i^T x)^2.$$

Therefore if all λ_i are nonnegative, $x^T A x \geq 0$ for any $x \in \mathbf{R}^n$, hence $A \in \mathcal{S}^n_+$. Now assume $A \in \mathcal{S}^n_+$, but $\lambda_j < 0$ for some $j \in \{1, \ldots, n\}$. Then

$$v_j^T A v_j = \sum_{i=1}^n \lambda_i (v_i^T v_j)^2 = \sum_{i=1}^n \lambda_i \delta_{i,j} = \lambda_j < 0 \tag{4.29}$$

since v_1, …, v_n are orthonormal where $\delta_{i,j}$ is the Kronecker delta function, hence $A \not\in \mathcal{S}^n_+$. Therefore if $A \in \mathcal{S}^n_+$, all λ_i are nonnegative.

Therefore $A \in \mathcal{S}^n_+$ if and only if all λ_i are nonnegative.

Now assume that all i are positive. Then for all nonzero $x \in \mathbf{R}^n$, there exists $i \in \{1, \ldots, n\}$ such tat $v_i^T x$ since if $v_i^T x = 0$ for all i, then $V^T x = 0$, hence $x = 0$ since V^T is nonsigular. Therefore

$$x^T A x = \sum_{i=1}^n \lambda_i (v_i^T x)^2 \geq \lambda_j (v_j^T x)^2 > 0. \tag{4.30}$$

Thus, $A \in \mathcal{S}^n_{++}$.

Now assume that $A \in \mathcal{S}^n_{++}$. If $\lambda_j \leq 0$ for some $j \in \{1, \dots, n\}$, then

$$v_j^T A v_j = \sum_{i=1}^{n} \lambda_i \delta_{i,j} = \lambda_j \leq 0, \tag{4.31}$$

hence $A \notin \mathcal{S}^n_{++}$. Therefore if $A \in \mathcal{S}^n_{++}$, all λ_i are positive.

Therefore $A \in \mathcal{S}^n_{++}$ if and only if all λ_i are positive.

4.5 Matrix norms

$$\mathbf{dist}(C_{\text{org1}}, C_{\text{org2}}) = \|C_{\text{org1}} - C_{\text{org2}}\| = |\lambda_{\max}(C_{\text{org1}} - C_{\text{org2}})| \tag{4.32}$$

Part II

Optimization

Chapter 5

Convex Optimization

5.1 Mathematical optimization problem

A mathematical optimization problem can be expressed as

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0 \text{ for } i = 1, \ldots, m \\ & h_i(x) = 0 \text{ for } i = 1, \ldots, p \end{array} \tag{5.1}$$

where $x \in \mathbf{R}^n$ is the optimization variable, $f_0 : \mathbf{R}^n \to \mathbf{R}$ is the objective function, $f_i : \mathbf{R}^n \to \mathbf{R}$ for $i = 1, \ldots, n$ are the inequality constraint functions, and $h_i : \mathbf{R}^n \to \mathbf{R}$ for $i = 1, \ldots, p$ are the equality constraint functions.

The conditions, $f_i(x) \leq 0$ for $i = 1, \ldots, m$, are called inequality constraints and the conditions, $h_i(x) = 0$ for $i = 1, \ldots, p$ are called equation constraints.

Note that this formulation covers pretty much every single-objective optimization problem. For example, consider the following optimization problem.

$$\begin{array}{ll} \text{maximize} & f(x_1, x_2) \\ \text{subject to} & x_1 \geq x_2 \\ & x_1 + x_2 = 2 \end{array} \tag{5.2}$$

This problem can be cast into an equivalent problem as follows.

$$\begin{array}{ll} \text{minimize} & -f(x_1, x_2) \\ \text{subject to} & -x_1 + x_2 \leq 0 \\ & x_1 + x_2 - 2 = 0 \end{array} \tag{5.3}$$

The feasible set for (5.1) is defined by the set of $x \in \mathbf{R}^n$ which satisfies all the contraints. Also, the optimal value for (5.1) is the infimum of $f_0(x)$ while x is in the feasible set. When the infimum is achievable, we define the optimal solution set as the set of all feasible x achieving the infimum value. These are defined in mathematically rigorous terms below.

- The feasible set for (5.1) is defined by

  $$\mathcal{F} = \{x \in \mathcal{D} \mid f_i(x) \leq 0 \text{ for } i = 0, \ldots, m,\ h_j(x) = 0 \text{ for } j = 1, \ldots, p\} \subseteq \mathbf{R}^n \tag{5.4}$$

  where

  $$\mathcal{D} = \left(\bigcap_{0 \leq i \leq m} \mathbf{dom}\, f_i \right) \cap \left(\bigcap_{1 \leq i \leq p} \mathbf{dom}\, h_i \right). \tag{5.5}$$

- The optimal value for (5.1) is defined by

  $$p^* = \inf_{x \in \mathcal{F}} f_0(x) \tag{5.6}$$

  We use the conventions that $p^* = -\infty$ if $f_0(x)$ is unbounded below for $x \in \mathcal{F}$ and that $p^* = \infty$ if $\mathcal{F} = \emptyset$.

- The optimal solution set for (5.1) is defined by

  $$\mathcal{X}^* = \{x \in \mathcal{F} \mid f_0(x) = p^*\}. \tag{5.7}$$

5.2 Convex optimization problem

A mathematical optimization problem is called a convex optimization problem if the objective function and all the inequality constraint functions are convex functions and all the equality constraint functions are affine functions.

Hence, a convex optimization problem can be expressed as

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0 \text{ for } i = 1, \ldots, m \\ & Ax = b \end{array} \tag{5.8}$$

where $x \in \mathbf{R}^n$ is the optimization variable, $f_i : \mathbf{R}^n \to \mathbf{R}$ for $i = 0, \ldots, n$ are convex functions, $h_i : \mathbf{R}^n \to \mathbf{R}$ for $i = 1, \ldots, p$ are the equality constraint functions, $A \in \mathbf{R}^{p \times n}$, and $b \in \mathbf{R}^p$.

A function, $f : \mathbf{R}^n \to \mathbf{R}$, is called a convex function if $\mathbf{dom}\, f \subseteq \mathbf{R}^n$ is a convex set and for all $x, y \in \mathbf{dom}\, f$ and all $0 \leq \lambda \leq 1$,

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y) \tag{5.9}$$

where $\mathbf{dom}\, f \subseteq \mathbf{R}^n$ denotes the domain of f.

A convex optimization enjoys a number of nice theoretical and practical properties.

- A local minimum of a convex optimization problem is a global minimum, *i.e.*, if for some $R > 0$ and $x_0 \in \mathcal{F}$, $\|x - x_0\| < R$ and $x \in \mathcal{F}$ imply $f_0(x_0) \leq f_0(x)$, then $f_0(x_0) \leq f_0(x)$ for all $x \in \mathcal{F}$.

  Proof: Assume that $x_0 \in \mathcal{F}$ is a local minimum, *i.e.*, for some $R > 0$, $\|x - x_0\| < R$ and $x \in \mathcal{F}$ imply $f_0(x_0) \leq f_0(x)$.

  Now assume that x_0 is not a global minimum, *i.e.*, there exists $y \in \mathcal{F}$ such that $y \neq x_0$ and $f_0(y) < f_0(x_0)$. Then for $z = \lambda y + (1-\lambda)x_0$ with $\lambda = \min\{R/\|y - x_0\|, 1\}/2$, the convexity of f_0 implies

  $$f_0(z) \leq \lambda f_0(y) + (1-\lambda) f_0(x_0) \tag{5.10}$$

  since $0 < \lambda \leq 1/2 < 1$. Furthermore

  $$\|z - x_0\| = \lambda \|y - x_0\| \leq R/2, \tag{5.11}$$

  hence $f_0(z) \geq f_0(x_0)$, which together with (5.10) implies

  $$f_0(x_0) \leq f_0(z) \leq \lambda f_0(y) + (1-\lambda) f_0(x_0) < \lambda f_0(x_0) + (1-\lambda) f_0(x_0) = f_0(x_0), \tag{5.12}$$

  which is a contradiction. Therefore there is no $y \in \mathcal{F}$ such that $y \neq x_0$ and $f_0(y) < f_0(x_0)$. Therefore x_0 is a global minimum.

- For a unconstrained problem, *i.e.*, the problem (5.8) with $m = p = 0$, with differentiable objective function, $x \in \mathbf{dom}\, f_0$ is an optimal solution if and only if $\nabla f_0(x) = 0 \in \mathbf{R}^n$.

Proof: The Taylor theorem implies that for any $x, y \in \mathbf{dom}\, f_0$,

$$f_0(y) = f(x) + \nabla f_0(x)^T(y-x) + \frac{1}{2}(y-x)^T\nabla^2 f_0(z)(y-x) \tag{5.13}$$

for some z on the line segment having x and y as its end points, *i.e.*, $z = \alpha x + (1-\alpha)y$ for some $0 \leq \alpha \leq 1$. Since $\nabla^2 f(x) \succeq 0$ for any $z \in \mathbf{dom}\, f_0$, we have

$$f_0(y) \geq f_0(x) + \nabla f_0(x)^T(y-x) \tag{5.14}$$

Thus, if for some $x_0 \in \mathbf{R}^n$, $\nabla f_0(x_0) = 0$, for any $x \in \mathbf{dom}\, f_0$,

$$f_0(x) \geq f_0(x_0) + \nabla f_0(x_0)^T(x - x_0) = f_0(x_0), \tag{5.15}$$

hence x_0 is an optimal solution. Now assume that x_0 is an optimal solution, but $\nabla f_0(x_0) \neq 0$. Then for any $k > 0$, if we let $x = x_0$ and $y = x_0 - k\nabla f_0(x_0)$, (5.13) becomes

$$\begin{aligned} f_0(y) = f(x_0) + \nabla f_0(x_0)^T(-k\nabla f_0(x_0)) + \frac{k^2}{2}\nabla f_0(x_0)^T\nabla^2 f_0(z)\nabla f_0(x_0) \\ = \quad f(x_0) - k\|\nabla f_0(x_0)\|^2 + \frac{k^2}{2}\nabla f_0(x_0)^T\nabla^2 f_0(z)\nabla f_0(x_0) \end{aligned}$$

for all $y = x_0 - k\nabla f_0(x_0) \in \mathbf{dom}\, f_0$.

Since for $k < 2\|\nabla f_0(x_0)\|^2/\nabla f_0(x_0)^T\nabla^2 f_0(z)\nabla f_0(x_0)$, $-k\|\nabla f_0(x_0)\|^2 + \frac{k^2}{2}\nabla f_0(x_0)^T\nabla^2 f_0(z)\nabla f_0(x_0) < 0$, thus $f_0(y) < f(x_0)$, hence the constradiction. Therefore, if x_0 is an optimal solution for the unconstrained problem, $\nabla f_0(x_0) = 0$.

5.3 Duality

5.3.1 The Lagrange dual problem

5.3.1.1 Examples

5.3.1.1.1 Standard form LP

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \succeq 0 \end{array} \tag{5.16}$$

The Lagrange dual problem is

$$\begin{array}{ll} \text{maximize} & -b^T\nu \\ \text{subject to} & A^T\nu + c \geq 0 \end{array} \tag{5.17}$$

5.3.1.1.2 Inequality form LP

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax \preceq b \end{array} \tag{5.18}$$

The Lagrange dual problem is

$$\begin{array}{ll} \text{maximize} & -b^T \lambda \\ \text{subject to} & A^T \lambda + c = 0 \\ & \lambda \succeq 0 \end{array} \tag{5.19}$$

5.3.1.1.3 Least-squares solution of linear equations

$$\begin{array}{ll} \text{minimize} & (1/2) x^T x \\ \text{subject to} & Ax = b \end{array} \tag{5.20}$$

The Lagrange dual problem is

$$\text{maximize} \quad -(1/2)\nu^T A A^T \nu - b^T \nu \tag{5.21}$$

5.3.1.1.4 Entropy maximization

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^n x_i \log x_i \\ \text{subject to} & Ax = b \\ & \mathbf{1}^T x = 1 \end{array} \tag{5.22}$$

with domain $\mathcal{D} = \mathbf{R}_+^n$

The Lagrange dual problem is

$$\begin{array}{ll} \text{maximize} & -b^T \lambda - \log\left(\sum_{i=1}^n \exp(-a_i^T \lambda)\right) \\ \text{subject to} & \lambda \succeq 0 \end{array} \tag{5.23}$$

5.3.2 Interpretations

5.3.2.1 Max-min characterization of weak and strong duality

We first note that for any $f : X \times Y \to \mathbf{R}$, we have

$$\sup_{y \in Y} \inf_{x \in X} f(x, y) \leq \inf_{x \in X} \sup_{y \in Y} f(x, y). \tag{5.24}$$

This inequality is called *max-min inequality.*

We can prove this as follows. Let $g : Y \to \mathbf{R}$ be a function defined by $g(y) = \inf_{x \in X} f(x, y)$ and let $h : X \to \mathbf{R}$ be a function defined by $h(x) = \sup_{y \in Y} f(x, y)$. Then we have that for any $x \in X$ and $y \in Y$

$$g(y) = \inf_{x \in X} f(x, y) \leq f(x, y), \tag{5.25}$$

which implies that for any $x \in X$

$$\sup_{y\in Y} g(y) \le \sup_{y\in Y} f(x,y) = h(x). \tag{5.26}$$

This again implies that

$$\sup_{y\in Y} g(y) \le \inf_{x\in X} h(x), \tag{5.27}$$

hence the proof.

5.3.2.2 Saddle-point interpretation

Suppose $f : X \times Y \to \mathbf{R}$. We refer a point $(\tilde{x},\tilde{y}) \in X \times Y$ a *saddle-point* for f (and X and Y) if

$$f(\tilde{x},y) \le f(\tilde{x},\tilde{y}) \le f(x,\tilde{y}) \tag{5.28}$$

for all $x \in X$ and $y \in Y$.

Now if x^* and λ^* are primal and dual optimal points for a problem in which strong duality obtains, the form a saddle-point for the Lagrangian. Conversely, if (x,λ) is a saddle-point of the Lagrangian, then x is primal optimal, λ is dual optimal, and the optimal duality gap is zero.

To prove these, assume that $x^* \in \mathcal{D}$ and $(\lambda^*,\nu^*) \in \mathbf{R}^m_+ \times \mathbf{R}^p$ are primal and dual optimal points for a problem in which strong duality obtains. Then for any $x \in \mathcal{D}$ and $(\lambda,\nu) \in \mathbf{R}^m_+ \times \mathbf{R}^p$, we have

$$L(x^*,\lambda,\nu) = f_0(x^*) + \sum_{i=1}^m \lambda_i f_i(x^*) + \sum_{i=1}^p \nu_i h_i(x^*) \le f_0(x^*) = g(\lambda^*,\nu^*) \le L(x,\lambda^*,\nu^*) \tag{5.29}$$

where the left inequality comes from the fact that $\lambda_i f_i(x^*) \le 0$ for all $i = 1,\ldots,m$ and $h_i(x^*) = 0$ for all $i = 1,\ldots,p$ and the right inequality comes from the definition of (Lagrange) dual function. Now from the complementary slackness we know that $\lambda_i f_i(x^*) = 0$ for all $i = 1,\ldots,m$. Therefore

$$L(x^*,\lambda^*,\nu^*) = f_0(x^*), \tag{5.30}$$

thus we have

$$L(x^*,\lambda,\nu) \le L(x^*,\lambda^*,\nu^*) \le L(x,\lambda^*,\nu^*), \tag{5.31}$$

hence the proof.

Now suppose that $\tilde{x} \in \mathcal{D}$ and $(\tilde{\lambda},\tilde{\nu}) \in \mathbf{R}^m_+ \times \mathbf{R}^p$ are the saddle-point of the Lagrangian, *i.e.*, for all $x \in \mathcal{D}$ and $(\lambda,\nu) \in \mathbf{R}^m_+ \times \mathbf{R}^p$,

$$L(\tilde{x},\lambda,\nu) \le L(\tilde{x},\tilde{\lambda},\tilde{\nu}) \le L(x,\tilde{\lambda},\tilde{\nu}). \tag{5.32}$$

First we show that $\tilde{x}$ is a feasible point. The left inequality says that for all $(\lambda,\nu) \in \mathbf{R}^m_+ \times \mathbf{R}^p$,

$$L(\tilde{x},\lambda,\nu) = f_0(\tilde{x}) + \sum_{i=1}^m \lambda_i f_i(\tilde{x}) + \sum_{i=1}^p \nu_i h_i(\tilde{x}) \le L(\tilde{x},\tilde{\lambda},\tilde{\nu}) \tag{5.33}$$

If $f_i(\tilde{x}) > 0$ for some $i \in \{1,\ldots,m\}$ or $h_i(\tilde{x}) \ne 0$ for some $i \in \{1,\ldots,p\}$, $L(\tilde{x},\lambda,\nu)$ is unbounded above and the above inequality cannot hold. Therefore $f_i(\tilde{x}) \le 0$ for all $i \in \{1,\ldots,m\}$ and $h_i(\tilde{x}) = 0$

for all $i \in \{1, \dots, p\}$, *i.e.*, $\tilde{x}$ is primal feasible. Since the inequality must hold when $\lambda = 0$ and $\nu = 0$, we have

$$f(\tilde{x}) \leq L(\tilde{x}, \tilde{\lambda}, \tilde{\nu}). \tag{5.34}$$

The right inequality of (5.32) implies that

$$L(\tilde{x}, \tilde{\lambda}, \tilde{\nu}) \leq g(\tilde{\lambda}, \tilde{\nu}) = \inf_{x \in \mathcal{D}} L(x, \tilde{\lambda}, \tilde{\nu}), \tag{5.35}$$

which implies that $f_0(\tilde{x}) \leq g(\tilde{\lambda}, \tilde{\nu})$. Since $g(\lambda, \nu)$ is an underestimator of $f_0(x)$ for any feasible $x \in \mathcal{D}$ and $(\tilde{\lambda}, \tilde{\nu}) \in \mathbf{R}_+^m \times \mathbf{R}^p$, *i.e.*, $g(\tilde{\lambda}, \tilde{\nu}) \leq f_0(\tilde{x})$, thus $g(\tilde{\lambda}, \tilde{\nu}) = f_0(\tilde{x})$. Therefore $\tilde{x}$ is an optimal solution for the primal problem and $(\tilde{\lambda}, \tilde{\nu})$ is an optimal solution for the dual problem, hence the proof.

5.4 Convex optimization problems

5.4.1 Equality constrained problem

Consider the following equality constrained problem:

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & Ax = b \end{array} \tag{5.36}$$

where $x \in \mathbf{R}^n$ is the optimization variable, $A \in \mathbf{R}^{m \times n}$, and $b \in \mathbf{R}^m$. The Lagrangian is

$$L(x, \nu) = f(x) + \nu^T (Ax - b) \tag{5.37}$$

and the Lagrange dual function is

$$g(\nu) = \inf_{x \in \mathbf{R}^n} L(x, \nu) = - \sup_{x \in \mathbf{R}^n} (-\nu^T Ax - f(x)) - b^T \nu = -f^*(-A^T \nu) - b^T \nu \tag{5.38}$$

The KKT optimality conditions are

$$\begin{array}{rl} \text{primal feasibility:} & Ax = b \quad (5.39) \\ \text{gradient of Lagrangian vanishes:} & \nabla f(x) + A^T \nu = 0 \quad (5.40) \end{array}$$

5.4.1.1 Equality constrained problem examples

Consider the following equality constraint quadratic problem:

$$\begin{array}{ll} \text{minimize} & x^T P x + q^T x \\ \text{subject to} & Ax = b \end{array} \tag{5.41}$$

where $x \in \mathbf{R}^n$ is the optimization variable, $P \in \mathcal{S}_{++}^n$, $q \in \mathbf{R}^n$, $A \in \mathbf{R}^{m \times n}$, and $b \in \mathbf{R}^m$.

The Lagrangian is

$$L(x, \nu) = x^T P x + q^T x + \nu^T (Ax - b). \tag{5.42}$$

The gradient of the Lagrangian with respect to x is

$$\nabla_x L(x,\nu) = 2Px + q + A^T\nu = 0, \tag{5.43}$$

hence

$$\operatorname*{argmin}_x L(x,\nu) = -\frac{1}{2}P^{-1}(q + A^T\nu) \tag{5.44}$$

The KKT conditions are

$$\begin{aligned} \text{primal feasibility:} \quad & Ax = b & (5.45)\\ \text{gradient of Lagrangian vanishes:} \quad & 2Px + q + A^T\nu = 0 & (5.46) \end{aligned}$$

which are equivalent to

$$\begin{bmatrix} 2P & A^T \\ A & 0 \end{bmatrix}\begin{bmatrix} x \\ \nu \end{bmatrix} = \begin{bmatrix} -q \\ b \end{bmatrix}. \tag{5.47}$$

The conjugate of the objective function is

$$f^*(y) = \sup_x (y^Tx - x^TPx - q^Tx). \tag{5.48}$$

Since the gradient of $y^Tx - x^TPx - q^Tx$ is $y - q - 2Px$,

$$\operatorname*{argsup}_x (y^Tx - x^TPx - q^Tx) = \frac{1}{2}P^{-1}(y-q), \tag{5.49}$$

thus

$$\begin{aligned} f^*(y) &= -\frac{1}{4}(y-q)^TP^{-1}(y-q) + \frac{1}{2}(y-q)^TP^{-1}(y-q) = \frac{1}{4}(y-q)^TP^{-1}(y-q) \\ &= \frac{1}{4}\left(y^TP^{-1}y - 2q^TP^{-1}y + q^TP^{-1}q\right) \end{aligned}$$

5.4.2 Special optimization problem examples

5.4.2.1 Unconstrained max-det problem

We consider the following unconstrained max-det problem:

$$\text{minimize} \quad -\log\det(X) + \mathbf{Tr}\, AX \tag{5.50}$$

where the optimization variable is $X \in \mathcal{S}^n_{++}$ and $A \in \mathcal{S}^n_{++}$.

Let $f : \mathcal{S}^n_{++} \to \mathbf{R}$ be the objective function, *i.e.*, $f(X) = -\log\det(X) + \mathbf{Tr}\, AX$. Now if we let $\mathbf{adj}(X) \in \mathcal{S}^n_{++}$ be the adjugate of X, *i.e.*, $\mathbf{adj}(X)_{ij} = (-1)^{i+j}\mathbf{minor}(X)_{ji}$ where $\mathbf{minor}(X)_{ij}$ is the (i,j)-minor of X, *i.e.*, the determinant of the $(n-1)$-by-$(n-1)$ matrix that results from deleting row i and column j of X, then we have $\det(X) = \sum_{j=1}^n X_{ij}\,\mathbf{adj}(X)_{ji}$. Since $\mathbf{adj}(X)_{ji}$ is not a function of X_{ij}, we have

$$\frac{\partial}{\partial X_{ij}}\det(X) = \mathbf{adj}(X)_{ji}. \tag{5.51}$$

Therefore

$$\frac{\partial}{\partial X_{ij}} f(X) = -\frac{\mathbf{adj}(X)_{ji}}{\det(X)} + A_{ji} = -\left(X^{-1}\right)_{ji} + A_{ji}. \tag{5.52}$$

Since $-\log\det(X)$ is a convex function in X as proved in XXX, $f(X)$ is a convex function, thus, since $\nabla_X f\left(A^{-1}\right) = 0$, $f(X)$ has its minimum value when $X = A^{-1}$.

Therefore the optimal solution for (5.50) is $X^* = A^{-1}$ and the optimal value is $\log\det(A) + n$.

5.5 Unconstrained minimization

5.5.1 Gradient descent method

5.5.1.1 Examples

5.5.1.1.1 A quadratic problem in $\mathbf{R}^2$ We consider the quadratic objective function on $\mathbf{R}^2$

$$f(x) = \frac{1}{2}(x_1^2 + \gamma x_2^2) \tag{5.53}$$

where $\gamma > 0$.

We apply the gradient descent method with exact line search. The gradient of f is

$$\nabla f(x) = \left[\begin{array}{c} x_1 \\ \gamma x_2 \end{array}\right] \tag{5.54}$$

Let $\tilde{f} : \mathbf{R}_+ \to \mathbf{R}$ defined by $\tilde{f}(t) = f(x - t\nabla f(x))$. Now

$$\tilde{f}(t) = f\left(\left[\begin{array}{c} (1-t)x_1 \\ (1-\gamma t)x_2 \end{array}\right]\right) = \frac{1}{2}\left((1-t)^2 x_1^2 + \gamma(1-\gamma t)^2 x_2^2\right) \tag{5.55}$$

and

$$\frac{d}{dt}\tilde{f}(t) = -(1-t)x_1^2 - \gamma^2(1-\gamma t)x_2^2 = 0 \tag{5.56}$$

implies

$$t = \frac{x_1^2 + \gamma^2 x_2^2}{x_1^2 + \gamma^3 x_2^2} \tag{5.57}$$

minimizes $\tilde{f}(t)$. Since

$$1 - t = \frac{\gamma^2(\gamma - 1)x_2^2}{x_1^2 + \gamma^3 x_2^2} \tag{5.58}$$

and

$$1 - \gamma t = \frac{(1-\gamma)x_1^2}{x_1^2 + \gamma^3 x_2^2} \tag{5.59}$$

Thus the exact line search yields

$$x^+ = x - t\nabla f(x) = \left[\begin{array}{c} (1-t)x_1 \\ (1-\gamma t)x_2 \end{array}\right] = \frac{(1-\gamma)x_1 x_2}{x_1^2 + \gamma^3 x_2^2}\left[\begin{array}{c} -\gamma^2 x_2 \\ x_1 \end{array}\right]. \tag{5.60}$$

If $x = \alpha[\gamma\ 1]^T$, then

$$x^+ = \frac{\alpha^3(1-\gamma)\gamma}{\alpha^2\gamma^2(1+\gamma)}\left[\begin{array}{c} -\gamma^2 \\ \gamma \end{array}\right] = \alpha\frac{1-\gamma}{1+\gamma}\left[\begin{array}{c} -\gamma \\ 1 \end{array}\right]. \tag{5.61}$$

If $x = \alpha[-\gamma\ 1]^T$, then

$$x^+ = -\frac{\alpha^3(1-\gamma)\gamma}{\alpha^2\gamma^2(1+\gamma)}\left[\begin{array}{c} -\gamma^2 \\ -\gamma \end{array}\right] = \alpha\frac{1-\gamma}{1+\gamma}\left[\begin{array}{c} \gamma \\ 1 \end{array}\right]. \tag{5.62}$$

Therefore if $x^{(0)} = [\gamma\ 1]^T$, then

$$x^{(k)} = \left(\frac{1-\gamma}{1+\gamma}\right)^k\left[\begin{array}{c} (-1)^k\gamma \\ 1 \end{array}\right] = \left(\frac{\gamma-1}{\gamma+1}\right)^k\left[\begin{array}{c} \gamma \\ (-1)^k \end{array}\right]. \tag{5.63}$$

Chapter 6

Portfolio optimization

6.1 Problem formulation

Suppose that we have n assets to invest on and that the return of each asset per unit invest is modeled by random variables R_i for $i = 1, \ldots, n$. Then we want to decide the amount of investment on each asset, $x_i \in \mathbf{R}$ for $i = 1, \ldots, n$, so that it optimizes the overall investment (in certain senses).

For formulization, we use the following definitions.

- Define a vector random variable $R \in \mathbf{R}^n$ such that

$$R = \begin{bmatrix} R_1 \\ \vdots \\ R_n \end{bmatrix} \in \mathbf{R}^n. \tag{6.1}$$

- Let $r \in \mathbf{R}^n$ be the expected value of R, *i.e.*,

$$r = \mathbf{E}(R) = \begin{bmatrix} \mathbf{E}(R_1) \\ \vdots \\ \mathbf{E}(R_n) \end{bmatrix} = \begin{bmatrix} r_1 \\ \vdots \\ r_n \end{bmatrix} \in \mathbf{R}^n. \tag{6.2}$$

- Define a vector $x \in \mathbf{R}$ which is an aggregate of the investments:

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbf{R}^n. \tag{6.3}$$

- Define a feasible set $\mathcal{X} \subseteq \mathbf{R}^n$ for x. For example, if we have a limit on the total investment,

$$\mathcal{X} = \{x \in \mathbf{R}^n \mid \sum_{i=1}^{n} c_i x_i \leq c_{\max}\}, \tag{6.4}$$

  or if we have the minimum and maximum amount to invest for each asset, we'd have

$$\mathcal{X} = \{x \in \mathbf{R}^n \mid d_{\min} \leq x_i \leq d_{\max} \text{ for } i = 1, \ldots, n\}. \tag{6.5}$$

  Generally, we'd prefer $\mathcal{X}$ to be a convex set, *i.e.*, for any $x, y \in \mathcal{X}$ and $0 \leq \lambda \leq 1$,

$$\lambda x + (1 - \lambda) y \in \mathcal{X}. \tag{6.6}$$

6.1.1 A portfolio optimization problem

A portfolio optimization problem can be formulized by

$$\begin{array}{ll} \text{maximize} & f(x) = \mathbf{E}(Z) \\ \text{minimize} & g(x) = \mathbf{Var}(Z) \\ \text{subject to} & x \in \mathcal{X} \end{array} \tag{6.7}$$

where the optimization variable is $x \in \mathbf{R}^n$ and

$$Z = \sum_{i=1}^{n} x_i R_i = x^T R \tag{6.8}$$

where $\mathbf{E}(\cdot)$ and $\mathbf{Var}(\cdot)$ refer to the expected value and the variance operators respectively.

This problem formulation tries to *maximize the expected return* while *minimizing the variance or uncertainty or risk*, which generally makes sense.

(6.4) (6.5)

Note that

$$\mathbf{E}(Z) = \mathbf{E}(x^T R) = \mathbf{E}\left(\sum_{i=1}^{n} x_i R_i\right) = \sum_{i=1}^{n} x_i \mathbf{E}(R_i) = \sum_{i=1}^{n} x_i r_i = r^T x \tag{6.9}$$

and

$$\begin{aligned} \mathbf{Var}(Z) = \mathbf{E}(Z - \mathbf{E}(Z))^2 &= \mathbf{E}\left(x^T R - x^T r\right)^2 \\ &= \mathbf{E}\left(x^T (R - r)\right)^2 = \mathbf{E}\left(x^T (R - r)(R - r)^T x\right) \\ &= x^T \mathbf{E}(R - r)(R - r)^T x = x^T \Sigma_R x \end{aligned}$$

where $\Sigma_R = \mathbf{E}(R - r)(R - r)^T$ is the covariance matrix of R. Note that $\Sigma_R \in \mathcal{S}_+^n$ since for any $y \in \mathbf{R}^n$,

$$y^T \Sigma_R y = y^T \mathbf{E}(R - r)(R - r)^T y = \mathbf{E}(x^T (R - r))^2 \geq 0. \tag{6.10}$$

Thus, (6.7) can be rewritten as

$$\begin{array}{ll} \text{maximize} & f(x) = r^T x \\ \text{minimize} & g(x) = x^T \Sigma_R x \\ \text{subject to} & x \in \mathcal{X}. \end{array} \tag{6.11}$$

Chapter 7

Part Appendix: Optimization

7.1 Some typical optimization problems

7.1.1 Strictly convex quadratic minimization

Consider a strictly convex quadratic minimization problem:

$$\text{minimize} \quad g(\alpha) = f(c + \alpha d) \tag{7.1}$$

where $\alpha \in \mathbf{R}$ is the optimization variable, $f : \mathbf{R}^n \to \mathbf{R}$ is defined by $f(x) = x^T Ax/2 - b^T x$ for some $A \in \mathcal{S}^n_{++}$ and $b \in \mathbf{R}^n$, $c \neq 0 \in \mathbf{R}^n$, and $d \in \mathbf{R}^n$. Then the optimization solution for (7.1) is given by

$$\alpha^* = \frac{d^T(b - Ac)}{d^T Ad}. \tag{7.2}$$

Proof: We show two different (but mathematically equivalent) proofs. First, we plug $c + \alpha d$ in the equation of f. Then

$$\begin{aligned} g(\alpha) &= \frac{1}{2}(c + \alpha d)^T A(c + \alpha d) - b^T(c + \alpha d) \\ &= \frac{1}{2}\alpha^2 d^T Ad - \alpha d^T(b - Ac) + \frac{1}{2}c^T Ac - c^T b \end{aligned}$$

Thus, since $c^T Ac > 0$, the optimization solution for (7.1) is what is given in (7.2).

Another method uses Theorem 2.4, which implies that

$$g'(\alpha) = \frac{d}{d\alpha}g(\alpha) = d^T\nabla f(c + \alpha d) = d^T(A(c + \alpha d) - b) = \alpha d^T Ad - d^T(b - Ac) \tag{7.3}$$

Since g is a strictly convex function in α, $g'(\alpha^*) = 0$, hence the optimization solution for (7.1) is what is given in (7.2).

Part III

Statistics

Chapter 8

Statistics Basics

8.1 Probability space and random variables

8.1.1 Correlation coefficients

The correlation coefficients of two random variables, X and Y, is defined by

$$\rho_{X,Y} = \frac{\mathbf{E}(X-\mu_X)(Y-\mu_Y)}{\sqrt{\mathbf{E}(X-\mu_X)^2\mathbf{E}(Y-\mu_Y)^2}} \tag{8.1}$$

8.2 Transformation of a random variable via a function

8.2.1 Scale random variable

Assume two random variables, $X \in \mathbf{R}$ and $Y \in \mathbf{R}$, which are related by a function $g : \mathbf{R} \to \mathbf{R} \in C^1$ such that

$$Y = g(X). \tag{8.2}$$

Now let's derive an equation for the probability density function (PDF) of Y given the PDF of X, $f_X : \mathbf{R} \to \mathbf{R}_+$.

The definition of cumulative distribution function (CDF) of Y implies that

$$F_Y(y) = \mathbf{Prob}\{Y \leq y\} = \mathbf{Prob}\{g(X) \leq y\} \tag{8.3}$$

for any $y \in \mathbf{R}$.

Now if we assume that g is a strictly increasing function, it has its inverse function $g^{-1} : g(\mathbf{R}) \to \mathbf{R}$ and (8.3) becomes

$$F_Y(y) = \mathbf{Prob}\{g(X) \leq y\} = \mathbf{Prob}\{X \leq g^{-1}(y)\} = F_X(g^{-1}(y))$$

for any $y \in g(\mathbf{R})$. Thus, we can differentiate both sides to have

$$f_Y(y) = \frac{d}{dy}F_Y(y) = \frac{d}{dy}F_X(g^{-1}(y)) = f_X(g^{-1}(y))\frac{d}{dy}g^{-1}(y) = \frac{1}{g'(g^{-1}(y))}f_X(g^{-1}(y)), \tag{8.4}$$

since (2.4) implies that

$$1 = \frac{d}{dx}g(g^{-1}(x)) = g'(g^{-1}(x))\frac{d}{dx}g^{-1}(x), \tag{8.5}$$

i.e., the derivative of the inverse function is the inverse of the derivative of the original function.

Now if we assume that g is a strictly decreasing function, we have

$$F_Y(y) = \mathbf{Prob}\{g(X) \leq y\} = \mathbf{Prob}\{x \geq g^{-1}(y)\} = 1 - F_X(g^{-1}(y)) + \mathbf{Prob}\{x = g^{-1}(y)\},$$

and

$$f_Y(y) = \frac{d}{dy}F_Y(y) = -\frac{d}{dy}F_X(g^{-1}(y)) = -f_X(g^{-1}(y))\frac{d}{dy}g^{-1}(y) = -\frac{1}{g'(g^{-1}(y))}f_X(g^{-1}(y)). \tag{8.6}$$

Since $g'(y) > 0$ for a strictly increasing function, and $g'(y) < 0$ for a strictly decreasing function, (8.4) and (8.6) imply

$$f_Y(y) = \frac{1}{|g'(g^{-1}(y))|} f_X(g^{-1}(y)) \tag{8.7}$$

for both cases.

Now consider a general function, g, *i.e.*, not necessarily a monotonic function. Suppose that $y \in g(\mathbf{R})$. Then for every $x \in \mathbf{R}$ such that $f(x) = y$, if $f'(x) \neq 0$, then $f(x)$ is locally strictly monotonic, *i.e.*, there exists $\delta > 0$ such that $f(x)$ is strictly monotonic for $x \in (x - \delta, x + \delta)$, hence (8.7) holds for such x.

The probability around y is a summation of the probabilities around such points, *i.e.*, the probabilities around all x such that $f(x) = y$ and $f'(x) \neq 0$. Therefore, we have

$$f_Y(y) = \sum_{x : g(x) = y} \frac{1}{|g'(x)|} f_X(x). \tag{8.8}$$

There is another way to derive the same equation (in a less strict way) which helps get more insight. Let's again suppose that g is a strictly increasing function. Then consider the probability that X lies in $(x, x + \Delta x)$. The probability should be the same as Y lies in $(y, y + \Delta y)$ where $y = g(x)$ and $\Delta y = g(x + \Delta x) - g(x)$, *i.e.*,

$$\begin{aligned} f_X(x)\Delta x &\approx \int_x^{x+\Delta x} f_X(x)\, dx = \mathbf{Prob}\{x \leq X \leq x + \Delta x\} \\ = &\quad \mathbf{Prob}\{y \leq Y \leq y + \Delta y\} = \int_y^{y+\Delta y} f_Y(y)\, dy \approx f_Y(y)\Delta y. \end{aligned}$$

The approximation becomes the equality when Δx goes to 0. Therefore we have

$$f_Y(y) = \lim_{\Delta x \to 0} \frac{\Delta x}{\Delta y} f_X(x) = \frac{1}{\lim_{\Delta x \to 0} \frac{g(x+\Delta x) - g(x)}{\Delta x}} f_X(x) = \frac{1}{g'(x)} f_X(x), \tag{8.9}$$

which is equivalent to (8.4). Following the very same argument as before will lead to (8.8), *i.e.*, applying the same method to strictly decreasing case, *etc.*

8.2.2 Multivariate random variable

8.2.3 Data Examples

Suppose that we have n random variables, $X_1, \ldots, X_n$ and they are independent and identically distributed Gaussian, $\mathcal{N}(0, 1)$. Then assume that a random variable, Y, is the sum of the X_i's, *i.e.*,

$$Y = \sum_{i=1}^n X_i = X_1 + \cdots + X_n \tag{8.10}$$

Then the covariance of X_i and Y for each i is

$$\mathbf{Cov}(X_i, Y) = \mathbf{E}(X_i - \mathbf{E}X_i)(Y - \mathbf{E}Y) = \mathbf{E}\left(\sum_{j=1}^n X_i X_j\right) = 1 \tag{8.11}$$

and the variance of Y is

$$\mathbf{Var}(Y) = \mathbf{E}(Y - \mathbf{E}Y)^2 = \mathbf{E}\left(\sum_{j=1}^{n} X_i X_j\right)^2 = \sum_{i=1}^{n} \mathbf{E}X_i^2 = n. \tag{8.12}$$

Hence, the correlation coefficient of X_i and Y for each i is

$$\rho_{X_i,Y} = \mathbf{Cov}(X_i, Y)/\sqrt{\mathbf{Var}X_i\mathbf{Var}Y} = 1/\sqrt(n). \tag{8.13}$$

Therefore Y has clear relation with X_i's, but each correlation coefficient can be arbitrarily small as n grows!

8.3 Empirical Cumulative Distribution Function

Suppose that we have n (scalar) data points, $x_1, \ldots, x_n \in \mathbf{R}$. We assume that they are from some probability distribution of a random variable X. The empirical cumulative distribution function (ECDF) is one of the non-parameterized estimation methods to estimate the distribution of the original random variable, X.

The ECDF is defined by

$$\hat{F}_n(x) = \frac{1}{n}\sum_{i=1}^{n} u(x - x_i) \tag{8.14}$$

where the step function $u : \mathbf{R} \to \mathbf{R}$ is defined by

$$u(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{8.15}$$

Under some mild conditions, this ECDF converges to the cumulative distribution function (CDF) of the original random variable, *i.e.*,

$$\lim_{n\to\infty} F_n(x) = F_X(x) \tag{8.16}$$

where $F_X : \mathbf{R} \to \mathbf{R}$ is the true cumulative distribution function, *i.e.*,

$$F_X(x) = \mathbf{Prob}\{X \leq x\}. \tag{8.17}$$

8.3.1 Mixture distribution of ECDF

Suppose that we have m set of data $\{x_{ji}\}_{i=1}^{n_j}$ where $j = 1, \ldots, m$ and n_j refers to the size of the jth data set.

If we assume that the jth data set is drawn from a random variable X_j, the jth ECDF is an approximate of the jth true CDF, *i.e.*,

$$\hat{F}_j(x) = \frac{1}{n_j}\sum_{i=1}^{n_j} u(x - x_{ji}) \sim F_{X_j}(x). \tag{8.18}$$

Now let us figure out what is the relationship between the ECDF formed by all these data points and the m original CDFs. Note that there are $\sum_{j=1}^m n_j$ data points in total.

It turns out that the ECDF formed by all data points is an approximate for the mixture distribution of X_1, …, X_m where the jth mixture probability is $\pi_j = n_j / \sum_{k=1}^m n_k$. Note that $\sum_{j=1}^m \pi_j = 1$.

To be more precise, let us defined a random variable $\tilde{X}$ the probability density function (PDF) of which is defined by

$$f_{\tilde{X}}(x) = \sum_{j=1}^m \pi_j f_{X_j}(x). \tag{8.19}$$

Then the CDF of $\tilde{X}$ is

$$F_{\tilde{X}}(x) = \int_{-\infty}^x f_{\tilde{X}}(\tilde{x}) d\tilde{x} = \sum_{j=1}^m \pi_j \int_{-\infty}^x f_{X_j}(\tilde{x}) d\tilde{x} = \sum_{j=1}^m \pi_j F_{X_j}(x). \tag{8.20}$$

Now the ECDF obtained from *all* $\sum_{j=1}^m n_j$ data points is (by definition)

$$\begin{aligned}
\hat{F}(x) &= \frac{1}{\sum_{j=1}^m n_j} \sum_{j=1}^m \sum_{i=1}^{n_j} u(x - x_{ji}) = \frac{1}{\sum_{j=1}^m n_j} \sum_{j=1}^m \frac{n_j}{n_j} \sum_{i=1}^{n_j} u(x - x_{ji}) \\
&= \sum_{j=1}^m \frac{n_j}{\sum_{j=1}^m n_j} \left(\frac{1}{n_j} \sum_{i=1}^{n_j} u(x - x_{ji}) \right) \\
&= \sum_{j=1}^m \pi_j \hat{F}_j(x).
\end{aligned}$$

Since $\lim_{n_j \to \infty} \hat{F}_j(x) = F_{X_j}(x)$ for each j,

$$\lim_{n_1, \ldots, n_j \to \infty} \hat{F}(x) = \sum_{j=1}^m \pi_j F_{X_j}(x) = \hat{F}_{\tilde{X}}(x), \tag{8.21}$$

i.e., the ECDF converges to the CDF of $\tilde{X}$.

Therefore the ECDF formed by all the data points $\{x_{ji}\}_{i=1}^{n_j}$ is the ECDF for the mixture distribution defined by (8.19).

8.3.2 ECDF of mixture distribution

We discussed what distribution a mixture of data points from different distributions represent. Here we try to do (kind of) converse, *i.e.*, given arbitrary mixture probabilities, π_j $(j = 1, \ldots, m)$, form ECDF approximating the following mixture distribution.

$$\sum_{j=1}^m \pi_j f_{X_j}(x). \tag{8.22}$$

It can be proved that the following (modified) ECDF approximates the mixture distribution in (8.22).

$$\hat{F}(x) = \sum_{j=1}^{m} \frac{\pi_j}{n_j} \sum_{i=1}^{n_j} u(x - x_{ji}). \tag{8.23}$$

8.4 Various distributions

8.4.1 Gaussian distribution

The Gaussian (or normal) distribution is a type of continuous probability distribution for a real-valued random variable or real-valued random variables. The PDF of (vector) Gaussian random variable $X \in \mathbf{R}^n$ with mean $\mu \in \mathbf{R}^n$ and covariance matrix $\Sigma \in \mathcal{S}_{++}^n$ is defined by $f_X : \mathbf{R}^n \to \mathbf{R}_+$ where

$$f_X(x) = \frac{1}{(2\pi)^{n/2} \det(\Sigma)^{1/2}} \exp\left(-\frac{1}{2} (x-\mu)^T \Sigma^{-1} (x-\mu) \right). \tag{8.24}$$

We use the notation $X \sim \mathcal{N}(\mu, \Sigma)$ to denote that X is a Gaussian random variable with with mean $\mu \in \mathbf{R}^n$ and covariance matrix $\Sigma \in \mathcal{S}_{++}^n$.

If $n = 1$, this reduces to a scalar Gaussian random variable the PDF of which is given by

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-(x-\mu)^2 / 2\sigma^2 \right) \tag{8.25}$$

where $\mu \in \mathbf{R}$ is the mean and σ^2 is the variance.

In a statistical inference or machine learning context, we are sometimes interested in log-likelihood. To evaluate the log-likelihood, we need to take logarithm on the PDF. For Gaussian random variable, this quantity is given by

$$\log f_X(x) = -\frac{1}{2} (x-\mu)^T \Sigma^{-1} (x-\mu) - \frac{n}{2} \log(2\pi) - \frac{1}{2} \log\det(\Sigma). \tag{8.26}$$

8.4.2 Log-normal distribution

We say Y is log-normally distributed, if, for $X \sim \mathcal{N}(\mu_X, \sigma_X^2)$,

$$Y = \exp(X). \tag{8.27}$$

Then (**??**) implies that

$$\begin{aligned} f_Y(y) &= \frac{1}{\exp(\log(y))} \cdot \frac{1}{\sqrt{2\pi}\sigma_X} \exp\left(-\frac{(\log(y) - \mu_X)^2}{2\sigma_X^2} \right) \\ &= \frac{1}{\sqrt{2\pi}\sigma_X y} \exp\left(-\frac{(\log(y) - \mu_X)^2}{2\sigma_X^2} \right). \end{aligned} \tag{8.28}$$

8.4.2.1 Some statistics

The definition of the expected value implies

$$
\begin{aligned}
\mathbf{E}Y = \int_0^\infty y f_Y(y)\,dy &= \int_0^\infty \frac{1}{\sqrt{2\pi}\sigma_X} \exp\left(-\frac{(\log(y)-\mu_X)^2}{2\sigma_X^2}\right) dy \\
&= \int_{-\infty}^\infty \frac{1}{\sqrt{2\pi}\sigma_X} \exp\left(-\frac{(x-\mu_X)^2}{2\sigma_X^2}\right) \exp(x)\,dx \\
&= \int_{-\infty}^\infty \frac{1}{\sqrt{2\pi}\sigma_X} \exp\left(-\frac{(x-\mu_X)^2}{2\sigma_X^2} + x\right) dx \\
&= \int_{-\infty}^\infty \frac{1}{\sqrt{2\pi}\sigma_X} \exp\left(-\frac{x^2 - 2(\mu_X+\sigma_X^2)x + \mu_X^2}{2\sigma_X^2}\right) dx \\
&= \int_{-\infty}^\infty \frac{1}{\sqrt{2\pi}\sigma_X} \exp\left(-\frac{(x-(\mu_X+\sigma_X^2))^2 + \mu_X^2 - (\mu_X+\sigma_X^2)^2}{2\sigma_X^2}\right) dx \\
&= \int_{-\infty}^\infty \frac{1}{\sqrt{2\pi}\sigma_X} \exp\left(-\frac{(x-(\mu_X+\sigma_X^2))^2 - 2\mu_X\sigma_X^2 - \sigma_X^4}{2\sigma_X^2}\right) dx \\
&= \int_{-\infty}^\infty \frac{1}{\sqrt{2\pi}\sigma_X} \exp\left(-\frac{(x-(\mu_X+\sigma_X^2))^2 - \sigma_X^2(2\mu_X+\sigma_X^2)}{2\sigma_X^2}\right) dx \\
&= \exp\left(\frac{2\mu_X+\sigma_X^2}{2}\right) \int_{-\infty}^\infty \frac{1}{\sqrt{2\pi}\sigma_X} \exp\left(-\frac{(x-(\mu_X+\sigma_X^2))^2}{2\sigma_X^2}\right) dx \\
&= \exp\left(\frac{2\mu_X+\sigma_X^2}{2}\right)
\end{aligned}
$$

since $dy = \exp(x)dx$ and $\frac{1}{\sqrt{2\pi}\sigma_X}\exp\left(-\frac{(x-(\mu_X+\sigma_X^2))^2}{2\sigma_X^2}\right)$ is the PDF of a random variable $\sim \mathcal{N}(\mu_X + \sigma_X^2, \sigma_X^2)$, thus

$$
\mu_Y = \mathbf{E}Y = \exp\left(\frac{2\mu_X+\sigma_X^2}{2}\right). \tag{8.29}
$$

Similarly,

$$
\begin{aligned}
\mathbf{E}Y^2 = \int_0^\infty y^2 f_Y(y)\,dy = \int_0^\infty \frac{1}{\sqrt{2\pi}\sigma_X} y \exp\left(-\frac{(\log(y)-\mu_X)^2}{2\sigma_X^2}\right) dy \\
&= \int_{-\infty}^\infty \frac{1}{\sqrt{2\pi}\sigma_X} \exp(x)\exp\left(-\frac{(x-\mu_X)^2}{2\sigma_X^2}\right)\exp(x)\,dx \\
&= \int_{-\infty}^\infty \frac{1}{\sqrt{2\pi}\sigma_X} \exp\left(-\frac{(x-\mu_X)^2}{2\sigma_X^2}+2x\right) dx \\
&= \int_{-\infty}^\infty \frac{1}{\sqrt{2\pi}\sigma_X} \exp\left(-\frac{x^2-2(\mu_X+2\sigma_X^2)x+\mu_X^2}{2\sigma_X^2}\right) dx \\
&= \int_{-\infty}^\infty \frac{1}{\sqrt{2\pi}\sigma_X} \exp\left(-\frac{(x-(\mu_X+2\sigma_X^2))^2+\mu_X^2-(\mu_X+2\sigma_X^2)^2}{2\sigma_X^2}\right) dx \\
&= \int_{-\infty}^\infty \frac{1}{\sqrt{2\pi}\sigma_X} \exp\left(-\frac{(x-(\mu_X+2\sigma_X^2))^2-4\mu_X\sigma_X^2-4\sigma_X^4}{2\sigma_X^2}\right) dx \\
&= \int_{-\infty}^\infty \frac{1}{\sqrt{2\pi}\sigma_X} \exp\left(-\frac{(x-(\mu_X+2\sigma_X^2))^2-4\sigma_X^2(\mu_X+\sigma_X^2)}{2\sigma_X^2}\right) dx \\
&= \exp\left(2(\mu_X+\sigma_X^2)\right)\int_{-\infty}^\infty \frac{1}{\sqrt{2\pi}\sigma_X} \exp\left(-\frac{(x-(\mu_X+2\sigma_X^2))^2}{2\sigma_X^2}\right) dx \\
&= \exp\left(2(\mu_X+\sigma_X^2)\right),
\end{aligned}
$$

thus

$$\sigma_Y^2 = \mathbf{Var}(Y) = \mathbf{E}Y^2-(\mathbf{E}Y)^2 = \exp(2(\mu_X+\sigma_X^2))-\exp(2\mu_X+\sigma_X^2) = (\exp(\sigma_X^2)-1)\exp(2\mu_X+\sigma_X^2)). \tag{8.30}$$

Note that (8.29) implies that

$$\mu_Y^2 = \exp(2\mu_X+\sigma_X^2)), \tag{8.31}$$

hence

$$\sigma_Y^2 = (\exp(\sigma_X^2)-1)\mu_Y^2. \tag{8.32}$$

This multiplicative dependency of the standard deviation on the expected value is attributed to the fact that $\log(Y)\sim\mathcal{N}(\mu_X,\sigma_X^2)$, *i.e.*, the log-scale of Y follows the normal distribution.

Now if we differentiate the PDF with respect to y, (8.28) implies that

$$
\begin{aligned}
\frac{d}{dy}f_Y(y) = \frac{d}{dy}\left(\frac{1}{\sqrt{2\pi}\sigma_X y}\exp\left(-\frac{(\log(y)-\mu_X)^2}{2\sigma_X^2}\right)\right) \\
&= -\frac{1}{\sqrt{2\pi}\sigma_X y^2}\exp\left(-\frac{(\log(y)-\mu_X)^2}{2\sigma_X^2}\right) \\
&\quad +\frac{1}{\sqrt{2\pi}\sigma_X y}\exp\left(-\frac{(\log(y)-\mu_X)^2}{2\sigma_X^2}\right)\left(-\frac{(\log(y)-\mu_X)}{\sigma_X^2}\right)\frac{1}{y} \\
&= -\frac{1}{\sqrt{2\pi}\sigma_X y^2}\exp\left(-\frac{(\log(y)-\mu_X)^2}{2\sigma_X^2}\right)\left(1+\frac{(\log(y)-\mu_X)}{\sigma_X^2}\right) \\
&= -\frac{1}{\sqrt{2\pi}\sigma_X^3 y^2}\exp\left(-\frac{(\log(y)-\mu_X)^2}{2\sigma_X^2}\right)\left(\log(y)-(\mu_X-\sigma_X^2)\right).
\end{aligned}
$$

Equating the derivative to zero yields

$$y = \exp(\mu_X - \sigma_X^2), \tag{8.33}$$

which is the mode of Y.

8.4.2.2 Parameter estimation

Now assume that we have a log-normally distributed random variable, $Y \in \mathbf{R}_{++}$, with μ_Y and σ_Y^2 as its mean and variance. We derived the parameters of the source distribution, μ_X and σ_X.

The two equations, (8.29) and (8.30), imply

$$\begin{aligned} \mu_Y &= \exp(\mu_X + \sigma_X^2/2), \\ \sigma_Y^2 &= (\exp(\sigma_X^2) - 1)\exp(2\mu_X + \sigma_X^2) = (\exp(\sigma_X^2) - 1)\mu_Y^2, \end{aligned}$$

thus

$$\begin{aligned} \sigma_X^2 &= \log(1 + \sigma_Y^2/\mu_Y^2), \\ \mu_X &= \log(\mu_Y) - \sigma_X^2/2 = \log(\mu_Y) - \log(1 + \sigma_Y^2/\mu_Y^2)/2 = \frac{1}{2}\log\left(\frac{\mu_Y^2}{1 + \sigma_Y^2/\mu_Y^2}\right). \end{aligned}$$

8.5 Exponential family

In probability and statistics, an exponential family is a parametric set of probability distributions of a certain form. This special form is chosen for mathematical convenience, based on some useful algebraic properties, as well as for generality, as exponential families are in a sense very natural sets of distributions to consider. The term exponential class is sometimes used in place of *exponential family*, or the older term Koopman–Darmois family. The terms *distribution* and *family* are often used loosely: properly, an exponential family is a set of distributions, where the specific distribution varies with the parameter; however, a parametric family of distributions is often referred to as *a distribution* (like *the normal distribution*, meaning *the family of normal distributions*) and the set of all exponential families is sometimes loosely referred to as *the* exponential family.

The concept of exponential families is credited to E. J. G. Pitman, G. Darmois, and B. O. Koopman in 1935–1936. Exponential families of distributions provides a general framework for selecting a possible alternative parameterisation of a parametric family of distributions, in terms of natural parameters, and for defining useful sample statistics, called the natural sufficient statistics of the family.

The exponential family of distributions over $x \in \mathbf{R}^n$ given parameters $\eta \in \mathbf{R}^m$ is defined to be the set of distributions of the form

$$p(x|\eta) = g(\eta)h(x)\exp(\eta^T u(x)) \tag{8.34}$$

for some functions $g : \mathbf{R}^m \to \mathbf{R}_+$, $h : \mathbf{R}^n \to \mathbf{R}_+$, and $u : \mathbf{R}^n \to \mathbf{R}^m$. Note that the requirement for the PDF implies

$$\int p(x|\eta)dx = g(\eta)\int h(x)\exp(\eta^T u(x))dx = 1 \Leftrightarrow g(\eta) = \left(\int h(x)\exp(\eta^T u(x))dx\right)^{-1}. \tag{8.35}$$

The exponential family of distributions covers many distributions as shown below.

8.5.1 Categorical distribution

For $\mu_i > 0$ with $\sum_{i=1}^n \mu_i = 1$, the PMF of the categorical distribution is given by

$$p(x|\mu) = \prod_{i=1}^n \mu_i^{x_i} = \mu_n^{1-\sum_{j=1}^{n-1} x_j} \prod_{i=1}^{n-1} \mu_i^{x_i} = \mu_n \prod_{i=1}^{n-1} (\mu_i/\mu_n)^{x_i} \tag{8.36}$$

where $x \in \{0,1\}^{n-1}$ with $0 \leq \sum_{i=1}^{n-1} x_i \leq 1$. Since

$$p(x|\mu) = \mu_n \exp\left(\sum_{i=1}^{n-1} x_i \log(\mu_i/\mu_n)\right), \tag{8.37}$$

we could let $\eta_i = \log(\mu_i/\mu_n)$ for $i = 1, \ldots, n-1$ and figure out how we can express μ_n in terms of $\eta \in \mathbf{R}^{n-1}$. However, here we use (8.35) for that. Now let

$$p(x|\eta) = g(\eta)\exp(\eta^T x). \tag{8.38}$$

Then

$$g(\eta) \sum_{0 \leq \mathbf{1}^T x \leq 1} \exp(\eta^T x) = g(\eta)\left(1 + \sum_{i=1}^{n-1} \exp(\eta_i)\right) = 1, \tag{8.39}$$

thus $g(\eta) = \left(1 + \sum_{i=1}^{n-1} \exp(\eta_i)\right)^{-1}$. Therefore the categorical distribution using the standard representation for the exponential family (8.34) is given by

$$p(x|\eta) = \frac{\exp(\eta^T x)}{1 + \sum_{i=1}^{n-1} \exp(\eta_i)} \tag{8.40}$$

with

$$\begin{aligned} \eta_i &= \log\left(\frac{\mu_i}{1 - \mathbf{1}^T \mu}\right) \text{ for } i = 1, \ldots, n-1 && (8.41) \\ u(x) &= x && (8.42) \\ g(\eta) &= \left(1 + \sum_{i=1}^{n-1} \exp(\eta_i)\right)^{-1} && (8.43) \\ h(x) &= 1. && (8.44) \end{aligned}$$

8.5.2 Bernoulli distribution

The Bernoulli distribution is a categorical distribution with $n = 2$. Therefore (8.40) implies that the Bernoulli distribution using the standard representation for the exponential family (8.34) is given by

$$p(x|\eta) = \frac{\exp(\eta x)}{1 + \exp(\eta)} \tag{8.45}$$

where $x \in \{0, 1\}$ and $\eta \in \mathbf{R}$ with

$$\begin{aligned} \eta &= \log\left(\frac{\mu}{1-\mu}\right) && (8.46)\\ u(x) &= x && (8.47)\\ g(\eta) &= (1+\exp(\eta))^{-1} = \sigma(-\eta) && (8.48)\\ h(x) &= 1 && (8.49) \end{aligned}$$

where σ is the logistic sigmoid function.

8.5.3 Gaussian distribution

The PDF of Gaussian random variable with mean $\mu \in \mathbf{R}^n$ and covariance matrix $\Sigma \in \mathcal{S}^n_{++}$ is given by (8.24). We can rewrite it as

$$p(x|\mu, \Sigma) = \frac{1}{(2\pi)^{n/2}\det(\Sigma)^{1/2}} \exp\left(-\frac{1}{2}x^T\Sigma^{-1}x + \mu^T\Sigma^{-1}x - \frac{1}{2}\mu^T\Sigma^{-1}\mu\right). \qquad (8.50)$$

If we let $\eta_1 = -\frac{1}{2}\Sigma^{-1}$ and $\eta_2 = \Sigma^{-1}\mu$, then

$$\begin{aligned} p(x|\eta) &= \frac{\det(-2\eta_1)^{1/2}}{(2\pi)^{n/2}} \exp\left(\mathbf{Tr}\,\eta_1 x x^T + \eta_2^T x - \frac{1}{2}\eta_2^T(-2\eta_1)^{-1}\eta_2\right)\\ &= \frac{2^{n/2}\det(-\eta_1)^{1/2}}{(2\pi)^{n/2}} \exp\left(\mathbf{Tr}\,\eta_1 x x^T + \eta_2^T x + \frac{1}{4}\eta_2^T\eta_1^{-1}\eta_2\right). \end{aligned}$$

Therefore the Gaussian distribution using the standard representation for the exponential family (8.34) is given by

$$p(x|\eta) = \frac{\det(-\eta_1)^{1/2}}{\pi^{n/2}} \exp\left(\frac{1}{4}\eta_2^T\eta_1^{-1}\eta_2\right) \exp\left(\mathbf{Tr}\,\eta_1 x x^T + \eta_2^T x\right) \qquad (8.51)$$

with

$$\begin{aligned} \eta &= (\eta_1, \eta_2) = \left(-\frac{1}{2}\Sigma^{-1}, \Sigma^{-1}\mu\right) \in (-\mathcal{S}^n_{++}) \times \mathbf{R}^n && (8.52)\\ u(x) &= \left(x x^T, x\right) \in \mathcal{S}^n_{+} \times \mathbf{R}^n && (8.53)\\ g(\eta) &= \frac{\det(-\eta_1)^{1/2}}{\pi^{n/2}} \exp\left(\frac{1}{4}\eta_2^T\eta_1^{-1}\eta_2\right) && (8.54)\\ h(x) &= 1. && (8.55) \end{aligned}$$

Chapter 9

Bayesian Statistics

9.1 Bayesian Theorem

Suppose that A and B are two events with $P(B) \neq 0$. Then

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}. \tag{9.1}$$

This is called *Bayesian theorem*.

9.2 Bayesian Inference

$$p(\theta|X) = \frac{p(X|\theta)p(\theta)}{p(X)} = \frac{p(X|\theta)p(\theta)}{\int p(X|\theta)p(\theta)d\theta} \propto p(X|\theta)p(\theta) = \text{likelihood} \times \text{prior} \tag{9.2}$$

9.3 Conjugate prior

9.3.1 Bernoulli distribution

For Bernoulli distribution, the conjugate prior is the beta distribution.

$$p(\theta) = \frac{1}{B(a,b)}\theta^{a-1}(1-\theta)^{b-1} \tag{9.3}$$

Then the posterior probability can be expressed as

$$\begin{aligned} p(\theta|X) &\propto p(X|\theta)p(\theta) \propto \left(\prod_{i=1}^{n} \theta^{I(x_i=1)}(1-\theta)^{I(x_i=0)}\right)\theta^{a-1}(1-\theta)^{b-1} \\ &= \theta^{k+a-1}(1-\theta)^{n-k+b-1} \end{aligned}$$

where k is the number of 1s in X. Thus $p(\theta|X) \sim \text{Beta}(k+a, n-k+b)$, *i.e.*,

$$p(\theta|X) = \frac{1}{B(k+a, n-k+b)}\theta^{k+a-1}(1-\theta)^{n-k+b-1}. \tag{9.4}$$

9.3.2 Gaussian distribution

Suppose that $X \sim \mathcal{N}(\mu, \tau^{-1})$ and $\mu \sim \mathcal{N}(m, \lambda^{-1})$. Then the posterior distribution is

$$p(\mu|X) \sim \mathcal{N}\left(\frac{\tau\sum_{i=1}^{n} x_i + \lambda m}{n\tau + \lambda}, (n\tau+\lambda)^{-1}\right) \tag{9.5}$$

Chapter 10

Information Theory

10.1 Basics

10.1.1 Entropy

10.1.2 Mutual Information

The mutual information (MI) is defined by

$$I(X;Y) = \mathbf{E}\log\frac{f_{X,Y}(X,Y)}{f_X(X)f_Y(Y)} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\log\frac{f_{X,Y}(x,y)}{f_X(x)f_Y(y)}\,dxdy \tag{10.1}$$

10.1.3 Relative Entropy (Kullback–Leibler divergence)

The relative entropy of the two random distributions, p and q, is defined by

$$D(p\|q) = \mathbf{E}_p\log\left(\frac{p(X)}{q(X)}\right). \tag{10.2}$$

The relative entropy represents the difference measure between the two distributions. Unlike the mutual information, relative entropy is asymmetric, *i.e.*, in general, $D(p\|q) \neq D(q\|p)$. It is always nonnegative quantity since the Jensen's inequality implies that

$$-D(p\|q) = \mathbf{E}_p\log\left(\frac{q(X)}{p(X)}\right) \leq \log\left(\mathbf{E}_p\left(\frac{q(X)}{p(X)}\right)\right) = \log\left(\int_{-\infty}^{\infty}q(x)\,dx\right) = 0. \tag{10.3}$$

The relative entropy can be rewritten as

$$D(p\|q) = -\mathbf{E}_p\log q(X) + \mathbf{E}_p\log p(X) = -\mathbf{E}_p\log q(X) - H(p) \tag{10.4}$$

where the first term, $-\mathbf{E}_p\log q(X)$ is called the *cross-entropy* of p and q.

Part IV

Machine Learning

Chapter 11

Machine Learning Basics

11.1 Optimal Predictor

Consider a regression problem where we predict $Y \in \mathbf{R}^m$ given $X \in \mathbf{R}^n$. We want to design a predictor $g : \mathbf{R}^n \to \mathbf{R}^m$ so that $g(X) \sim Y$ in some statistical sense. We first show that $g(X) = \mathbf{E}(Y|X)$ is the optimal predictor (or estimator) in least-mean-square sense.

We define $g^* : \mathbf{R}^n \to \mathbf{R}^m$ where $g(x) = \mathbf{E}(Y|X = x)$. Then

$$\begin{aligned}
&\mathbf{E}\|g(X) - Y\|_2^2 = \mathbf{E}\|g(X) - g^*(X) + g^*(X) - Y\|_2^2 \\
&= \quad \mathbf{E}\|g(X) - g^*(X)\|_2^2 + \mathbf{E}\|g^*(X) - Y\|_2^2 + 2\mathbf{E}(g(X) - g^*(X))^T (g^*(X) - Y) \\
&= \quad \mathbf{E}\|g(X) - g^*(X)\|_2^2 + \mathbf{E}\|g^*(X) - Y\|_2^2 + 2\mathbf{E}_X \mathbf{E}_Y \left((g(X) - g^*(X))^T (g^*(X) - Y)|X\right) \\
&= \quad \mathbf{E}\|g(X) - g^*(X)\|_2^2 + \mathbf{E}\|g^*(X) - Y\|_2^2 + 2\mathbf{E}_X (g(X) - g^*(X))^T \mathbf{E}_Y \left(g^*(X) - Y|X\right) \\
&= \quad \mathbf{E}\|g(X) - g^*(X)\|_2^2 + \mathbf{E}\|g^*(X) - Y\|_2^2 + 2\mathbf{E}_X (g(X) - g^*(X))^T \left(g^*(X) - \mathbf{E}(Y|X)\right) \\
&= \quad \mathbf{E}\|g(X) - g^*(X)\|_2^2 + \mathbf{E}\|g^*(X) - Y\|_2^2 \geq \mathbf{E}\|g^*(X) - Y\|_2^2.
\end{aligned}$$

Therefore $g^*(X)$ is the optimal predictor for Y in the least-mean-square sense.

11.2 Bias and Variance

In §11.1, we proved that $g^*(X) = \mathbf{E}(Y|X)$ is the optimal predictor (or estimator) in the least-mean-square sense. However, unless we have the full knowledge of the joint probability distribution of X and Y, *i.e.*, $p(X, Y)$, or know $\mathbf{E}(Y|X = x)$ as a function of x, it is not possible to obtain g^*.

Here we assume that we obtain the predictor for Y given X from a dataset D where

$$D = \{(x_1, y_1), \ldots, (x_N, y_N)\} \subseteq \mathbf{R}^n \times \mathbf{R}^m.^1 \tag{11.1}$$

Now suppose that we have a predictor $g(\cdot; D) : \mathbf{R}^n \to \mathbf{R}^m$, which depends on D. Now let $\mathcal{D}$ denote the random variable for this data set, *i.e.*,

$$\mathcal{D} = \{(X_1, Y_1), \ldots, (X_N, Y_N)\} \subseteq \mathbf{R}^n \times \mathbf{R}^m. \tag{11.2}$$

[1] Note that strictly speaking, $\mathcal{D}$ is *not* a set since the order of $(x_i, y_i) \in \mathbf{R}^n \times \mathbf{R}^m$ matters, *i.e.*, if the order is changed, we generally have different predictor, and we are allowed to have identical data point. Thus, we should say $\mathcal{D}$ is a (ordered) list of points, $(x_i, y_i) \in \mathbf{R}^n \times \mathbf{R}^m$.

Then the mean square error of this predictor can be decomposed as following.

$$
\begin{aligned}
&\mathbf{E}_{X,Y,\mathcal{D}}\|g(X;\mathcal{D})-Y\|_2^2 = \mathbf{E}_{X,Y,\mathcal{D}}\|g(X;\mathcal{D})-g^*(X)+g^*(X)-Y\|_2^2\\
&= \mathbf{E}_{X,Y,\mathcal{D}}\|g(X;\mathcal{D})-g^*(X)\|_2^2+\mathbf{E}_{X,Y,\mathcal{D}}\|g^*(X)-Y\|_2^2\\
&\quad +2\mathbf{E}_{X,Y,\mathcal{D}}(g(X;\mathcal{D})-g^*(X))^T(g^*(X)-Y)\\
&= \mathbf{E}_{X,\mathcal{D}}\|g(X;\mathcal{D})-g^*(X)\|_2^2+\mathbf{E}_{X,Y}\|g^*(X)-Y\|_2^2\\
&\quad +2\mathbf{E}_{X,\mathcal{D}}\mathbf{E}_{Y}\left((g(X;\mathcal{D})-g^*(X))^T(g^*(X)-Y)|X,\mathcal{D}\right)\\
&= \mathbf{E}_{X,\mathcal{D}}\|g(X;\mathcal{D})-g^*(X)\|_2^2+\mathbf{E}_{X,Y}\|g^*(X)-Y\|_2^2\\
&\quad +2\mathbf{E}_{X,\mathcal{D}}(g(X;\mathcal{D})-g^*(X))^T\mathbf{E}_{Y}\left(g^*(X)-Y|X,\mathcal{D}\right)\\
&= \mathbf{E}_{X,\mathcal{D}}\|g(X;\mathcal{D})-g^*(X)\|_2^2+\mathbf{E}_{X,Y}\|g^*(X)-Y\|_2^2\\
&\quad +2\mathbf{E}_{X,\mathcal{D}}(g(X;\mathcal{D})-g^*(X))^T\mathbf{E}_{Y}\left(g^*(X)-Y|X\right)\\
&= \mathbf{E}_{X,\mathcal{D}}\|g(X;\mathcal{D})-g^*(X)\|_2^2+\mathbf{E}_{X,Y}\|g^*(X)-Y\|_2^2\\
&\quad +2\mathbf{E}_{X,\mathcal{D}}(g(X;\mathcal{D})-g^*(X))^T(g^*(X)-\mathbf{E}_{Y}(Y|X))\\
&= \mathbf{E}_{X,\mathcal{D}}\|g(X;\mathcal{D})-\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})+\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})-g^*(X)\|_2^2+\mathbf{E}_{X,Y}\|g^*(X)-Y\|_2^2\\
&= \mathbf{E}_{X,\mathcal{D}}\|g(X;\mathcal{D})-\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})\|_2^2+\mathbf{E}_{X,\mathcal{D}}\|\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})-g^*(X)\|_2^2+\mathbf{E}_{X,Y}\|g^*(X)-Y\|_2^2\\
&\quad +2\mathbf{E}_{X,\mathcal{D}}(g(X;\mathcal{D})-\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D}))^T(\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})-g^*(X))\\
&= \mathbf{E}_{X,\mathcal{D}}\|g(X;\mathcal{D})-\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})\|_2^2+\mathbf{E}_{X}\|\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})-g^*(X)\|_2^2+\mathbf{E}_{X,Y}\|g^*(X)-Y\|_2^2\\
&\quad +2\mathbf{E}_{X}\mathbf{E}_{\mathcal{D}}\left((g(X;\mathcal{D})-\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D}))^T(\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})-g^*(X))|X\right)\\
&= \mathbf{E}_{X,\mathcal{D}}\|g(X;\mathcal{D})-\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})\|_2^2+\mathbf{E}_{X}\|\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})-g^*(X)\|_2^2+\mathbf{E}_{X,Y}\|g^*(X)-Y\|_2^2\\
&\quad +2\mathbf{E}_{X}\mathbf{E}_{\mathcal{D}}(g(X;\mathcal{D})-\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})|X)^T(\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})-g^*(X))\\
&= \mathbf{E}_{X,\mathcal{D}}\|g(X;\mathcal{D})-\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})\|_2^2+\mathbf{E}_{X}\|\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})-g^*(X)\|_2^2+\mathbf{E}_{X,Y}\|g^*(X)-Y\|_2^2\\
&\quad +2\mathbf{E}_{X}(\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})-\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})|X)^T(\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})-g^*(X))\\
&= \mathbf{E}_{X,\mathcal{D}}\|g(X;\mathcal{D})-\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})\|_2^2+\mathbf{E}_{X}\|\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})-g^*(X)\|_2^2+\mathbf{E}_{X,Y}\|g^*(X)-Y\|_2^2.
\end{aligned}
$$

Note that we use the fact that $\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})$ is a function of X only (hence does not depend on X).

In the last equation, the first term is called the *variance* since it is the expected value (with respect to X) of variance of the predictor, $g(X;\mathcal{D})$, with respect to the dataset, $\mathcal{D}$. It represents the extent to which the prediction varies around its expected value. The second term is the expected value of the square of the bias where the bias is defined to be the difference between the expected value of prediction with respect to dataset and the optimal prediction. The second term itself is sometimes called *bias*. The third term is called *noise* since it is caused by the intrinsic noise residing in Y which cannot be reduced even with the optimal predictor (in least-mean-square sense).

The following equation summarizes these three quantities.

$$
\begin{aligned}
&\mathbf{E}_{X,Y,\mathcal{D}}\|g(X;\mathcal{D})-Y\|_2^2 \qquad (11.3)\\
&= \underbrace{\mathbf{E}_{X,\mathcal{D}}\|g(X;\mathcal{D})-\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})\|_2^2}_{\text{variance}}+\underbrace{\mathbf{E}_{X}\|\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})-g^*(X)\|_2^2}_{\text{bias}}+\underbrace{\mathbf{E}_{X,Y}\|g^*(X)-Y\|_2^2}_{\text{noise}}.
\end{aligned}
$$

In general, we do not know the optimal predictor; if we knew, we would not need to train our in the first place. Thus we can only estimate $g(X;\mathcal{D}) - \mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})$ and $\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D}) - Y$. The mean square error can also be expressed in these two quantities as follows.

$$\begin{aligned}
&\mathbf{E}_{X,Y,\mathcal{D}}\|g(X;\mathcal{D}) - Y\|_2^2 = \mathbf{E}_{X,Y,\mathcal{D}}\|g(X;\mathcal{D}) - \mathbf{E}_{\mathcal{D}}g(X;\mathcal{D}) + \mathbf{E}_{\mathcal{D}}g(X;\mathcal{D}) - Y\|_2^2 \\
=\ & \mathbf{E}_{X,Y,\mathcal{D}}\|g(X;\mathcal{D}) - \mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})\|_2^2 + \mathbf{E}_{X,Y,\mathcal{D}}\|\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D}) - Y\|_2^2 \\
& +2\mathbf{E}_{X,Y,\mathcal{D}}(g(X;\mathcal{D}) - \mathbf{E}_{\mathcal{D}}g(X;\mathcal{D}))^T(\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D}) - Y) \\
=\ & \mathbf{E}_{X,\mathcal{D}}\|g(X;\mathcal{D}) - \mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})\|_2^2 + \mathbf{E}_{X,Y}\|\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D}) - Y\|_2^2 \\
& +2\mathbf{E}_{X,Y}\mathbf{E}_{\mathcal{D}}\left((g(X;\mathcal{D}) - \mathbf{E}_{\mathcal{D}}g(X;\mathcal{D}))^T(\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D}) - Y)|X,Y\right) \\
=\ & \mathbf{E}_{X,\mathcal{D}}\|g(X;\mathcal{D}) - \mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})\|_2^2 + \mathbf{E}_{X,Y}\|\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D}) - Y\|_2^2 \\
& +2\mathbf{E}_{X,Y}\mathbf{E}_{\mathcal{D}}(g(X;\mathcal{D}) - \mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})|X,Y)^T(\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D}) - Y) \\
=\ & \mathbf{E}_{X,\mathcal{D}}\|g(X;\mathcal{D}) - \mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})\|_2^2 + \mathbf{E}_{X,Y}\|\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D}) - Y\|_2^2 \\
& +2\mathbf{E}_{X,Y}(\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D}) - \mathbf{E}_{\mathcal{D}}g(X;\mathcal{D}))^T(\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D}) - Y) \\
=\ & \mathbf{E}_{X,\mathcal{D}}\|g(X;\mathcal{D}) - \mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})\|_2^2 + \mathbf{E}_{X,Y}\|\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D}) - Y\|_2^2.
\end{aligned}$$

Equating the last equation with (11.3) yields

$$\mathbf{E}_{X,Y,\mathcal{D}}\|g(X;\mathcal{D}) - Y\|_2^2 = \underbrace{\mathbf{E}_{X,\mathcal{D}}\|g(X;\mathcal{D}) - \mathbf{E}_{\mathcal{D}}g(X;\mathcal{D})\|_2^2}_{\text{variance}} + \underbrace{\mathbf{E}_{X,Y}\|\mathbf{E}_{\mathcal{D}}g(X;\mathcal{D}) - Y\|_2^2}_{\text{bias + noise}} \tag{11.4}$$

Therefore in reality, we can only obtain the sum of the bias and noise (not separately) unless we know the quantity of the noise.

11.3 Maximum Likelihood Estimation

Suppose that $X \in \mathbf{R}^n$ and $Y \in \mathbf{R}^m$ are random variables representing inputs (or independent variables or predictors or features) and outputs (or dependent variables or responses). We want to find a parameterized model to predict Y from X.

We consider the parameter $\theta \in \Theta$ where $\Theta \subset \mathbf{R}^l$ is the set of all possible parameter values, and the model, $g : \mathbf{R}^n \times \mathbf{R}^l \to \mathbf{R}^m$, such that $g(X;\theta)$ is close to Y in some statistical sense.

We further assume that the conditional probability of Y given $g(X;\theta)$ can be characterized by $\beta \in \mathcal{B}$, *i.e.*, $p(Y|g(X;\theta))$ is a function of β where $\mathcal{B}$ is the set of all possible values for β. To express that $p(Y|g(X;\theta))$ is a function of β, we will use a notation, $p(Y|X;\theta,\beta)$, for $p(Y|g(X;\theta))$.

Now suppose that we have observed N independent data sample, $\{(x_i,y_i)\}_{i=1}^N$ where $x_i \in \mathbf{R}^n$ and $y_i \in \mathbf{R}^m$. We want to find $\theta \in \Theta$ which maximizes the probability of this event, *i.e.*,

$$p\left((X_1,Y_1) = (x_1,y_1),\dots(X_N,Y_N) = (x_N,y_N)\right) \tag{11.5}$$

For notational convenience, we define two random variables, $\tilde{X} \in \mathbf{R}^{n\times N}$ and $\tilde{Y} \in \mathbf{R}^{m\times N}$, such that

$$\tilde{X} = \left[\begin{array}{ccc} X_1 & \dots & X_N \end{array}\right] \tag{11.6}$$

and

$$\tilde{Y} = \left[\begin{array}{ccc} Y_1 & \dots & Y_N \end{array}\right]. \tag{11.7}$$

We also defined $\tilde{x} \in \mathbf{R}^{n\times N}$ and $\tilde{y} \in \mathbf{R}^{m\times N}$, such that

$$\tilde{x} = \left[\begin{array}{ccc} x_1 & \dots & x_N \end{array}\right] \tag{11.8}$$

and

$$\tilde{y} = \left[\begin{array}{ccc} y_1 & \dots & y_N \end{array}\right]. \tag{11.9}$$

Then (11.5) becomes

$$p(\tilde{X} = \tilde{x}, \tilde{Y} = \tilde{y}). \tag{11.10}$$

Bayes's theorem and the independence among (X_i, Y_i) imply

$$\begin{aligned} p(\tilde{X} = \tilde{x}, \tilde{Y} = \tilde{y}) &= \prod_{i=1}^{N} p(X_i = x_i, Y_i = y_i) \\ &= \prod_{i=1}^{N} p(X_i = x_i | Y_i = y_i; \theta, \beta) p(X_i = x_i) \\ &\propto \prod_{i=1}^{N} p(X_i = x_i | Y_i = y_i; \theta, \beta). \end{aligned}$$

Here $p(X_i = x_i | Y_i = y_i; \theta, \beta)$ is called *likelihood function*.

The maximum likelihood estimation (MLE) (or learning) is to find θ (and β) which maximizes this probability. Since the log function is a strictly increasing function, taking log on this probability does give us the same results when maximizing it.

$$\log p(\tilde{X} = \tilde{x}, \tilde{Y} = \tilde{y}) = \sum_{i=1}^{N} \log p(X_i = x_i | Y_i = y_i; \theta, \beta) + \text{constant} \tag{11.11}$$

In many cases, this log form is preferable due to its numerical property.

The optimal value of θ for the maximum likelihood estimation (when β is fixed) can be written as

$$\theta_{\text{ML}}(\beta) = \underset{\theta \in \Theta}{\text{argmax}} \sum_{i=1}^{N} \log p(X_i = x_i | Y_i = y_i; \theta, \beta). \tag{11.12}$$

Note that the solution is a function of β when it is fixed. We sometimes want to find the optimal value for β, too. In this case, we have

$$(\theta_{\text{ML}}, \beta_{\text{ML}}) = \underset{(\theta,\beta) \in \Theta \times \mathcal{B}}{\text{argmax}} \sum_{i=1}^{N} \log p(X_i = x_i | Y_i = y_i; \theta, \beta). \tag{11.13}$$

Note that both of these are point estimation, *i.e.*, we want to find one value for θ (and β) to maximize the log likelihood function.

Now the MLE model is given by $g(\cdot; \theta_{\text{ML}}) : \mathbf{R}^n \to \mathbf{R}^m$, *i.e.*, given $x \in \mathbf{R}^n$, we can predict $y \in \mathbf{R}^m$ by $g(x; \theta_{\text{ML}})$.

11.4 Maximum a Posteriori Estimation

In MLE, we regard θ as a deterministic variable, which is called Frequentist perspective.

However, we sometimes have prior knowledge of the distribution of θ (or belief about θ). In this situation, we want to find probability distribution of θ after observing some evidence, *e.g.*, the N data sample we have observed, $\{(x_i, y_i)\}_{i=1}^N$.

The natural way of finding the distribution of θ after observing this evidence is to evaluate the condition probability of θ given $\tilde{x}$ and $\tilde{y}$, *i.e.*,

$$p(\theta|\tilde{X} = \tilde{x}, \tilde{Y} = \tilde{y}). \tag{11.14}$$

Note that θ is considered to be a *random* variable unlike in MLE case.

Here we introduce a new parameter $\alpha \in \mathcal{A}$ that characterizes the distribution of θ where $\mathcal{A}$ is the set of all the possible values of α.

Since the data samples are assumed to be independent, the Bayes' theorem implies that

$$\begin{aligned}
p(\theta|\tilde{X} = \tilde{x}, \tilde{Y} = \tilde{y}; \alpha, \beta) &= p(\tilde{X} = \tilde{x}, \tilde{Y} = \tilde{y}|\theta; \alpha, \beta) p(\theta; \alpha, \beta) / p(\tilde{X} = \tilde{x}, \tilde{Y} = \tilde{y}) \\
&= p(\tilde{X} = \tilde{x}, \tilde{Y} = \tilde{y}|\theta; \beta) p(\theta; \alpha) / p(\tilde{X} = \tilde{x}, \tilde{Y} = \tilde{y}) \\
&\propto p(\tilde{X} = \tilde{x}, \tilde{Y} = \tilde{y}|\theta; \beta) p(\theta; \alpha) \\
&= p(\theta; \alpha) \prod_{i=1}^N p(X_i = x_i, Y_i = y_i|\theta; \beta) \\
&= p(\theta; \alpha) \prod_{i=1}^N p(Y_i = y_i|X_i = x_i, \theta; \beta) p(X_i = x_i|\theta; \beta) \\
&= p(\theta; \alpha) \prod_{i=1}^N p(Y_i = y_i|X_i = x_i, \theta; \beta) p(X_i = x_i) \\
&\propto p(\theta; \alpha) \prod_{i=1}^N p(Y_i = y_i|X_i = x_i, \theta; \beta)
\end{aligned}$$

The maximum a posteriori (MAP) estimation is to find θ (when β is fixed) which maximizes this posteriori probability. Thus, the MAP solution can be expressed as

$$\theta_{\text{MAP}}(\alpha, \beta) = \underset{\theta \in \Theta}{\text{argmax}} \left(\log p(\theta; \alpha) + \sum_{i=1}^N \log p(Y_i = y_i|X_i = x_i, \theta; \beta) \right) \tag{11.15}$$

where $p(Y_i = y_i|X_i = x_i, \theta; \beta)$ is called *likelihood function*.

Note the difference between the likelihood function, $p(Y_i = y_i|X_i = x_i; \theta, \beta)$, in (11.12) and the likelihood function, $p(Y_i = y_i|X_i = x_i, \theta; \beta)$, in (11.15) where θ in (11.12) is an optimization variable and θ in (11.15) is a variable for a random variable.

Now the MAP model is given by $g(\cdot; \theta_{\text{MAP}}) : \mathbf{R}^n \to \mathbf{R}^m$, *i.e.*, given $x \in \mathbf{R}^n$, we can predict $y \in \mathbf{R}^m$ by $g(x; \theta_{\text{MAP}})$.

11.5 Bayesian prior update

Note that both MLE and MAP estimation is a point estimation, *i.e.*, to find one solution that maximizes some probability.

However, the posterior probability $p(\theta|\tilde{X}=\tilde{x},\tilde{Y}=\tilde{y};\alpha,\beta)$ can be used to update the prior probability.

In Bayesian probability theory, if the posterior distributions are in the same probability distribution family as the prior probability distribution, the prior and posterior are then called *conjugate distributions*, and the prior is called a *conjugate prior* for the likelihood function.

In this case, we can update the prior by updating α. Suppose that we have initial prior, $\alpha^{(0)} \in \mathcal{A}$. After we observing first data samples, $(\tilde{x}^{(1)},\tilde{y}^{(1)})$, we evaluate the posterior probability $p(\theta|\tilde{X}=\tilde{x}^{(1)},\tilde{Y}=\tilde{y}^{(1)};\alpha^{(0)},\beta)$ which can be characterized by some θ^{+} due to conjugate distribution assumption. We let $\alpha^{(1)}$ be this updated parameter. We can repeat this process every time we observe new set of data samples. This process can be expressed as

$$\alpha^{(0)} \xrightarrow{\tilde{x}^{(1)},\tilde{y}^{(1)}} \alpha^{(1)} \xrightarrow{\tilde{x}^{(2)},\tilde{y}^{(2)}} \alpha^{(2)} \xrightarrow{\tilde{x}^{(2)},\tilde{y}^{(2)}} \alpha^{(3)} \cdots \tag{11.16}$$

We can see this process as the one similar to what happens in our brain. A simplified version of explaining human learning process is to update its prior knowledge whenever it observes new evidence. For example, if one has observed that when it rains, the temperature is high, her prior knowledge is that

$$\text{rain} \rightarrow \text{high temperature.} \tag{11.17}$$

However, if she experiences a rainy day with low temperature, her knowledge is updated as something like

$$\text{rain} \rightarrow \begin{cases} \text{high temperature} & \text{with probability } 0.9 \\ \text{low temperature} & \text{with probability } 0.1 \end{cases} \tag{11.18}$$

Now this becomes her new prior. If she observes more rainy cold days, her knowledge is updated as something like

$$\text{rain} \rightarrow \begin{cases} \text{high temperature} & \text{with probability } 0.7 \\ \text{low temperature} & \text{with probability } 0.3 \end{cases} \tag{11.19}$$

This analogy tells why the prior in Bayesian statistics is sometimes called Bayesian belief. This prior belief is something that can be constantly updated with new evidence.

11.6 Predictive Distribution

If θ is fixed, the probability of $y \in \mathbf{R}^m$ given $x \in \mathbf{R}^n$, $p(Y=y|X=x;\theta,\beta)$, is solely characterized by $\beta \in \mathcal{B}$. However, if we regard θ as a random variable with distribution characterized by α, the probability of $y \in \mathbf{R}^m$ given $x \in \mathbf{R}^n$ can be evaluated by

$$\begin{aligned} p(Y=y|X=x;\alpha,\beta) &= \int_{\theta\in\Theta} p(Y=y,\theta|X=x;\alpha,\beta)d\theta \\ &= \int_{\theta\in\Theta} p(Y=y|X=x,\theta;\alpha,\beta)p(\theta|X=x;\alpha,\beta)d\theta \\ &= \int_{\theta\in\Theta} p(Y=y|X=x,\theta;\beta)p(\theta;\alpha)d\theta, \end{aligned}$$

i.e.,

$$p(Y=y|X=x;\alpha,\beta) = \int_{\theta\in\Theta} p(Y=y|X=x,\theta;\beta)p(\theta;\alpha)d\theta. \tag{11.20}$$

This is called predictive distribution. This Bayesian statistical predictor, if (11.20) can be efficiently evaluated, not only gives the point estimation, *e.g.*, by mean or mode, but also the distribution of the output. One advantage of this approach is that we can evaluate the confidence interval

Chapter 12

Optimization for Machine Learning

12.1 Gradient method

Suppose that $f : \mathbf{R}^n \to \mathbf{R}$. An unconstrained optimization problem is

$$\text{minimize} \quad f(x) \tag{12.1}$$

The gradient method is

$$x^{k+1} = x^k - \alpha^k \nabla f(x^k) \tag{12.2}$$

12.2 Stochastic gradient method

Suppose that $f_i : \mathbf{R}^n \to \mathbf{R}$ for $i = 1, \ldots, n$. An unconstrained optimization problem is

$$\text{minimize} \quad \tfrac{1}{n} \textstyle\sum_{i=1}^{n} f_i(x) \tag{12.3}$$

The gradient method is

$$x^{k+1} = x^k - \alpha^k \sum^{k} \nabla f(x^k) \tag{12.4}$$

Chapter 13

Bayesian Network

$$\Pr\{X_1,\ldots,X_n\} = \prod_{i=1}^{n} \Pr\{X_i \mid X_1,\ldots,X_{i-1}\} = \prod_{i=1}^{n} \Pr\{X_i \mid \mathbf{parent}(X_i)\} \tag{13.1}$$

Chapter 14

Collaborative Filtering

14.1 Item-based Collaborative Filtering

The purpose of the item-based collaborative filtering is to recommend items using the information obatined from similar items. Since the information from other items help provide better recommendation to customers, it is called an *item-based collaborative filtering*.

We formulate an item-based collaborative filtering as follows. We assume that there are n_C customers and n_I items. We further assume that we have partial information as to the taste of each customer for each item. In typical cases, we have customers can leave reviews with ratings for n items where $n \ll n_I$. When we have hundreds of thousands of items, this makes the data structure extremely sparse. This fact plays a critical role when we calculate similarities among items or customers, or learn the matrix factorization-based collaborative filtering models, e.g., using gradient descent (GD) method or alternating least squares (ALS) method.[1]

Now suppose that we have the rating matrix

$$R \in (\mathbf{R} \cup \{\mathrm{NaN}\})^{n_C \times n_I} \tag{14.1}$$

where $R_{ij} \in (\mathbf{R} \cup \{\mathrm{NaN}\})$ denotes *certain score* corresponding to ith customer and jth item and NaN denotes we do not know the values. Note that this certain score can refer to rating of movies, but can mean virtually anything related to customers' taste or activities. For example, it can mean the frequency of customers' clicks on certain menus where the items mean the menu items for this example.

This matrix is sparse, but not in a traditional way. Generally, we say a matrix is sparse if the number of nonzero entries is much less than $n_C \times n_I$. We say R is sparse because there are huge number of entries for which we *do not know* the true values for them. In a general recommendation problem, the purpose is to guess or estimate the true values, *e.g.*, what rating the customer would give if she did, or how frequently a customer would click on the menu item if she knew there exist such menu item.

Now we want to evaluate the similarity (or distance) among items. In many places in the recommendation systems and information retrieval literature, it is shown that applying some transformation or weighting on the values before calculating the similarity measure, *e.g.*, cosine similarity or correlation coefficient[2]. There weighting methods and similarity measures will be discussed in later sections.

14.1.1 Rating matrix modeling for menu personalization for mobile shopping app

Here we consider a problem of using a collaborative filtering for efficient menu personalization problem for an mobile shoppiong app. Suppose that we have some history data of the number of clicks or tabs on each menu item by each customer for certain period. Therefore we can define the matrix in (14.1) where n_C denotes the number of customers, n_I denotes the number of menu items, and R_{ij} denotes the number of clicks or tabs on jth menu item by ith customer with $1 \leq i \leq n_C$ and $1 \leq j \leq n_I$. recommendation system design cases where each component represents the rating

[1] These models will be discussed in later sections.

[2] The correlation coefficient is sometimes called the Pearson correlation coefficient after Karl Pearson, who was an English mathematician and biostatistician.

of an item, these numbers vary depending on the time interval for which we collect the data. So here we model this matrix as a function of time. Moreover, we can have more than one source for the customers' data. Thus we assume multi-source model.

14.1.1.1 Rating matrix modeling

Suppose that $R_s : \mathbf{R} \to \mathbf{R}^{n_C \times n_I}$ for $s \in \mathcal{S}$ is the rating matrix dependent on time, *i.e.*,

$$R_s(t) \in \mathbf{R}^{n_C \times n_I} \tag{14.2}$$

where $\mathcal{S}$ refers to the set of data sources and $R_s(t)_{ij}$ refers to the number of clicks or tabs for jth item by ith customer which was collected from data source, $s \in \mathcal{S}$. For example, $\mathcal{S} = \{\mathrm{mobile}, \mathrm{web}\}$.

For our purpose, we want to have one matrix which represents the customers' activity or behavior with their taste by properly aggregating the history data. Among many possible choices, we choose exponentially decaying weight method together with proper weight on data sources. We define the following aggregate rating matrix

$$\bar{R}(t) = \sum_{s \in \mathcal{S}} \sum_{\tau=0}^{\infty} \gamma^{\tau} w_s R_s(t-\tau) \tag{14.3}$$

where γ is a positive constant less than 1 and w_s are the weights on each data source such that $\sum_{s \in \mathcal{S}} w_s = 1$.

This aggregate matrix, however, has one problem. Suppose that the 1st customer and the 2nd customer have the very same activity pattern on menu items. However, the 1st customer happens to have actively used the shopping app or website for this week, but the 2nd customer has not used the app or website for the past few days or weeks. Because of the decay factor, the 2nd customer's activity would appear much less than those of the 1st customer even ideally we need to have the very same pattern for those two customers. Therefore we need to normalize the values so that these two customers' preferences are considered the same. For this we normalize each row, so that the sum of each row is always one.

$$\tilde{R}(t) = \mathbf{diag}(\bar{R}(t)\mathbf{1}_{n_I})^{-1}\bar{R}(t) \tag{14.4}$$

where $\mathbf{1}_{n_I} \in \mathbf{R}^{n_I}$ is the vector where every entry is 1 and $\mathbf{diag}(x)$ for $x \in \mathbf{R}^n$ refers to a diagonal matrix whose diagonal entries are the entries of x in the same order. We can readily see that

$$\tilde{R}(t)\mathbf{1}_{n_I} = \mathbf{diag}(\bar{R}(t)\mathbf{1}_{n_I})^{-1}\bar{R}(t)\mathbf{1}_{n_I} = \mathbf{1}_{n_C}, \tag{14.5}$$

i.e., the sum of each row is 1. From this point on, we will remove the subscript for $\mathbf{1}$ unless it can cause confusion as to the dimension of the vector.

One problem of this approach is that $\tilde{R}(t)$ cannot be defined if some row has all zero entries. Even if the sum of every row is nonzero, if the values is very small, *e.g.*, a customer has not shown any activity except for a few clicks or tabs on a handful of menu items, that doesn't mean that the corresponding row represents the customer's preference. Hence, we can augment the values of $\tilde{R}(t)$ by prior distribution of menu items as in the following section.

14.1.2 Value augmentation based on Bayesian MAP

To prevent the divide-by-zero errors from occurring while evaluating (14.4), we consider maximum a posteriori (MAP) estimation for all the rows of $\bar{R}(t)$ in (14.3).

We can think of a problem of filling out each entry in $\bar{R}(t)$ as performing N multinomial experiments on each item $j \in \{1, \ldots, n_I\}$ and count the occurrences for each item and fill in $\bar{R}(t)$ for each customer $i \in \{1, \ldots, n_C\}$ after normalization.

Now assume that we the prior as Dirichlet-multinomial model, *i.e.*,

$$\mathrm{Dir}(\theta|\alpha) = \frac{1}{B(\alpha)} \prod_{k=1}^{n_I} \theta_k^{\alpha_k - 1} I(x = k). \tag{14.6}$$

Then since the likelihood for the multinomial distribution has the form

$$p(\mathcal{D}|\theta) = \prod_{k=1}^{n_I} \theta_k^{N_k}, \tag{14.7}$$

the posterior is

$$p(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta)p(\theta) \propto \prod_{k=1}^{n_I} \theta_k^{\alpha_k + N_k - 1} = \mathrm{Dir}(\theta|\alpha_1 + N_1, \ldots, \alpha_{n_I} + N_{n_I}). \tag{14.8}$$

It can be proved that the maximum a posteriori (MAP) estimate for $\theta_1, \ldots, \theta_{n_I}$ is

$$\hat{\theta}_k = \frac{N_k + \alpha_k - 1}{N + \alpha_0 - n_I} \tag{14.9}$$

where $\alpha_0 = \sum_{k=1}^{n_I} \alpha_k$ (K.P. Murphy). By choosing proper values for $\alpha_k > 1$, we can make every entry nonzero in $\bar{R}(t)$.

Adding this information to the rating matrix is equivalent to Bayesian inference since this uses a priori distribution. Therefore, it will play a regularization role in our inference.

14.1.3 Similarity measure among items

Suppose that we have the transformed matrix, $\tilde{R} \in \mathbf{R}^{n_C \times n_I}$. For the case of mobile shopping app menu personalization, $\tilde{R} = \tilde{R}(t)$ for some t. For general cases, $\tilde{R}$ equals to R with all NaNs replaced by 0.

Suppose that $c_1, \ldots, c_{n_C} \in \mathbf{R}^{n_I}$ are the row vectors of $\tilde{R}$ and $d_1, \ldots, d_{n_I} \in \mathbf{R}^{n_C}$ are the column vectors of $\tilde{R}$, *i.e.*,

$$\tilde{R} = \begin{bmatrix} c_1^T \\ \vdots \\ c_{n_C}^T \end{bmatrix} = \begin{bmatrix} d_1 & \cdots & d_{n_I} \end{bmatrix} \in \mathbf{R}^{n_C \times n_I}. \tag{14.10}$$

14.1.3.1 Cosine similarity

The cosine similarity measures the cosine of the angle between the two vectors, *i.e.*, it is defined by

$$s(d_i, d_j) = \frac{d_i^T d_j}{\|d_i\|_2 \|d_j\|_2} \tag{14.11}$$

where $x^T y$ denotes the inner product of two vectors x and y, and $\|\cdot\|_2$ denotes the 2-norm of a vector. The Jensen's inequality guarantees that $-1 \leq s(d_i, d_j) \leq 1$. It can be easily shown that we have $0 \leq s(d_i, d_j) \leq 1$ when every entry in $\tilde{R}$ is nonzero.

14.1.3.2 Cosine similarity when prior distribution is used

When a prior distribution is added to $\tilde{R}$ as in §14.1.2, the sparsity breaks and the matrix becomes a dense matrix. Therefore it seems that we lose huge advantage in computation efforts when calculating the similarities. However, because the added matrix is rank-one matrix, we can still exploit the sparsity and calculate the similarities at almost the same cost as before.

Assume that $\hat{\theta}_k$ in (14.9) has been calculated and is added to $\tilde{R}$. Thus we have a new rating matrix.

$$\tilde{R}_{\hat{\theta}} = \tilde{R} + \lambda \mathbf{1}_{n_C \times n_I} \,\mathbf{diag}(\hat{\theta}) = \tilde{R} + \lambda \begin{bmatrix} \hat{\theta}_1 \mathbf{1} & \cdots & \hat{\theta}_{n_I} \mathbf{1} \end{bmatrix} \in \mathbf{R}^{n_C \times n_I} \tag{14.12}$$

where $\hat{\theta} = \begin{bmatrix} \hat{\theta}_1 & \cdots & \hat{\theta}_{n_I} \end{bmatrix}^T \in \mathbf{R}^{n_I}$. Here λ plays a similar role as the coefficient for the regularization when we assume that

$$p(\hat{\theta}) \sim N(0, \lambda I_{n_I}). \tag{14.13}$$

Now let $\tilde{d}_1, \ldots, \tilde{d}_{n_I}$ are the column vectors of $\tilde{R}_{\hat{\theta}}$. Then the cosine similarity of ith item and jth item is

$$s(\tilde{d}_i, \tilde{d}_j) = \frac{\tilde{d}_i^T \tilde{d}_j}{\|\tilde{d}_i\|_2 \|\tilde{d}_j\|_2}. \tag{14.14}$$

Now note that

$$\tilde{d}_i^T \tilde{d}_j = (d_i + \lambda \hat{\theta}_i \mathbf{1})^T (d_j + \lambda \hat{\theta}_j \mathbf{1}) = d_i^T d_j + \lambda \hat{\theta}_j \mathbf{1}^T d_i^T + \lambda \hat{\theta}_i \mathbf{1}^T d_j + \lambda^2 \hat{\theta}_i \hat{\theta}_j n_C \tag{14.15}$$

hence

$$\|\tilde{d}_i\|_2^2 = \tilde{d}_i^T \tilde{d}_i = \|d_i\|_2^2 + 2\lambda \hat{\theta}_i \mathbf{1}^T d_i^T + \lambda^2 \hat{\theta}_i^2 n_C \tag{14.16}$$

We can pre-compute $\mathbf{1}^T d_i$ for $i = 1, \ldots, n_I$ which takes less than n_R where n_R refers to the number of nonzero entries in $\tilde{R}$. Hence, the additional computational cost is negligible.

We can also solve this problem at a different angle. In order to calculate the inner projects and the norms in (14.15) and (14.16), we can do the following matrix multiplication.

$$\begin{aligned}
\tilde{R}_{\hat{\theta}}^T \tilde{R}_{\hat{\theta}} &= (\tilde{R} + \lambda \mathbf{1}_{n_C \times n_I} \,\mathbf{diag}(\hat{\theta}))^T (\tilde{R} + \lambda \mathbf{1}_{n_C \times n_I} \,\mathbf{diag}(\hat{\theta})) && (14.17)\\
&= \tilde{R}^T \tilde{R} + \lambda \tilde{R}^T \mathbf{1}_{n_C \times n_I} \,\mathbf{diag}(\hat{\theta}) + \lambda \,\mathbf{diag}(\hat{\theta}) \mathbf{1}_{n_I \times n_C} \tilde{R} && (14.18)\\
&\quad + \lambda^2 \,\mathbf{diag}(\hat{\theta}) \mathbf{1}_{n_I \times n_C} \mathbf{1}_{n_C \times n_I} \,\mathbf{diag}(\hat{\theta}) && (14.19)\\
&= \tilde{R}^T \tilde{R} + \lambda \hat{\theta} \tilde{r}^T + \lambda \tilde{r} \hat{\theta}^T + \lambda^2 n_C \hat{\theta} \hat{\theta}^T && (14.20)
\end{aligned}$$

14.1.3.3 Correlation coefficient similarity

The correlation coefficient is defined by

$$s(d_i, d_j) = \frac{(d_i - \mathbf{1}^T d_i/n_C)^T (d_j - \mathbf{1}^T d_j/n_C)}{\|d_i - \mathbf{1}^T d_i/n_C\|_2 \|d_j - \mathbf{1}^T d_j/n_C\|_2}. \tag{14.21}$$

Again the Jensen's inequality guarantees that $-1 \leq s(d_i, d_j) \leq 1$, but the fact that every entry in $\tilde{R}$ is nonzero does not make this similarity nonnegative.

14.1.4 Data value transformation

When there are popular items across many customers, those values can dominate in the similarity measure evaluation. For example, the aggregate rating matrix, $\bar{R}$, looks like the following:

$$\begin{array}{c} \left[\begin{array}{ccccc} \cdots & 100 & \cdots & 130 & \cdots \\ \cdots & 2 & \cdots & 5 & \cdots \\ \cdots & 3 & \cdots & 3 & \cdots \\ \cdots & 10 & \cdots & 1 & \cdots \end{array}\right] \\ \begin{array}{ccccc} & \uparrow & & \uparrow & \\ & d_i & & d_j & \end{array} \end{array} \tag{14.22}$$

Note here that the scores in the first row are much larger than the other terms. This can be caused by the fact that some customers are way more active than other customers, *e.g.*, they can listen to some musics many times or uses an shopping app very frequently. Now the cosine similarity and the correlation coefficient similarity between d_i and d_j are

$$0.9956 \text{ and } 0.9951 \tag{14.23}$$

respectively. However, if we remove the first customer and recalculate both similarities, it yields

$$0.4611 \text{ and } -0.9176 \tag{14.24}$$

respectively.

Note that both similarity measures give very different measures. The correlation coefficient similarity, which also tells whether both have positive or negative correlations by its sign, gives opposite signs. This shows how *some* customers activity or behavior can change the item-to-item similarities drastically.

However, this doesn't seem to be right. The item-to-time similarities are supposed to represent the nature of the relation among items, hence should not be decided by a small number of extremely active customers.

For this reason, in many recommendation systems literature, it has been reported that the similarity measures mentioned above do not show good performance. To address this issue, various value transformation methods have been introduced. We will discuss some of these methods in the following sections.

14.1.4.1 TFIDF (or tf-idf)

The term frequency–inverse document frequency (TFIDF or tf-idf) is a numerical statistic which has been widely used in the field of information retrieval. It was orginally designed to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. Tf–idf is one of the most popular term-weighting schemes today; 83% of text-based recommender systems in digital libraries use tf–idf.

14.1.4.2 Okapi BM25 transformation

Instead, the Okapi BM25 score has shown much better performance when applied to the entries in $\tilde{R}$ before calculating similarity measures. It is defined by

$$\mathrm{bm}_{25}(i,j) = \log\left(\frac{n_C - \|d_j\|_0 + 0.5}{\|d_j\|_0 + 0.5}\right) \frac{\tilde{R}_{ij}(k_1+1)}{\tilde{R}_{ij} + k_1\left(1 - b + b\frac{\|c_i\|_0}{\bar{c}}\right)} \tag{14.25}$$

where $\|\cdot\|_0$ denotes the number of nonzero entries of a vector and k_1 and b are some parameters usually with $k_1 \in [1.2, 2.0]$ and $b \sim 0.75$.

14.1.5 Recommendation based on item similarities

Now finally, we can evaluate new (menu) item scores for each (menu) items for each customer. Here we suggest two methods.

- For customer i and item j, we calculate new recommendation as

$$\hat{R}_{ij} = \frac{\sum_{i=1}^{n_I} s(i,j)\tilde{R}_{i,j}}{\sum_{i=1}^{n_I} s(i,j)} \tag{14.26}$$

  where $s(i,j)$ is the cosine similarity between d_i and d_j.

- The second candidate considers the deviation from the average, *i.e.*,

$$\hat{R}_{ij} = \mathbf{1}^T c_i/n_I + \frac{\sum_{i=1}^{n_I} s(i,j)(\tilde{R}_{ij} - \mathbf{1}^T c_i/n_I)}{\sum_{i=1}^{n_I} s(i,j)} \tag{14.27}$$

  where $s(i,j)$ is the correlation coefficient similarity between d_i and d_j.

These new values are the ones obtained from the history data for that particular customer together with those for neighboring customers. Hence, this will give better recommendation.

14.2 Collaborative Filtering using Matrix Factorization

Here we discuss the collaborative filtering using matrix factorization, which uses latent factor models. There are two types of latent factors; one for customers and one for items.

The customer latent factors represent customers' taste or tendency. In psychological perspective, even customers themselves may not realize these, but they implicitly express these by their activities. They can be

- the degree of pursuing economical life
- the degree of interest in tableware
- the degree of caring children
- the degree of interest in books
- the degree of putting values on family

On the other hand, the item latent factors represent items' attributes. Again, customers may not realize these, but they tend to click on some menus or items according items attributes and their inclination. They can be

- the probability of leading to inexpensive items
- the probability of leading to tableware
- the probability of leading to items related to children
- the probability of leading to book purchase
- the probability of leading to items related to family

We will discuss how we can utilize these latent factors to design our collaborative filtering below. Note, however, that it is generally impossible to identify the actual latent factors a ML algorithm yields with any of the aforementioned factors. These latent factors are *not predefined or decided apriori*, but are learned or revealed through the values coming out of the algorithms. What's more important is that these methods work very well in practice.

We will also describe different types of problem formulations and different approaches to solve them, *e.g.*, stochastic gradient descent (SGD) method, alternating SGD, and alternating least squares (ALS).

14.2.1 Problem definition and formulations

As before, suppose that there are n_C customers and n_I items. We assume that we have a (sparse) rating matrix, $R \in (\mathbf{R} \cup \{\mathrm{NaN}\})^{n_C \times n_I}$, in (14.1). Sometimes we need to deal with the aggregate rating matrix, $\bar{R}(t) \in (\mathbf{R} \cup \{\mathrm{NaN}\})^{n_C \times n_I}$, in (14.3). To simplify notations, we will refer to both types of rating matrix as R.

Now we assume that there are n_L latent factors. This means that every customer n_L latent factors and every item has n_L latent factors.

Let $x_i \in \mathbf{R}^{n_L}$ be a column vector representing n_L latent factors for the ith customer and let $y_j \in \mathbf{R}^{n_L}$ be a column vector representing n_L latent factors for the jth item with

$$x_i = \left[\begin{array}{c} x_{i,1} \\ \vdots \\ x_{i,n_L} \end{array}\right] \in \mathbf{R}^{n_L} \tag{14.28}$$

and

$$y_j = \left[\begin{array}{c} y_{j,1} \\ \vdots \\ y_{j,n_L} \end{array}\right] \in \mathbf{R}^{n_L}. \tag{14.29}$$

Then $x_{i,1}, \ldots, x_{i,n_L}$ are the n_L tendency or taste factors of the ith customer and let $y_{j,1}, \ldots, y_{j,n_L}$ are the n_L attributes of the jth item.

Then we assume that the rating of the j item by the i customer can be inferenced (or estimated) by the sum of the corresponding factors, *i.e.*,

$$R_{i,j} \simeq \hat{R}_{i,j} = x_{i,1}y_{j,1} + \cdots + x_{i,n_L}y_{j,n_L}. \tag{14.30}$$

This is the most critical assumption in the matrix factorization.

The purpose of the collaborative filtering is to find these latent factors so as to make these inference as accurate as possible, *i.e.*, to solve the following optimization problem

$$\text{minimize} \quad \textstyle\sum_{1\leq i\leq n_C, 1\leq j\leq n_I : R_{i,j}\in\mathbf{R}} l(R_{i,j}, \hat{R}_{i,j}) \tag{14.31}$$

where the $n_L(n_C + n_I)$ optimization variables are $x_{i,k}$ and $y_{j,k}$ for $1 \leq i \leq n_C$, $1 \leq j \leq n_I$, and $1 \leq k \leq n_L$, and $l : \mathbf{R} \times \mathbf{R} \to \mathbf{R}_+$ is a loss function measuring the distance between the true value and the estimate, hoping that $\hat{R}_{i,j}$ can accurately predict the rating for $1 \leq i \leq n_C$ and $1 \leq j \leq n_I$ where $R_{i,j}$ is not given.

In most cases, we use squared loss for l, so we will also sue the squared loss (except some special cases).

$$l(y_1, y_2) = (y_1 - y_2)^2. \tag{14.32}$$

Now let us come up with more compact notation to describe our problem. Let $X \in \mathbf{R}^{n_C \times n_L}$ be the customer latent factor matrix whose ith row is x_i^T, *i.e.*,

$$X = \left[\begin{array}{c} x_1^T \\ \vdots \\ x_{n_C}^T \end{array}\right] = \left[\begin{array}{cccc} x_{1,1} & x_{1,2} & \cdots & x_{1,n_L} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n_L} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n_C,1} & x_{n_C,2} & \cdots & x_{n_C,n_L} \end{array}\right] \in \mathbf{R}^{n_C \times n_L}. \tag{14.33}$$

Likewise, let $Y \in \mathbf{R}^{n_I \times n_L}$ be the item latent factor matrix whose jth row is y_j^T, *i.e.*,

$$Y = \left[\begin{array}{c} y_1^T \\ \vdots \\ y_{n_I}^T \end{array}\right] = \left[\begin{array}{cccc} y_{1,1} & y_{1,2} & \cdots & y_{1,n_L} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,n_L} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n_I,1} & y_{n_I,2} & \cdots & y_{n_I,n_L} \end{array}\right] \in \mathbf{R}^{n_I \times n_L}. \tag{14.34}$$

Note that these two matrices are extremely skinny meaning that we have much more rows then columns in general since n_C could be hundreds of millions and n_I could be hundreds of thousands, but n_L is 100 or so at most.

Now using this notation, we can say

$$\hat{R} = XY^T, \tag{14.35}$$

which is mathematically equivalent to

$$\hat{R}_{i,j} = x_i^T y_j = x_{i,1}y_{j,1} + \cdots + x_{i,n_L}y_{j,n_L}, \tag{14.36}$$

which again is equivalent to (14.30).

Then the optimization problem (14.31) can be rewritten as

$$\text{minimize} \quad \textstyle\sum_{1\leq i\leq n_C, 1\leq j\leq n_I : R_{i,j}\in\mathbf{R}} l(R_{i,j}, x_i^T y_j) \tag{14.37}$$

where the optimization variables are $X \in \mathbf{R}^{n_C\times n_L}$ and $Y \in \mathbf{R}^{n_I\times n_L}$.

If we use the squared loss for l, then we can write this equation more compactly as follows.

$$\text{minimize} \quad \|R - XY^T\|_{F,R}^2 \tag{14.38}$$

where $\|Z\|_{F,R}$ refers to Frobenius norm calculated for those entries $Z_{i,j}$ with $R_{i,j} \in \mathbf{R}$.

Note that the number of real variables we need to optimize is the same for all the formulations (14.31), (14.37), and (14.38) (because they are equivalent optimization problems).

14.2.2 Solution methods

14.2.2.1 Matrix factorization via singular value decomposition (SVD)

First we discuss obtaining the latent factors, *i.e.*, X and Y, using sparse singular value decomposition (SVD).

For any rank-k matrix $A \in \mathbf{R}^{m\times n}$, we always have a SVD

$$A = U\Sigma V^T \tag{14.39}$$

where $U \in \mathbf{R}^{m\times k}$, $V \in \mathbf{R}^{n\times k}$, and $\Sigma \in \mathbf{R}^{k\times k}$. (Refer to §14.3.1.1.) The basic idea of obtaining the latent factors using SVD is to apply SVD to the rating matrix.

However, the matrix $R \in (\mathbf{R} \cup \{\text{NaN}\})^{n_C\times n_I}$ can have many unknown values, hence we cannot apply SVD directly. Instead, we employ a certain type of missing data imputation to replace the unknowns with real values, then apply SVD to the matrix. There can be many options for the missing data imputation. Here we list some oft them.

- replacing every unknown with zeros
- replacing unknowns with global item average ratings
- replacing unknowns with global customer average ratings
- use some model based methods (*e.g.*, the item-based collaborative filtering itself can be considered as such a method)

Now suppose that $\tilde{R}$ is R with missing values replaced by proper real values. We compute the largest n_L singular values with corresponding singular vectors. Then we can obtain the approximation of $\tilde{R}$ by

$$\tilde{R} \simeq U_{n_L} \Sigma_{n_L} V_{n_L}^T \tag{14.40}$$

where

$$U_{n_L} = \left[\begin{array}{ccc} u_1 & \cdots & u_{n_L} \end{array} \right] \in \mathbf{R}^{n_C \times n_L} \tag{14.41}$$

$$V_{n_L} = \left[\begin{array}{ccc} v_1 & \cdots & v_{n_L} \end{array} \right] \in \mathbf{R}^{n_I \times n_L} \tag{14.42}$$

and

$$\Sigma_{n_L} = \mathbf{diag}(\sigma_1, \cdots, \sigma_{n_L}) \in \mathbf{R}^{n_L \times n_L}. \tag{14.43}$$

Refer to §14.3.1.1 for more details. This approximation is the best approximation to $\tilde{R}$ in Frobenius norm sense. (Refer to §14.3.1.2 for further details.)

Now since (14.40) can be rewritten as

$$\tilde{R} \simeq U_{n_L} \Sigma_{n_L} V_{n_L}^T = U_{n_L} \Sigma_{n_L}^{1/2} \Sigma_{n_L}^{1/2} V_{n_L}^T = (U_{n_L} \Sigma_{n_L}^{1/2})(V_{n_L} \Sigma_{n_L}^{1/2})^T \tag{14.44}$$

where

$$\Sigma_{n_L}^{1/2} = \mathbf{diag}(\sigma_1^{1/2}, \cdots, \sigma_{n_L}^{1/2}) \in \mathbf{R}^{n_L \times n_L} \tag{14.45}$$

we can obtain the customer and item latent factor matrices as follows.

$$X_{\mathrm{svd}} = U_{n_L} \Sigma_{n_L}^{1/2} = \left[\begin{array}{ccc} \sigma_1^{1/2} u_1 & \cdots & \sigma_{n_L}^{1/2} u_{n_L} \end{array} \right] \in \mathbf{R}^{n_C \times n_L} \tag{14.46}$$

$$Y_{\mathrm{svd}} = V_{n_L} \Sigma_{n_L}^{1/2} = \left[\begin{array}{ccc} \sigma_1^{1/2} v_1 & \cdots & \sigma_{n_L}^{1/2} v_{n_L} \end{array} \right] \in \mathbf{R}^{n_C \times n_L} \tag{14.47}$$

14.2.2.2 Matrix factorization via gradient descent (GD) method

One obvious way to obtain the latent factor matrices is to directly apply gradient descent method to the following optimization problem directly.

$$\text{minimize} \quad f(X,Y) = \textstyle\sum_{i,j:R_{i,j} \in \mathbf{R}} l(R_{i,j}, x_i^T y_j) \tag{14.48}$$

which is equivalent to (14.37). Here we denote the objective function of the optimization problem by $f : \mathbf{R}^{n_C \times n_L} \times \mathbf{R}^{n_I \times n_L} \to \mathbf{R}_+$.

In order to apply gradient descent, we need to evaluate the partial derivative of the objective function with respect to

- $x_{i,k}$ for all $1 \leq k \leq n_L$ and all $i \in \{1 \leq i \leq n_C \mid R_{i,j} \in \mathbf{R}$ for some $1 \leq j \leq n_I\}$
- $y_{j,k}$ for all $1 \leq k \leq n_L$ and all $j \in \{1 \leq j \leq n_I \mid R_{i,j} \in \mathbf{R}$ for some $1 \leq i \leq n_C\}$

For simplicity of the equation derivation, let us assume that

$$\{1 \leq i \leq n_C \mid R_{i,j} \in \mathbf{R} \text{ for some } 1 \leq j \leq n_I\} = \{1, \ldots, n_C\} \tag{14.49}$$

$$\{1 \leq j \leq n_I \mid R_{i,j} \in \mathbf{R} \text{ for some } 1 \leq i \leq n_C\} = \{1, \ldots, n_I\} \tag{14.50}$$

i.e., every item has at least one rating and every customer has at least one rating.

Now the gradient of $f(X,Y)$ with respect to each x_i is

$$\nabla_{x_i} f(X,Y) = \sum_{j:R_{i,j}\in\mathbf{R}} \frac{\partial}{\partial y_2} l(R_{i,j}, x_i^T y_j) y_j \in \mathbf{R}^{n_L} \text{ for } 1 \leq i \leq n_C \tag{14.51}$$

and that with respect to y_j is

$$\nabla_{y_j} f(X,Y) = \sum_{i:R_{i,j}\in\mathbf{R}} \frac{\partial}{\partial y_2} l(R_{i,j}, x_i^T y_j) x_i \in \mathbf{R}^{n_L} \text{ for } 1 \leq j \leq n_I \tag{14.52}$$

If we use the squared loss for l, the optimization problem becomes

$$\text{minimize} \quad f(X,Y) = \|R - XY^T\|_{F,R}^2 \tag{14.53}$$

which is equivalent to (14.38). Then (14.51) and (14.52) imply that the gradients of $f(X,Y)$ can be calculated by

$$\nabla_{x_i} f(X,Y) = -\sum_{j:R_{i,j}\in\mathbf{R}} (R_{i,j} - x_i^T y_j) y_j \in \mathbf{R}^{n_L} \text{ for } 1 \leq i \leq n_C \tag{14.54}$$

and that with respect to y_j is

$$\nabla_{y_j} f(X,Y) = -\sum_{i:R_{i,j}\in\mathbf{R}} (R_{i,j} - x_i^T y_j) x_i \in \mathbf{R}^{n_L} \text{ for } 1 \leq j \leq n_I \tag{14.55}$$

For notational convenience, we stack these vectors to form derivative matrices as below.

If $R \in \mathbf{R}^{n_C \times n_I}$, *i.e.*, there are no missing values, we can write the gradients more compact form, *i.e.*, as the derivatives of X and Y.

$$D_X f(X,Y) = -2(R - XY^T)Y \in \mathbf{R}^{n_C \times n_L} \tag{14.56}$$

and

$$D_Y f(X,Y) = -2(R^T - YX^T)X \in \mathbf{R}^{n_I \times n_L} \tag{14.57}$$

Finally, we describe the gradient descent method.

- choose learning rate strategy $\eta_k > 0$
- choose initial $\tilde{X}$ and $\tilde{Y}$
- let $X_0 := \tilde{X}$ and $Y_0 := \tilde{Y}$
- let $k := 0$
- for each iteration, update X and Y

$$X_{k+1} = X_k - \eta_k D_X f(X_k, Y_k) \tag{14.58}$$
$$Y_{k+1} = Y_k - \eta_k D_Y f(X_k, Y_k) \tag{14.59}$$

- stops if certain stopping criterion is satisfied
- update $k := k+1$ and repeat iterations

14.2.2.3 Matrix factorization via alternating gradient descent (GD) method

Instead of updating X and Y simultaneously, we can update X and Y after the other is updated. The algorithm can be described as follows.

- choose learning rate strategy $\eta_k > 0$
- choose initial $\tilde{X}$ and $\tilde{Y}$
- let $X_0 := \tilde{X}$ and $Y_0 := \tilde{Y}$
- let $k := 0$
- for each iteration, update X and Y

$$\begin{aligned} X_{k+1} &= X_k - \eta_k D_X f(X_k, Y_k) && (14.60) \\ Y_{k+1} &= Y_k - \eta_k D_Y f(X_{k+1}, Y_k) && (14.61) \end{aligned}$$

- stops if certain stopping criterion is satisfied
- update $k := k + 1$ and repeat iterations

Note the difference in (14.59) and (14.61). This method updates Y with most recent X values.

14.2.2.4 Matrix factorization via stochastic gradient descent (SGD) method

Theoretically both GD and alternating GD converge (to local minima) with proper learning rate strategies. However, the purpose of the collaborative filtering is not minimize (the sum of) errors (or loss function values), but accurately predict missing ratings, *i.e.*, the ratings that a customer has never given, or sometimes accurately predict the ranking of the items. In this case, what we want to achieve is the solution to the following stochastic optimization problem.

$$\text{minimize} \quad \mathbf{E} \textstyle\sum_{i,j:R_{i,j}=\text{NaN}} l(\tilde{R}_{i,j}, x_i^T y_j) \tag{14.62}$$

where $\mathbf{E}(\cdot)$ denotes the expected value (or mean/average) of a random variable and $\tilde{R}_{i,j}$ denotes the rating that the ith customer would give to jth item in the future.

Unfortunately there is no direct way to solve this problem because this stochastic optimization problem is not (exactly) solvable. Most importantly, we do not have the future data (some of them would be never available). Therefore solving (14.48) is not what we want.

There is also another problem with solving (14.48); computational cost per iteration. The gradient evaluation takes $3n_Ln_R$ multiplications and $3n_Ln_R$ additions where n_R refers to the number of known ratings. Thus if the density of R is α, the number of multiplications and additions required to evaluate the gradient is $3\alpha n_L n_C n_I$. Therefore, when n_C or n_I or both are huge, the cost for the gradient calculation can be huge.

To resolve these two problems at the same time, we can use stochastic gradient descent (SGD) method with mini-batch methods. The mini-batch method is to use fixed size of training sets for each gradient descent iteration. This can save the computational cost considerably while approximately solving the stochastic optimization problem (14.62).

14.2.2.5 Matrix factorization via alternating least-squares (ALS)

When we use the squared loss, we solve the problem (14.53). The objective function of this problem is

$$f(X,Y) = \|R - XY^T\|_{F,R}^2 \tag{14.63}$$

This is a quadratic function of a bilinear function of X and Y. Unfortunately, it is not a convex function. If it were a convex function, probably we would not need to use (stochastic) gradient descent method because we have much more efficient methods to solve it (*e.g.*, Newton's method). And if it were a convex function, we would be able to obtain the global minimum (even with gradient descent method).

However, if we fix either X or Y, it becomes the convex function of the other. Indeed, the problem (14.53) becomes a least-squares problem when one of X and Y is fixed, which leads to the alternating least-squares algorithm.

- choose initial $\tilde{X}$ and $\tilde{Y}$
- let $X_0 := \tilde{X}$ and $Y_0 := \tilde{Y}$
- let $k := 0$
- for each iteration, update X and Y

$$X_{k+1} = \operatorname*{argmin}_X \|R - X{Y_k}^T\|_{F,R} \tag{14.64}$$

$$Y_{k+1} = \operatorname*{argmin}_Y \|R - X_{k+1}Y^T\|_{F,R} \tag{14.65}$$

- stops if certain stopping criterion is satisfied
- update $k := k + 1$ and repeat iterations

The ALS is named so since solving (14.64) or (14.65) is equivalent to solving a least-squares problem. Note that the functions in (14.64) and (14.65) can be separated by x_is or y_js, *e.g.*, the function in (14.64) is

$$f_Y(X) = \|R - XY^T\|_{F,R}^2 = \sum_{i=1}^{n_C} \|\tilde{r}_i^T - x_i^T Y^T\|_{F,\tilde{r}_i^T}^2 = \sum_{i=1}^{n_C} \|\tilde{r}_i - Yx_i\|_{F,\tilde{r}_i}^2 \tag{14.66}$$

where $\tilde{r}_i \in \mathbf{R}^{n_I}$ are the row vectors of R, thus solving (14.64) is equivalent to solving n_C uncorrelated problems separately. This is another great advantage of ALS since we can use parallelism to save the training time considerably. For example, if we have N processing units, the training time per iteration would be equivalent to that for solving each (small) least-squares n_C/N times. Note that we cannot separate the objective function in (14.63) when we consider X and Y simultaneously since each of n_R terms in (14.63) are intertwined through x_is and y_js.

When $R \in \mathbf{R}^{n_C \times n_I}$, it can be easily shown that

$$x_i^* = Y^\dagger \tilde{r}_i \tag{14.67}$$

where

$$Y^\dagger = (Y^T Y)^{-1} Y^T \tag{14.68}$$

is the pseudo-inverse of Y. Thus (14.64) becomes

$$X_{k+1} = R{Y_k^\dagger}^T = RY_k \left({Y_k}^T Y_k\right)^{-1}. \tag{14.69}$$

Likewise, (14.65) becomes

$$Y_{k+1} = R^T {X_{k+1}^\dagger}^T = R^T X_{k+1} \left({X_{k+1}}^T X_{k+1}\right)^{-1}. \tag{14.70}$$

Note that in practice, we do not evaluate the pseudo-inverse as in (14.68) because it causes catastrophic numerical instability especially when the matrix is huge (as in most recommendation system cases). Instead, we use QR decomposition, *i.e.*, if $Y = QR$ is the QR decomposition of Y, $Y^\dagger = R^{-1}Q^T$.

We now discuss the interpretation of (14.69) and (14.70). Let X^* and Y^* be the optimal solutions for the problem (14.38). Then these should satisfy (14.64) and (14.65), *i.e.*,

$$\begin{aligned} X^* &= \operatorname*{argmin}_X \|R - X{Y^*}^T\|_{F,R} && (14.71)\\ Y^* &= \operatorname*{argmin}_Y \|R - X^*Y^T\|_{F,R} && (14.72) \end{aligned}$$

Then (14.69) and (14.70) imply that

$$\begin{aligned} X^*{Y^*}^T &= RY^*({Y^*}^TY^*)^{-1}{Y^*}^T = RY^{\mathrm{sim}} && (14.73)\\ &= X^*({X^*}^TX^*)^{-1}{X^*}^TR = X^{\mathrm{sim}}R && (14.74) \end{aligned}$$

where X^{sim} and Y^{sim}

$$\begin{aligned} X^{\mathrm{sim}} &= X^*({X^*}^TX^*)^{-1}{X^*}^T \in \mathbf{R}^{n_C\times n_C} && (14.75)\\ Y^{\mathrm{sim}} &= Y^*({Y^*}^TY^*)^{-1}{Y^*}^T \in \mathbf{R}^{n_I\times n_I} && (14.76) \end{aligned}$$

These yield very interesting interpretations for X^{sim} and Y^{sim}. These equations imply that the optimal prediction for the rating of the jth item by the ith customer is

$$\begin{aligned} {x_i^*}^T y_j^* &= \sum_{k=1}^{n_I} Y^{\mathrm{sim}}_{k,j} R_{i,k} && (14.77)\\ &= \sum_{k=1}^{n_C} X^{\mathrm{sim}}_{i,k} R_{k,j} && (14.78) \end{aligned}$$

The equation (14.77) tells us that the optimal prediction is the weighted sum of the ith customer's ratings where the weights are determined by Y^{sim}. This is *exactly the same as the prediction by item-based collaborative filtering* where item-to-item similarity matrix is given by Y^{sim}. On the other hand, the equation (14.78) tells us that the optimal prediction is the weighted sum of the jth item ratings where the weights are determined by X^{sim}. This is *exactly the same as the prediction by user-based collaborative filtering* where user-to-user similarity matrix is given by X^{sim}(which we have not covered in this document).

14.2.2.6 Weighted matrix factorization via alternating least-squares (ALS)

Here we consider the weighted norm for the objective function.

$$f_W(X,Y) = \|W \bullet (R - XY^T)\|_{F,R}^2 \tag{14.79}$$

where $W \in {\mathbf{R}_+}^{n_C \times n_I}$ and $\bullet$ denotes the component-wise multiplication.

14.2.3 Collaborative filtering for implicit feedback dataset

- binarized variables

$$p_{i,j} = \left\{ \begin{array}{ll} 1 & R_{i,j} \in \mathbf{R} \\ 0 & R_{i,j} \not\in \mathbf{R} \end{array} \right. \tag{14.80}$$

- confidence variables

$$c_{i,j} = 1 + \alpha R_{i,j} \tag{14.81}$$

or

$$c_{i,j} = 1 + \alpha \log(1 + R_{i,j}/\epsilon) \tag{14.82}$$

- objective function

$$f(X,Y;\lambda) = \sum_{i=1}^{n_C} \sum_{j=1}^{n_I} c_{i,j}(p_{i,j} - x_i^T y_j)^2 + \lambda_X \|X\|_F^2 + \lambda_Y \|Y\|_F^2 \tag{14.83}$$

where $\lambda = (\lambda_X, \lambda_Y) \in {\mathbf{R}_{++}}^2$ is the coefficient for the regularization.

- gradients

$$\begin{aligned} \nabla_{x_i} f(X,Y;\lambda) &= -2 \sum_{j=1}^{n_I} c_{i,j}(p_{i,j} - x_i^T y_j) y_j + 2\lambda_X x_i \\ &= 2 \left(\left(\sum_{j=1}^{n_I} c_{i,j} y_j y_j^T + \lambda_X I_{n_L} \right) x_i - \sum_{j=1}^{n_I} c_{i,j} p_{i,j} y_j \right) \\ &= 2 \left(\left(Y^T \mathbf{diag}(\tilde{c}_i) Y + \lambda_X I_{n_L} \right) x_i - Y^T \mathbf{diag}(\tilde{c}_i) \tilde{p}_i \right) \end{aligned} \tag{14.84}$$

Likewise,

$$\nabla_{y_j} f(X,Y;\lambda) = 2 \left(\left(X^T \mathbf{diag}(c_j) X + \lambda_Y I_{n_L} \right) y_i - X^T \mathbf{diag}(c_j) p_j \right) \tag{14.85}$$

Here $\tilde{c}_i \in \mathbf{R}^{n_I}$ and $c_j \in \mathbf{R}^{n_C}$ are the ith row vector and the jth column vector of $C \in \mathbf{R}^{n_C \times n_I}$ respectively, and $\tilde{p}_i \in \mathbf{R}^{n_I}$ and $p_j \in \mathbf{R}^{n_C}$ are the ith row vector and the jth column vector of $P \in \mathbf{R}^{n_C \times n_I}$ respectively, *i.e.*,

$$C = \begin{bmatrix} \tilde{c}_1 \\ \tilde{c}_2 \\ \vdots \\ \tilde{c}_{n_C} \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & \cdots & c_{n_C} \end{bmatrix} \in \mathbf{R}^{n_C \times n_I} \tag{14.86}$$

$$P = \begin{bmatrix} \tilde{p}_1 \\ \tilde{p}_2 \\ \vdots \\ \tilde{p}_{n_C} \end{bmatrix} = \begin{bmatrix} p_1 & p_2 & \cdots & p_{n_C} \end{bmatrix} \in \mathbf{R}^{n_C \times n_I} \tag{14.87}$$

14.2.3.1 Regularization coefficient conversion

$$f(X,Y;\lambda) = \sum_{i=1}^{n_C} \sum_{j=1}^{n_I} c_{i,j}(p_{i,j} - x_i^T y_j)^2 + \lambda_X \|X\|_F^2 + \lambda_Y \|Y\|_F^2 \tag{14.88}$$

where $\lambda = (\lambda_X, \lambda_Y) \in {\mathbf{R}_+}^2$.

Suppose that $(X_\lambda^*, Y_\lambda^*)$ is the optimal solution for the following problem:

$$\text{minimize} \quad f(X,Y;\lambda) = \sum_{i=1}^{n_C} \sum_{j=1}^{n_I} c_{i,j}(p_{i,j} - x_i^T y_j)^2 + \lambda_X \|X\|_F^2 + \lambda_Y \|Y\|_F^2 \tag{14.89}$$

for some λ. Since for any X, Y, and $a \neq 0$,

$$f(aX, (1/a)Y; (1/a^2)\lambda_X, a^2\lambda_Y) = f(X,Y;\lambda_X,\lambda_Y), \tag{14.90}$$

for any X and Y,

$$\begin{aligned} f(aX_\lambda^*, (1/a)Y_\lambda^*; (1/a^2)\lambda_X, a^2\lambda_Y) &= f(X_\lambda^*, Y_\lambda^*; \lambda_X, \lambda_Y) \\ \leq \quad f((1/a)X, aY; \lambda_X, \lambda_Y) &= f(X,Y;(1/a^2)\lambda_X, a^2\lambda_Y). \end{aligned}$$

Therefore $(aX_\lambda^*, (1/a)Y_\lambda^*)$ is the optimal solution for the following problem:

$$\text{minimize} \quad f(X,Y;(1/a^2)\lambda_X, a^2\lambda_Y) \tag{14.91}$$

Therefore if we obtain an optimal solution for a certain $(\tilde{\lambda}_X, \tilde{\lambda}_Y)$, we have readily obtained optimal solutions for all (λ_X, λ_Y) pairs such that $\lambda_X\lambda_Y = \tilde{\lambda}_X\tilde{\lambda}_Y$.

Therefore solving the problem (14.89) is equivalent to solving the following optimization problem:

$$\text{minimize} \quad f(X,Y;\lambda) = \sum_{i=1}^{n_C} \sum_{j=1}^{n_I} c_{i,j}(p_{i,j} - x_i^T y_j)^2 + \sqrt{\lambda_X\lambda_Y}(\|X\|_F^2 + \|Y\|_F^2) \tag{14.92}$$

Now if assume that we have an optimal regularization coefficient $\lambda \in \mathbf{R}_+$ for the following ML problem:

$$\text{minimize} \quad f(X,Y;\lambda) = \sum_{i=1}^{n_C} \sum_{j=1}^{n_I} c_{i,j}(p_{i,j} - x_i^T y_j)^2 + \lambda(\|X\|_F^2 + \|Y\|_F^2) \tag{14.93}$$

If we consider the normalization of each of the three terms in the objective function, we can formulate the problem as follows.

$$\text{minimize} \quad f(X,Y;\lambda) = \textstyle\sum_{i=1}^{n_C} \sum_{j=1}^{n_I} c_{i,j}(p_{i,j} - x_i^T y_j)^2/n_C n_I + \lambda(\|X\|_F^2/n_C n_L + \|Y\|_F^2/n_I n_L) \tag{14.94}$$

which is equivalent to

$$\text{minimize} \quad f(X,Y;\lambda) = \textstyle\sum_{i=1}^{n_C} \sum_{j=1}^{n_I} c_{i,j}(p_{i,j} - x_i^T y_j)^2 + \lambda((n_I/n_L)\|X\|_F^2 + (n_C/n_L)\|Y\|_F^2) \tag{14.95}$$

which again is equivalent to

$$\text{minimize} \quad f(X,Y;\lambda) = \textstyle\sum_{i=1}^{n_C} \sum_{j=1}^{n_I} c_{i,j}(p_{i,j} - x_i^T y_j)^2 + (\lambda\sqrt{n_C n_I}/n_L)(\|X\|_F^2 + \|Y\|_F^2) \tag{14.96}$$

Now assume that $\lambda^*(n_C, n_I, n_L)$ is the optimal values for λ in (14.93) (which, for example, can be approximately obtained by hyperparameter optimization with the formulation (14.93)). Then, for some other values $({n_C}', {n_I}', {n_L}')$, (14.96) implies that the approximate optimal λ can be found by

$$\begin{aligned} & \lambda^*(n_C, n_I, n_L)\sqrt{n_C n_I}/n_L = \lambda^*({n_C}', {n_I}', {n_L}')\sqrt{{n_C}'{n_I}'}/{n_L}' \\ \Leftrightarrow \quad & \lambda^*({n_C}', {n_I}', {n_L}') = \lambda^*(n_C, n_I, n_L){n_L}'\sqrt{n_C n_I}/n_L\sqrt{{n_C}'{n_I}'} \end{aligned}$$

So for example, if we have $n_I = {n_I}'$ and $n_L = {n_L}'$, then

$$\lambda^*({n_C}', {n_I}', {n_L}') = \sqrt{\frac{n_C}{{n_C}'}}\lambda^*(n_C, n_I, n_L). \tag{14.97}$$

14.3 Appendix

14.3.1 Linear algebra

14.3.1.1 Singular value decomposition (SVD)

For any rank-k matrix $A \in \mathbf{R}^{m\times n}$, there exist three matrix $U \in \mathbf{R}^{m\times k}$, $\Sigma \in \mathbf{R}^{k\times k}$, and $V \in \mathbf{R}^{n\times k}$, such that

$$A = U\Sigma V^T \tag{14.98}$$

where the columns of U are orthonormal, the columns of V are orthonormal, and Σ is a diagonal matrix with nonincreasing positive diagonal entries, *i.e.*,

$$U^T U = V^T V = I_k \in \mathbf{R}^{k\times k} \tag{14.99}$$

and

$$\Sigma = \mathbf{diag}(\sigma_1, \ldots, \sigma_k) = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_k \end{bmatrix} \in \mathbf{R}^{k\times k} \tag{14.100}$$

with

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k > 0 \tag{14.101}$$

where I_k referst to k-by-k identity matrix.

If we let $u_1, \ldots, u_k \in \mathbf{R}^n$ be the k column vectors of U, *i.e.*,

$$U = \begin{bmatrix} u_1 & \cdots & u_k \end{bmatrix} \in \mathbf{R}^{n\times k}, \tag{14.102}$$

$U^T U = I_k$ holds if and only if

$$u_i^T u_j = \delta_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{14.103}$$

where $\delta_{i,j}$ denotes the Kronecker delta, which means the length of each u_i is 1 and all of them are orthogonal to each other. Likewise, if we let $v_1, \ldots, v_k \in \mathbf{R}^m$ be the k column vectors of V, *i.e.*,

$$V = \begin{bmatrix} v_1 & \cdots & v_k \end{bmatrix} \in \mathbf{R}^{m\times k}, \tag{14.104}$$

$V^T V = I_k$ holds if and only if

$$v_i^T v_j = \delta_{i,j} \tag{14.105}$$

which means the length of each v_i is 1 and all of them are orthogonal to each other. Note that

- $\sigma_1, \ldots, \sigma_k$ are called *singular values* of A.
- $u_1, \ldots, u_k$ are called *left singular vectors* of A.
- $v_1, \ldots, v_k$ are called *right singular vectors* of A.

Note also that A can be expressed as

$$A = \sum_{i=1}^{k} \sigma_i u_i v_i^T, \tag{14.106}$$

i.e., A can be express as a linear combination of k rank-1 matrices.

14.3.1.2 Singular value decomposition as rank-k approximation

Given a matrix $A \in \mathbf{R}^{m\times n}$ where $\mathbf{rank}(A) = k$, consider the following optimization problem with $r \leq k$.

$$\begin{array}{ll} \text{minimize} & \|A - B\|_F \\ \text{subject to} & \mathbf{rank}(B) = r \end{array} \tag{14.107}$$

where the optimization variable is $B \in \mathbf{R}^{m\times n}$ and $\|\cdot\|_F$ denotes the Frobenius norm.

It turns out that

$$B^* = U_r \Sigma_r V_r^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T, \tag{14.108}$$

is an optimal solution for the problem (14.107), *i.e.*,

$$\|A - U_r \Sigma_r V_r^T\|_F \leq \|A - B\|_F \tag{14.109}$$

for every B with $\mathbf{rank}(B) = r$ where $U_r \in \mathbf{R}^{m\times r}$, $V_r \in \mathbf{R}^{n\times r}$, and $\Sigma_r \in \mathbf{R}^{r\times r}$ are defined as

$$U_r = \begin{bmatrix} u_1 & \cdots & u_r \end{bmatrix} \in \mathbf{R}^{n\times r} \tag{14.110}$$

$$V_r = \begin{bmatrix} v_1 & \cdots & v_r \end{bmatrix} \in \mathbf{R}^{m\times r} \tag{14.111}$$

$$\Sigma_r = \mathbf{diag}(\sigma_1, \ldots, \sigma_r) \in \mathbf{R}^{r\times r} \tag{14.112}$$

This means the close rank-r matrix to A (in Frobenius norm sense) can be obtained with k largest singular values together with corresponding k left and right singular vectors.

Chapter 15

Time Series Anomaly Detection

15.1 Real-Time Anomaly Detection

15.1.1 Computing Anomaly Likelihood

At every time step, we evaluate the moving sample means and the moving sample standard deviations:

$$\mu(t) = \frac{1}{w}\sum_{i=0}^{w-1} s(t-i) \tag{15.1}$$

$$\sigma(t) = \sqrt{\frac{1}{w-1}\sum_{i=0}^{w-1}(s(t-i)-\mu(t))^2} \tag{15.2}$$

Then compute a recent short term average of raw anomaly scores, and apply a threshold to the Gaussian tail probability (Q-function) to decided whether or not to declare an anomaly:

$$L(t) = 1 - Q\left(\frac{\tilde{\mu}(t)-\mu(t)}{\sigma(t)}\right) \tag{15.3}$$

where the short-term moving sample mean is defined by

$$\tilde{\mu}(t) = \frac{1}{w'}\sum_{0}^{w'-1} s(t-i) \tag{15.4}$$

and the Q-function is defined by

$$Q(x) = \frac{1}{\sqrt{2\pi}}\int_x^\infty \exp\left(-\frac{\tau^2}{2}\right) d\tau. \tag{15.5}$$

We threshold $L(t)$ and report an anomaly if it is very close to 1:

$$L(t) > 1 - \epsilon. \tag{15.6}$$

To take longer history data into considering while simutaneously emphasizing recent data more, we can consider the exponentially weighted moving sample mean and standard deviation:

$$\mu(t) = (1-\gamma)\sum_{i=0}^{\infty}\gamma^i s(t-i) \tag{15.7}$$

$$\sigma(t) = \sqrt{(1-\gamma)\sum_{i=0}^{\infty}\gamma^i(s(t-i)-\mu(t))^2} \tag{15.8}$$

To combine the advantages of finite window method and exponentially weighted method, we can consider the exponentially weighted moving sample mean and standard deviation with finite window as follows:

$$\mu(t) = \frac{1}{\sum_{i=0}^{w-1} \gamma^i} \sum_{i=0}^{w-1} \gamma^i s(t-i) \tag{15.9}$$

$$\sigma(t) = \sqrt{\frac{1}{\sum_{i=0}^{w-1} \gamma^i} \sum_{i=0}^{w-1} \gamma^i (s(t-i) - \mu(t))^2} \tag{15.10}$$

Chapter 16

Reinforcement Learning

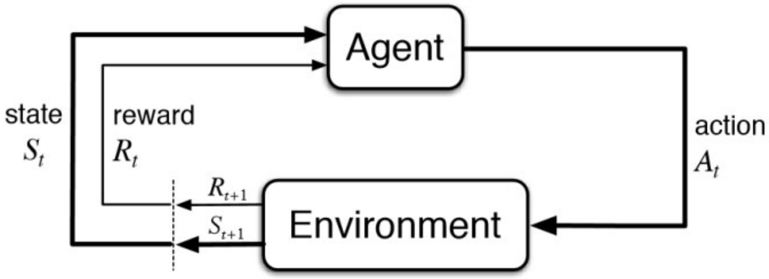

Figure 16.1: The agent-environment interaction in a Markov decision process.

The reinforcement learning is a machine learning where an agent learns how to take actions to achieve a goal by maximizing cumulative reward while interacting with environment. Learning from interaction is a foundational idea underlying nearly all theories of learning and intelligence.

It differs from supervised learning in that labeled input and output pairs need not be presented (and sub-optimal actions need not be explicitly corrected). Instead the focus is finding a balance between exploration of uncharted territory and exploitation of current knowledge. It is much more focused on goal-directed learning from interaction than other approaches to machine learning.

In the following sections, we introduce finite Markov decision process (MDP) and three typical methods to solve reinforcement learning problems. All these methods are called tabular solution methods because they need to store values for all the states or all the state-action pairs (that have been visited).

16.1 Finite Markov decision processes

We introduce the formal problem of finite Markov decision processes (MDPs), which we try to solve. This problem involves evaluative feedback, but also an associative aspect, *i.e.*, choosing different actions in different situations. MDP is a classical formalization of sequential decision making, where actions influence not just immediate rewards, but also subsequent states through those future rewards. Thus MDPs involve delayed reward and the need to trade-off immediate and delayed reward.

Figure 16.1 depicts MDP where an agent interacts with environment. The current state of the environment is known to the agent. With knowledge of state, the agent makes a decision as to which action to take. This action, in turn, will change the state of the environment and the agent will receive a reward at the same time. The agent remembers this reward in some indirect way and uses it for making future decisions.

16.1.1 Markov property

Suppose that the agent is in state S_t takes action A_t at time t. Then the agent receives reward R_{t+1} (from the environment) and the environment transitions to state S_{t+1}. MDP assumes that all these quantities are random variables.

Let $\mathcal{S}$ and $\mathcal{A}$ be the set of all the states and that of all the actions the agent can take respectively.

Now suppose that the environment is in state $S_0 \in \mathcal{S}$ the agent takes action $A_0 \in \mathcal{A}$ at $t = 0$. Then the state of the environment becomes $S_1 \in \mathcal{S}$ giving the agent $R_1 \in \mathbf{R}$ as reward. Suppose that the agent repeat taking actions.

Then we have a sequence of random variables

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, \ldots \tag{16.1}$$

We assume that these random variables satisfy the Markov Property (as assumed by the name) in the following sense.

$$S_{t+1}, R_{t+1} | S_t, A_t, R_t, S_{t-1}, A_{t-1}, R_{t-1}, \ldots = S_{t+1}, R_{t+1} | S_t, A_t \tag{16.2}$$

i.e., two random variables, S_{t+1} and R_{t+1}, conditioned on every state, action, and reward before $t+1$ are the same as those conditioned on S_t and R_t only.

This can be formally expressed using the probability density function (PDF) as follows.

$$p\left(S_{t+1}, R_{t+1} \middle| \, S_t, A_t, R_t, S_{t-1}, A_{t-1}, R_{t-1}, \ldots \right) = p\left(S_{t+1}, R_{t+1} \middle| \, S_t, A_t \right). \tag{16.3}$$

This is the reason that the process is called *Markov* decision process.

16.1.2 Policy

The *policy* is defined by the conditional probability of A_t given S_t, *i.e.*,

$$\pi(A|S) = p(A_t|S_t), \tag{16.4}$$

which implies the probability of taking certain action depends only on the current state, not the time. The policy decides which actions the agent takes in each state.

Let Π be the set of all the policies.

16.1.3 Return

The *return* at t is defined by

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \tag{16.5}$$

where $\gamma \in [0,1]$ is called the *discount factor.* If $\gamma = 0$, the agent is myopic, *i.e.*, it only cares the immediate reward. If $\gamma = 1$, the agent is truly far-sighted, *i.e.*, it cares all the future rewards without discounting. If γ is somewhere between 0 and 1, it considers near-future rewards more importantly than those in far future.

16.1.4 State value function and action value function

The state value function (which is sometimes referred to as just value function) is defined by

$$v_\pi(s) = \mathbf{E}_{\pi,p}\left\{ G_t \middle| S_t = s \right\} = \mathbf{E}_{\pi,p}\left\{ \sum_{k=0}^{\infty} \gamma^k R_{t+k} \middle| S_t = s \right\}. \tag{16.6}$$

In other words, the state value function is a function of a state representing the expected return the agent will get from the state when following the policy π.

The action value function (which is sometimes referred to as just action function) is defined by

$$q_\pi(s,a) = \mathbf{E}_{\pi,p}\left\{ G_t \middle| S_t = s, A_t = a \right\} = \mathbf{E}_{\pi,p}\left\{ \sum_{k=0}^{\infty} \gamma^k R_{t+k} \middle| S_t = s, A_t = a \right\}. \tag{16.7}$$

In other words, the action value function is a function of a state and an action representing the expected return the agent will get from the state when the agent takes a certain action and follows the policy π.

As mentioned above, most reinforcement learning algorithms try to maximize either one of these functions, *i.e.*, not maximizing the immediate reward, but the long-term return.

16.2 Bellman equation

Richard E. Bellman, who introduced dynamic programming in 1953, proposed an equation as a necessary condition for optimality associated with dynamic programming, which is called Bellman equation. One of the properties that Markov property implies is that the value functions only depend on the current state (and the action taken) and that the function value is closely related to the function values of the next states. These facts are cleverly used to derive the Bellman equation. Here we introduce two Bellman equations; one for the state value function and the other for action value function.

16.2.1 Bellman equations

To derive Bellman equations, we use some basic statistics facts regarding conditional expectations. (Refer to §16.12.)

Since the definitions of state value function and action value function together with (16.62) imply

$$\begin{aligned}
v_\pi(s) &= \mathbf{E}_{\pi,p}\left\{ G_t \middle| S_t = s \right\} \\
&= \mathbf{E}_{A_t|S_t=s}\mathbf{E}_{\pi,p}\left\{ G_t \middle| S_t = s, A_t \right\} \\
&= \sum_a p(A_t = a|S_t = s)\mathbf{E}_{\pi,p}\left\{ G_t \middle| S_t = s, A_t = a \right\} \\
&= \sum_a \pi(a|s)\mathbf{E}_{\pi,p}\left\{ G_t \middle| S_t = s, A_t = a \right\} \\
&= \sum_a \pi(a|s) q_\pi(s,a)
\end{aligned}$$

and

$$\begin{aligned}
q_\pi(s,a) &= \mathbf{E}_{\pi,p}\left\{G_t \middle| S_t = s, A_t = a\right\} \\
&= \mathbf{E}_{S_{t+1},R_{t+1}|S_t=s,A_t=a}\mathbf{E}_{\pi,p}\left\{G_t \middle| S_t = s, A_t = a, S_{t+1}, R_{t+1}\right\} \\
&= \mathbf{E}_{S_{t+1},R_{t+1}|S_t=s,A_t=a}\mathbf{E}_{\pi,p}\left\{\sum_{k=0}^{\infty}\gamma^k R_{t+k+1} \middle| S_t = s, A_t = a, S_{t+1}, R_{t+1}\right\} \\
&= \mathbf{E}_{S_{t+1},R_{t+1}|S_t=s,A_t=a}\mathbf{E}_{\pi,p}\left\{R_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k R_{t+k+2} \middle| S_t = s, A_t = a, S_{t+1}, R_{t+1}\right\} \\
&= \sum_{s',r} p_{S_{t+1},R_{t+1}|S_t,A_t}(s',r|s,a)\mathbf{E}_{\pi,p}\left\{R_{t+1} + \gamma G_{t+1} \middle| S_t = s, A_t = a, S_{t+1} = s', R_{t+1} = r\right\} \\
&= \sum_{s',r} p_{S_{t+1},R_{t+1}|S_t,A_t}(s',r|s,a)\left(r + \gamma\mathbf{E}_{\pi,p}\left\{G_{t+1} \middle| S_t = s, A_t = a, S_{t+1} = s', R_{t+1} = r\right\}\right) \\
&= \sum_{s',r} p_{S_{t+1},R_{t+1}|S_t,A_t}(s',r|s,a)\left(r + \gamma\mathbf{E}_{\pi,p}\left\{G_{t+1} \middle| S_{t+1} = s'\right\}\right) \\
&= \sum_{s',r} p_{S_{t+1},R_{t+1}|S_t,A_t}(s',r|s,a)\left(r + \gamma v_\pi(s')\right),
\end{aligned}$$

we have the following two equations relating state value function to action value function and vise versa.

$$v_\pi(s) = \sum_a \pi(a|s)q_\pi(s,a). \tag{16.8}$$

$$q_\pi(s,a) = \sum_{s',r} p_{S_{t+1},R_{t+1}|S_t,A_t}(s',r|s,a)\left(r + \gamma v_\pi(s')\right). \tag{16.9}$$

Now (16.8) and (16.9) imply that

$$v_\pi(s) = \sum_a \pi(a|s)q_\pi(s,a) = \sum_a \pi(a|s)\sum_{s',r} p(s',r|s,a)\left(r + \gamma v_\pi(s')\right) \tag{16.10}$$

and

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)\left(r + \gamma v_\pi(s')\right) = \sum_{s',r} p(s',r|s,a)\left(r + \gamma\sum_{a'}\pi(a'|s')q_\pi(s',a')\right). \tag{16.11}$$

The equation (16.10) is called *Bellman equation for state value function* and the equation (16.11) is called *Bellman equation for action value function.*

16.2.2 Bellman optimality equations

Now suppose that the policy π_* is the optimal policy. Then we define the *optimal state-value function* as that of π_*, *i.e.*,

$$v_*(s) = v_{\pi_*}(s) = \max_{\pi\in\Pi} v_\pi(s). \tag{16.12}$$

Likewise, we define the *optimal action-value function* as that of π_*, *i.e.*,

$$q_*(s,a) = q_{\pi_*}(s,a) = \max_{\pi \in \Pi} q_\pi(s,a). \tag{16.13}$$

Then (16.8) and (16.9) imply that

$$v_*(s) = v_{\pi_*}(a) = \max_{a \in \mathcal{A}} q_{\pi_*}(s,a) = \max_{a \in \mathcal{A}} \sum_{s',r} p(s',r|s,a)\left(r + \gamma v_\pi(s')\right). \tag{16.14}$$

and

$$q_*(s,a) = q_{\pi_*}(s,a) = \sum_{s',r} p(s',r|s,a)\left(r + \gamma v_{\pi_*}(s')\right). = \sum_{s',r} p(s',r|s,a)\left(r + \gamma \max_{a' \in \mathcal{A}} q_{\pi_*}(s',a')\right). \tag{16.15}$$

The equation (16.14) is called *Bellman optimality equation for state value function* and the equation (16.15) is called *Bellman optimality equation for action value function.*

16.3 Dynamic programming

The term dynamic programming (DP) refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov decision process (MDP). DP provides an essential foundation for the understanding of the methods presented in the rest of this chapter. All of these methods can be viewed as attempts to achieve much the same effect as DP, only with less computation and without assuming a perfect model of the environment.

The key idea of DP, and of reinforcement learning generally, is the use of value functions to organize and structure the search for good policies.

16.3.1 Policy evaluation (prediction)

We consider how to compute the state-value function v_π for an arbitrary policy π. This is called *policy evaluation* in the DP literature. We also refer to it as the *prediction problem.* The existence and uniqueness of v_π are guaranteed as long as either $\gamma < 1$ or eventual termination is guaranteed from all states under the policy π.

The policy evaluation algorithm uses the fact that all the state value functions satisfy the Bellman equation for state value function. We use this equation, but in an iterative manner.

$$v_{k+1}(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\left(r + \gamma v_k(s')\right). \tag{16.16}$$

This equation resembles (16.10), but different because now we put subscript k in place of the policy π. Indeed, the sequence v_k can be shown in general to converge to v_π as k goes to ∞ the same conditions that guarantee the existence of v_π. This algorithm is called *iterative policy evaluation.*

We can consider *in-place* version of this algorithm, *i.e.*, we replace the values for v_k for each state without waiting until we sweep all the states for one iteration. This in-place algorithm also converges to v_π. In fact, it usually converges faster. This in-place algorithm is described in Table 16.1.

```
Inputs: $\pi$, MDP
Algorithm parameters: $\theta > 0$ (small threshold determining accuracy of estimation)

Initialize $V(s) \in \mathbf{R}$ for all $s \in \mathcal{S}$ except that $V(\text{terminal}) = 0$

Loop:
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left(r + \gamma V(s')\right)$
        $\Delta \leftarrow \max\{\Delta, |v - V(s)|\}$
until $\Delta < \theta$
```

Table 16.1: Iterative Policy Evaluation for estimating $V \sim v_\pi$.

16.3.2 Policy iteration

The policy iteration is the iterative process of improving policy as to maximize the value functions. The algorithm is described in Table 16.2.

16.3.3 Value iteration

One drawback to policy iteration is that each of its iterations involves policy evaluation. In fact, the policy evaluation step of policy iteration can be truncated in several ways without losing the convergence guarantees of policy iteration. One important special case is when policy evaluation is stopped after just one sweep (one update of each state). This algorithm is called *value iteration*. It can be written as a particularly simple update operation that combines the policy improvement and truncated policy evaluation steps.

$$v_{k+1}(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s',r} p(s', r|s, a) \left(r + \gamma v_k(s')\right). \tag{16.17}$$

Note that value iteration is obtained simply by turning the Bellman optimality equation for state value function (16.14) into an update rule.

The in-place version of value iteration algorithm is described in Table 16.3.

16.4 Monte Carlo methods

Here we consider learning methods for estimating value functions and discovering optimal policies. Unlike the previous methods, we do not assume complete knowledge of the environment. Monte Carlo (MC) methods require only experience sample sequences of states, actions, and rewards from *actual or simulated interaction with an environment.* Learning from actual experience is striking because it requires no prior knowledge of the environment's dynamics, yet can still attain optimal behavior. *Learning from simulated experience is also powerful.* Although a model is required, the model need only generate sample transitions, *not the complete probability distributions of all possible transitions* that is required for dynamic programming (DP).

```
Inputs: MDP
Algorithm parameters: $\theta > 0$ (small threshold determining accuracy of estimation)

1. Initialization
    $V(s) \in \mathbf{R}$ and $\pi(s) \in \mathcal{A}(s)$ for all $s \in \mathcal{S}$

2. Policy Evaluation
    Loop:
        $\Delta \leftarrow 0$
        For each $s \in \mathcal{S}$:
            $v \leftarrow V(s)$
            $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \, (r + \gamma V(s'))$
            $\Delta \leftarrow \max\{\Delta, |v - V(s)|\}$
    until $\Delta < \theta$

3. Policy Improvement
$u \leftarrow$ true
    For each $s \in \mathcal{S}$
        $b \leftarrow \pi(s)$
        $\pi(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \, (r + \gamma v_\pi(s'))$
        If $b \neq \pi(s)$, then $t \leftarrow$ false
    If $u$, then stop and return $V \sim v_*$ and $\pi \sim \pi_*$; else go to 2
```

Table 16.2: Policy Iteration (using iterative policy evaluation) for estimating $\pi \sim \pi_*$.

```
Inputs: MDP
Algorithm parameters: $\theta > 0$ (small threshold determining accuracy of estimation)

Initialize $V(s) \in \mathbf{R}$ for all $s \in \mathcal{S}$ except that $V(\text{terminal}) = 0$

Loop:
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s', r|s, a) \, (r + \gamma V(s'))$
        $\Delta \leftarrow \max\{\Delta, |v - V(s)|\}$
until $\Delta < \theta$

Output: deterministic policy $\pi$ such that
    $\pi(s) = \text{argmax}_{a \in \mathcal{A}(s)} \sum_{s',r} p(s', r|s, a) \, (r + \gamma V(s'))$
```

Table 16.3: Value Iteration for estimating $\pi \sim \pi_*$.

```
Inputs: $\pi$

Initialize:
    $V(s) \in \mathbf{R}$ for all $s \in \mathcal{S}$
    $R(s) \leftarrow \texttt{list}()$ for all $s \in \mathcal{S}$

Loop:
    Generate an episode following $\pi$: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t + T - 1, T - 2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        If $S_t \notin \{S_0, S_1, \ldots, S_{t-1}\}$:
            $R(S_t).\texttt{append}(G)$
            $V(S_t) \leftarrow R(S_t).\texttt{average}()$
Until a certain criterion is satisfied
```

Table 16.4: First-visit MC prediction for estimating $V \sim v_\pi$.

MC methods are ways of solving the reinforcement learning problem based on averaging sample returns. To ensure that well-defined returns are available, here we define Monte Carlo methods only for episodic tasks.

Monte Carlo methods sample and average returns for each state–action pair much like the bandit methods where we sample and average rewards for each action. The main difference is that now there are multiple states, each acting like a different bandit problem (like an associative-search or contextual bandit) and the different bandit problems are interrelated. That is, the return after taking an action in one state depends on the actions taken in later states in the same episode. Because all the action selections are undergoing learning, the problem becomes nonstationary from the point of view of the earlier state.

To handle the nonstationarity, we adapt the idea of general policy iteration (GPI) developed for DP. Whereas there we computed value functions from knowledge of the MDP, here we learn value functions from sample returns with the MDP. The value functions and corresponding policies still interact to attain optimality in essentially the same way (GPI). As in DP, first we consider the prediction problem, then policy improvement, and, finally, the control problem and its solution by GPI. Each of these ideas taken from DP is extended to the Monte Carlo case in which only sample experience is available.

16.4.1 Monte Carlo prediction

An obvious way to estimate it from experience, then, is simply to average the returns observed after visits to that state. There are two Monte Carlo (MC) prediction methods; *first-visit MC method* and *every-visit MC method*. These two MC methods are very similar but have slightly different theoretical properties. First-visit MC has been most widely studied, dating back to the 1940s. Every-visit MC extends more naturally to function approximation and eligibility traces.

Table 16.4 describes the first-visit MC prediction algorithm.

```
Initialize:
    $\pi(s) \in \mathcal{A}(s)$ for all $s \in \mathcal{S}$
    $Q(s,a) \in \mathbf{R}$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
    $R(s,a) \leftarrow \texttt{list}()$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Loop:
    Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability $> 0$
    Generate an episode from $S_0$, $A_0$ following $\pi$: $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t + T - 1, T - 2, \dots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        If $S_t \notin \{S_0, S_1, \dots, S_{t-1}\}$:
            $R(S_t, A_t).\texttt{append}(G)$
            $Q(S_t, A_t) \leftarrow R(S_t, A_t).\texttt{average}()$
            $\pi(S_t) \leftarrow \mathrm{argmax}_{a \in \mathcal{A}(S_t)} Q(S_t, a)$
Until a certain criterion is satisfied
```

Table 16.5: MC ES (exploring starts) for estimating $\pi \sim \pi_*$.

16.4.2 Monte Carlo control

The overall idea is to proceed according to the same pattern as in the dynamic programming, *i.e.*, according to the idea of generalized policy iteration (GPI). In GPI one maintains both an approximate policy and an approximate value function. The value function is repeatedly altered to more closely approximate the value function for the current policy, and the policy is repeatedly improved with respect to the current value function. These two kinds of changes work against each other to some extent, as each creates a moving target for the other, but together they cause both policy and value function to approach optimality.

For Monte Carlo policy evaluation it is natural to alternate between evaluation and improvement on an episode-by-episode basis. After each episode, the observed returns are used for policy evaluation, and then the policy is improved at all the states visited in the episode. A complete simple algorithm along these lines, which is called Monte Carlo ES for Monte Carlo with Exploring Starts, is described in Table 16.5.

16.4.3 Monte Carlo control without exploring starts

How can we avoid the unlikely assumption of exploring starts? The only general way to ensure that all actions are selected infinitely often is for the agent to continue to select them. There are two approaches to ensuring this, resulting in what we call on-policy methods and off-policy methods. On-policy methods attempt to evaluate or improve the policy that is used to make decisions, whereas off-policy methods evaluate or improve a policy different from that used to generate the data. The Monte Carlo ES method developed above is an example of an on-policy method. In this section we show how an on-policy Monte Carlo control method can be designed that does not use the unrealistic assumption of exploring starts.

The on-policy first-visit MC control using ϵ-greedy is described in Table 16.6.

```
Algorithm parameters: small $\epsilon > 0$

Initialize:
    $\pi(s) \in \mathcal{A}(s)$ for all $s \in \mathcal{S}$
    $Q(s, a) \in \mathbf{R}$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
    $R(s, a) \leftarrow \texttt{list}()$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Loop:
    Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability $> 0$
    Generate an episode from $S_0$, $A_0$ following $\pi$: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t + T - 1, T - 2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        If $S_t \notin \{S_0, S_1, \ldots, S_{t-1}\}$:
            $R(S_t, A_t).\texttt{append}(G)$
            $Q(S_t, A_t) \leftarrow R(S_t, A_t).\texttt{average}()$
            $A^* \leftarrow \text{argmax}_{a \in \mathcal{A}(S_t)}$
            For all $a \in \mathcal{A}(S_t)$
                $\pi(a|S_t) \leftarrow \begin{cases} 1 - \epsilon + \epsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \epsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$
Until a certain criterion is satisfied
```

Table 16.6: On-policy first-visit MC control (for ϵ-soft policies) for estimating $\pi \sim \pi_*$.

16.4.4 Off-policy prediction via important sampling

XXX

16.4.5 Off-policy Monte Carlo control

XXX

16.5 Temporal-difference learning

Temporal-difference (TD) learning is a combination of Monte Carlo ideas and dynamic programming (DP) ideas. Like Monte Carlo methods, TD methods can learn directly from raw experience without a model of the environment's dynamics. Like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome (they bootstrap). The relationship between TD, DP, and Monte Carlo methods is a recurring theme in the theory of reinforcement learning.

We start by focusing on the policy evaluation or prediction problem, the problem of estimating the value function v_π for a given policy π. For the control problem (finding an optimal policy), DP, TD, and Monte Carlo methods all use some variation of generalized policy iteration (GPI).

16.5.1 TD prediction

Both TD and Monte Carlo methods use experience to solve the prediction problem. A simple every-visit MC method suitable for nonstationary environments is

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t)) = (1-\alpha)V(S_t) + \alpha G_t. \tag{16.18}$$

TD methods need to wait only until the next time step. At time $t+1$, they immediately form a target and make a useful update using the observed reward R_{t+1} and the estimate $V(S_{t+1})$. The simplest TD method makes the update

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) = (1-\alpha)V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1})). \tag{16.19}$$

This TD method is called TD(0), or one-step TD, because it is a special case of the TD(λ) and n-step TD methods. Table 16.7 specifies TD(0) completely in procedural form.

The quantity in brackets in the TD(0) update is a sort of error, measuring the difference between the estimated value of S_t and the better estimate $R_{t+1} + \gamma V(S_{t+1})$. This quantity, called the TD error, arises in various forms throughout reinforcement learning. It can be formally defined as follows.

$$\delta_t \coloneqq R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t) \tag{16.20}$$

It is interesting to observe that we can express Monte Carlo error in terms of modified TD errors if we defined the modified TD error as follows.

$$\delta_t' \coloneqq R_{t+1} + \gamma V_{t+1}(S_{t+1}) - V_t(S_t) \tag{16.21}$$

```
Inputs: the policy $\pi$ to be evaluated
Algorithm parameters: step size $\alpha \in (0, 1]$

Initialize:
    $V(s) \in \mathbf{R}$ for all $s \in \mathcal{S}$ except that $V(\text{terminal}) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        $A \leftarrow$ action given by $\pi$ for $S$
        Take action $A$, observe $R$, $S'$
        $V(S) \leftarrow (1-\alpha)V(S) + \alpha(R + \gamma V(S'))$
        $S \leftarrow S'$
    until $S$ is terminal
Until a certain criterion is satisfied
```

Table 16.7: TD(0) for estimating v_π.

Then the Monte Carlo error is defined by $G_t - V_t(S_t)$ is

$$\begin{aligned}
G_t - V_t(S_t) &= R_{t+1} + \gamma G_{t+1} - V_t(S_t) \\
&= R_{t+1} + \gamma\left(G_{t+1} - V_{t+1}(S_{t+1}) + V_{t+1}(S_{t+1})\right) - V_t(S_t) \\
&= R_{t+1} + \gamma V_{t+1}(S_{t+1}) - V_t(S_t) + \gamma\left(G_{t+1} - V_{t+1}(S_{t+1})\right) \\
&= \delta'_t + \gamma\left(G_{t+1} - V_{t+1}(S_{t+1})\right) \\
&= \delta'_t + \gamma\delta'_{t+1} + \gamma^2\left(G_{t+2} - V_{t+2}(S_{t+2})\right) \\
&= \delta'_t + \gamma\delta'_{t+1} + \gamma^2\delta'_{t+2} + \cdots + \gamma^{T-t-2}\delta'_{T-2} + \gamma^{T-t-1}\left(G_{T-1} - V_{T-1}(S_{T-1})\right) \\
&= \delta'_t + \gamma\delta'_{t+1} + \gamma^2\delta'_{t+2} + \cdots + \gamma^{T-t-2}\delta'_{T-2} + \gamma^{T-t-1}\left(R_T + \gamma V_T(S_T) - V_{T-1}(S_{T-1})\right) \\
&= \delta'_t + \gamma\delta'_{t+1} + \gamma^2\delta'_{t+2} + \cdots + \gamma^{T-t-2}\delta'_{T-2} + \gamma^{T-t-1}\delta'_{T-1} \\
&= \sum_{k=t}^{T-1}\gamma^{k-t}\delta'_k = \sum_{k=0}^{T-t-1}\gamma^k\delta'_{k+t} \qquad (16.22)
\end{aligned}$$

where the fact that the state-value function for a terminal state, $V_{T-1}(S_T)$, is 0 is used.

This means the Monte Carlo error, *i.e.*, the difference between the return along the path from t to a terminal state of the episode and the state-value function of S_t can be expressed as sum of discounted (modified) one-step TD errors. If we assume that every V_t does not change during the episode, δ_t coincides with δ'_t. Hence (16.22) becomes

$$G_t - V(S_t) = \sum_{k=t}^{T-1}\gamma^{k-t}\delta_k = \sum_{k=0}^{T-t-1}\gamma^k\delta_{k+t}. \qquad (16.23)$$

```
Algorithm parameters: step size $\alpha \in (0, 1]$ and small $\epsilon > 0$

Initialize:
    $Q(s, a) \in \mathbf{R}$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$ except $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
    Loop for each step of episode:
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
        $Q(S, A) \leftarrow (1 - \alpha)Q(S, A) + \alpha(R + \gamma Q(S', A'))$
        $S \leftarrow S'$, $A \leftarrow A'$,
    until $S$ is terminal
Until a certain criterion is satisfied
```

Table 16.8: Sarsa (on-policy TD control) for estimating $Q \sim q_*$.

16.5.2 Sarsa: on-policy TD Control

As in all on-policy methods, we continually estimate q_π for the behavior policy π, and at the same time change π toward greediness with respect to q_π.

The convergence properties of the Sarsa algorithm depend on the nature of the policy's dependence on Q. For example, one could use ϵ greedy or ϵ-soft policies. Sarsa converges with probability 1 to an optimal policy and action-value function as long as all state–action pairs are visited an infinite number of times and the policy converges in the limit to the greedy policy (which can be arranged, for example, with ϵ-greedy policies by setting $\epsilon = 1/t$).

This algorithm is described in Table 16.8.

16.5.3 Q-learning: off-policy TD control

One of the early breakthroughs in reinforcement learning was the development of an off-policy TD control algorithm known as Q-learning (Watkins, 1989), defined by

$$\begin{aligned} Q(S_t, A_t) \quad \leftarrow \quad & Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right) \\ & = (1 - \alpha) Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) \right). \end{aligned} \tag{16.24}$$

The learned action-value function, Q, directly approximates q_*, the optimal action-value function, independent of the policy being followed This dramatically simplifies the analysis of the algorithm and enabled early convergence proofs. The policy still has an effect in that it determines which state–action pairs are visited and updated. However, all that is required for correct convergence is that all pairs continue to be updated.

```
Algorithm parameters: step size $\alpha \in (0, 1]$ and small $\epsilon > 0$

Initialize:
    $Q(s, a) \in \mathbf{R}$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$ except $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S, A) \leftarrow (1 - \alpha)Q(S, A) + \alpha(R + \gamma \max_{a \in \mathcal{A}(S')} Q(S', a))$
        $S \leftarrow S'$
    until $S$ is terminal
Until a certain criterion is satisfied
```

Table 16.9: Q-learning (off-policy TD control) for estimating $\pi \sim \pi_*$.

Under this assumption and a variant of the usual stochastic approximation conditions on the sequence of step-size parameters, Q has been shown to converge with probability 1 to q_*. The Q-learning algorithm is described in Table 16.9.

16.5.4 Maximization bias and double learning

XXX

16.6 n-step bootstrapping

There exists another method which unifies the Monte Carlo (MC) methods and the one-step temporal-difference (TD) methods. Neither MC methods nor one-step TD methods are always the best. Here we present n-step TD methods that generalize both methods so that one can shift from one to the other smoothly as needed to meet the demands of a particular task. n-step methods span a spectrum with MC methods at one end and one-step TD methods at the other. The best methods are often intermediate between the two extremes.

Another way of looking at the benefits of n-step methods is that they free one from the tyranny of the time step. With one-step TD methods the same time step determines how often the action can be changed and the time interval over which bootstrapping is done. In many applications one wants to be able to update the action very fast to take into account anything that has changed, but bootstrapping works best if it is over a length of time in which a significant and recognizable state change has occurred. With one-step TD methods, these time intervals are the same, and so a compromise must be made. n-step methods enable bootstrapping to occur over multiple steps, freeing us from the tyranny of the single time step.

The idea of n-step methods is usually used as an introduction to the algorithmic idea of eligibility traces.

16.6.1 n-step TD prediction

n-step TD prediction is a method lying between Monte Carlo and (one-step) TD method, *i.e.*, TD(0). Consider estimating v_π from sample episodes generated using π. Monte Carlo methods perform an update for each state based on the entire sequence of observed rewards from that state until the end of the episode. The update of one-step TD methods, on the other hand, is based on just the one next reward, bootstrapping from the value of the state one step later as a proxy for the remaining rewards.

One kind of intermediate method, then, would perform an update based on an intermediate number of rewards: more than one, but less than all of them until termination.

The methods that use n-step updates are still TD methods because they still change an earlier estimate based on how it differs from a later estimate. Now the later estimate is not one step later, but n steps later. Methods in which the temporal difference extends over n steps are called n-step TD methods.

Suppose that the process is episodic, *i.e.*, ever episode ends or enters a terminal state within finite number of steps. Then the *target* of Monte Carlo update is the return at time step t, *i.e.*,

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T \tag{16.25}$$

where T is the last time step of the episode.

The target of the one-step TD method is the first reward plus the discounted estimated value of the next state, *i.e.*,

$$G_{t:t+1} = R_{t+1} + \gamma V_t(S_{t+1}) \tag{16.26}$$

where $V_t : \mathcal{S} \to \mathbf{R}$ is the estimate of $v_\pi(S_{t+1})$ at time t. Note that the second term $\gamma V_t(S_{t+1})$ is the estimate for $\gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$, but using the state-value function estimate of the next state, instead of using future discounted returns. Thus, this is a bootstrapping. Likewise, we can define two-step return as a target for the two-step update.

$$G_{t:t+2} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2}) \tag{16.27}$$

where $V_{t+1} : \mathcal{S} \to \mathbf{R}$ is the estimate of $v_\pi(S_{t+2})$ at time t+1. Again here the third term $\gamma^2 V_{t+1}(S_{t+2})$ is the estimate for $\gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$ using bootstrapping. In general, we can define the n-step return as the target for the n-step update.

$$G_{t:t+n} = \left\{ \begin{array}{ll} R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}) & \text{if } t+n < T \\ G_t & \text{if } t+n \geq T \end{array} \right. \tag{16.28}$$

for $t \geq 0$. An n-step return can be considered as an approximation to the full return G_t.

We can consider algorithm with the n-step update using this n-step return, *i.e.*,

$$V_{t+n}(S_t) \leftarrow V_{t+n-1}(S_t) + \alpha_{t+n-1}(G_{t:t+n} - V_{t+n-1}(S_t)) = (1 - \alpha_{t+n-1}) V_{t+n-1}(S_t) + \alpha_{t+n-1} G_{t:t+n} \tag{16.29}$$

Note that this update cannot be performed before time step $t+n$ because only by then, all the rewards necessary to evaluate (16.28) become available. Also note that (16.29) is reduced to (16.19)

```
Inputs: policy $\pi$ to be evaluated

Algorithm parameters: step size $\alpha_t \in (0, 1]$ and $n \in \mathbf{N}$

Initialize:
    $V(s) \in \mathbf{R}$ for all $s \in \mathcal{S}$ except that $V(\text{terminal}) = 0$

Loop for each episode:
    Initialize and store $S_0$
    $T \leftarrow \infty$
    Loop for each $t = 0, 1, 2, \ldots$:
        If $t < T$:
            $A_t \leftarrow$ action given by $\pi(\cdot|S_t)$
            Take action $A_t$, observe $R_{t+1}$, $S_{t+1}$
            If $S_{t+1}$ is terminal:
                $T \leftarrow t + 1$
        $\tau \leftarrow t - n + 1$
        If $\tau \geq 0$:
            $G \leftarrow \sum_{i=\tau+1}^{\min\{\tau+n,T\}} \gamma^{i-\tau-1} R_i$
            If $\tau + n < T$:
                $G \leftarrow G + \gamma^n V(S_{\tau+n})$
            $V(S_\tau) \leftarrow (1 - \alpha_t)V(S_\tau) + \alpha_t G$
    while $\tau < T - 1$
Until a certain criterion is satisfied
```

Table 16.10: n-step TD for estimating $V \sim v_\pi$.

```
Inputs: policy $\pi$ to be evaluated

Algorithm parameters: step size $\alpha_t \in (0, 1]$ and $n \in \mathbf{N}$

Initialize:
    $V(s) \in \mathbf{R}$ for all $s \in \mathcal{S}$ except that $V(\text{terminal}) = 0$
    w_list $= [1, \gamma, \ldots, \gamma^n]$

Loop for each episode:
    S_list $\leftarrow$ list(), R_list $\leftarrow$ list()
    Initialize $S$
    S_list.append($S$), not_terminated $\leftarrow True$
    Loop for each $t = 0, 1, 2, \ldots$:
        If not_terminated:
            $A \leftarrow$ action given by $\pi(\cdot|S)$
            Take action $A$, observe $R$, $S'$
            R_list.append($R$)
            If $S$ is terminal:
                not_terminated $\leftarrow False$
            else:
                S_list.append($S'$)
                $S \leftarrow S'$
        If $t \geq n - 1$:
            n_R_list $\leftarrow$ R_list$[t - n + 1 : t + 1]$
            $G \leftarrow$ (n_R_list $*$ w_list[: len(n_R_list)]).sum()
            If $t + 1 <$ len(S_list):
                $G \leftarrow G +$ w_list$[-1] \times V($S_list$[t + 1])$
            $V($S_list$[t - n + 1]) \leftarrow (1 - \alpha_t)V($S_list$[t - n + 1]) + \alpha_t G$
    while $t - n + 1 <$ len(R_list) $-$ 1
Until a certain criterion is satisfied
```

Table 16.11: n-step TD for estimating $V \sim v_\pi$ (Pythonic style).

when $n = 1$. Therefore one-step TD method is a special case of n-step TD method. The n-step TD prediction is described in Table 16.10, a Pythonian version of which is described in Table 16.11.

As the Monte Carlo error can be express as the sum of discounted TD errors, the n-step TD error can be expressed as the sum of the discounted one-step TD errors. Note that $G_{t:t+n} = R_{t+1} + \gamma G_{t+1:t+n}$. Thus,

$$\begin{aligned}
G_{t:t+n} - V_t(S_t) &= R_{t+1} + \gamma G_{t+1:t+n} - V_t(S_t) \\
&= R_{t+1} + \gamma \left(G_{t+1:t+n} + V_{t+1}(S_{t+1}) - V_{t+1}(S_{t+1}) \right) - V_t(S_t) \\
&= R_{t+1} + \gamma V_{t+1}(S_{t+1}) - V_t(S_t) + \gamma \left(G_{t+1:t+n} - V_{t+1}(S_{t+1}) \right) \\
&= \delta'_t + \gamma \left(G_{t+1:t+n} - V_{t+1}(S_{t+1}) \right) \\
&= \delta'_t + \gamma \delta'_{t+1} + \gamma^2 \left(G_{t+2:t+n} - V_{t+2}(S_{t+2}) \right) \\
&= \delta'_t + \gamma \delta'_{t+1} + \gamma^2 \delta'_{t+2} + \cdots + \gamma^{n-1} \left(G_{t+n-1:t+n} - V_{t+n-1}(S_{t+n-1}) \right) \\
&= \delta'_t + \gamma \delta'_{t+1} + \gamma^2 \delta'_{t+2} + \cdots + \gamma^{n-1} \left(R_{t+n} + \gamma V_{t+n-1}(S_{t+n}) - V_{t+n-1}(S_{t+n-1}) \right) \\
&= \delta'_t + \gamma \delta'_{t+1} + \gamma^2 \delta'_{t+2} + \cdots + \gamma^{n-1} \left(R_{t+n} + \gamma V_{t+n}(S_{t+n}) - V_{t+n-1}(S_{t+n-1}) \right) \\
&\quad + \gamma^n \left(V_{t+n-1}(S_{t+n}) - V_{t+n}(S_{t+n}) \right) \\
&= \delta'_t + \gamma \delta'_{t+1} + \gamma^2 \delta'_{t+2} + \cdots + \gamma^{n-1} \delta'_{t+n-1} + \gamma^n \left(V_{t+n-1}(S_{t+n}) - V_{t+n}(S_{t+n}) \right) \\
&= \sum_{k=t}^{t+n-1} \gamma^{k-t} \delta'_k + \gamma^n \left(V_{t+n-1}(S_{t+n}) - V_{t+n}(S_{t+n}) \right) \\
&= \sum_{k=0}^{n-1} \gamma^k \delta'_{k+t} + \gamma^n \left(V_{t+n-1}(S_{t+n}) - V_{t+n}(S_{t+n}) \right)
\end{aligned}$$

where δ'_t is defined in (16.21). Thus, the n-step TD error is

$$G_{t:t+n} - V_{t+n-1}(S_t) = \sum_{k=0}^{n-1} \gamma^k \delta'_{k+t} + \gamma^n \left(V_{t+n-1}(S_{t+n}) - V_{t+n}(S_{t+n}) \right) + \left(V_t(S_t) - V_{t+n-1}(S_t) \right). \tag{16.30}$$

If V_t does not change during the episode, $\delta_t = \delta'_t$ and $V_{t+n-1}(S_{t+n}) = V_{t+n}(S_{t+n})$, hence

$$G_{t:t+n} - V(S_t) = \sum_{k=t}^{t+n-1} \gamma^{k-t} \delta_k = \sum_{k=0}^{n-1} \gamma^k \delta_{t+k} \tag{16.31}$$

where δ_t is defined in (16.20).

The n-step return uses the value function V_{t+n-1} to correct for the missing rewards beyond R_{t+n}. An important property of n-step returns is that their expectation is guaranteed to be a better estimate of v_π than V_{t+n-1} is, in a worst-state sense. The worst error of the expected n-step return is guaranteed to be less than or equal to γ^n times the worst error under V_{t+n-1}.

$$\max_{s \in \mathcal{S}} \left| \mathbf{E}_\pi \left\{ G_{t:t+n} \middle| S_t = s \right\} - v_\pi(s) \right| \leq \gamma^n \max_s \left| V_{t+n-1}(s) - v_\pi(s) \right| \tag{16.32}$$

for all $n \geq 1$.

This is called the error reduction property of n-step returns. Because of the error reduction property, one can show formally that all n-step TD methods converge to the correct predictions under appropriate technical conditions. The n-step TD methods thus form a family of sound methods, with one-step TD methods and Monte Carlo methods as extreme members.

16.6.2 n-step Sarsa

The n-step Sarsa uses the previous n-step temporal difference idea for control. Like one-step Sarsa, we use a behavior policy based on Q functions that the model learns, but update Q functions using the n-step return based on Q function.

$$G^Q_{t:t+n} = \begin{cases} R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}) & \text{if } t+n < T \\ G_t & \text{if } t+n \geq T \end{cases} \tag{16.33}$$

where $t \geq 0$.

Then, the n-step update using this n-step return is

$$\begin{aligned} Q_{t+n}(S_t, A_t) &= Q_{t+n-1}(S_t, A_t) + \alpha_{t+n-1}\left(G^Q_{t:t+n} - Q_{t+n-1}(S_t, A_t)\right) \\ &= (1 - \alpha_{t+n-1}) Q_{t+n-1}(S_t, A_t) + \alpha_{t+n-1} G^Q_{t:t+n} \end{aligned}$$

The algorithm using this update rule is called *n-step Sarsa*. Table 16.12 describes this algorithm.

As before, the n-step return for Sarsa can be expressed as the sum of TD errors in terms of Q function. Suppose that $t + n < T$. Then

$$\begin{aligned} G^Q_{t:t+n} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}) \\ &= \sum_{k=t}^{t+n-1} \gamma^{k-t} R_{k+1} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}) \\ &= \sum_{k=t}^{t+n-1} \gamma^{k-t} R_{k+1} + \sum_{k=t}^{t+n-1} \gamma^{k-t} \left(Q_k(S_k, A_k) - Q_k(S_k, A_k)\right) + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}) \\ &= \sum_{k=t}^{t+n-1} \gamma^{k-t} R_{k+1} + \sum_{k=t-1}^{t+n-2} \gamma^{k-t+1} Q_{k+1}(S_{k+1}, A_{k+1}) - \sum_{k=t}^{t+n-1} \gamma^{k-t} Q_k(S_k, A_k) \\ &\quad + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}) \\ &= \sum_{k=t}^{t+n-1} \left(\gamma^{k-t} R_{k+1} + \gamma^{k-t+1} Q_{k+1}(S_{k+1}, A_{k+1}) - \gamma^{k-t} Q_k(S_k, A_k)\right) \\ &\quad + Q_t(S_t, A_t) - \gamma^n Q_{t+n}(S_{t+n}, A_{t+n}) + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}) \\ &= \sum_{k=t}^{t+n-1} \gamma^{k-t} \left(R_{k+1} + \gamma Q_{k+1}(S_{k+1}, A_{k+1}) - Q_k(S_k, A_k)\right) \\ &\quad + Q_t(S_t, A_t) + \gamma^n \left(Q_{t+n-1}(S_{t+n}, A_{t+n}) - Q_{t+n}(S_{t+n}, A_{t+n})\right) \end{aligned}$$

```
Algorithm parameters: step size $\alpha_t \in (0,1]$, small $\epsilon > 0$, and $n \in \mathbf{N}$

Initialize:
    $Q(s,a) \in \mathbf{R}$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$ except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:
    Initialize and store $S_0$
    Select an action $A_0$ using policy derived from $Q$ ($e.g.$, $\epsilon$-greedy)
    $T \leftarrow \infty$
    Loop for each $t = 0, 1, 2, \ldots$:
        If $t < T$:
            Take action $A_t$, observe $R_{t+1}$, $S_{t+1}$
            If $S_{t+1}$ is terminal:
                $T \leftarrow t + 1$
            else:
                Select an action $A_{t+1}$ using policy derived from $Q$ ($e.g.$, $\epsilon$-greedy)
        $\tau \leftarrow t - n + 1$
        If $\tau \geq 0$:
            $G \leftarrow \sum_{i=\tau+1}^{\min\{\tau+n,T\}} \gamma^{i-\tau-1} R_i$
            If $\tau + n < T$:
                $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$
            $Q(S_\tau, A_\tau) \leftarrow (1 - \alpha_t) Q(S_\tau, A_\tau) + \alpha_t G$
    while $\tau < T - 1$
Until a certain criterion is satisfied
```

Table 16.12: n-step Sarsa for estimating $Q \sim q_*$ or q_π.

```
Loop:
    Select a state, $S \in \mathcal{S}$, and an action, $A \in \mathcal{A}(S)$, at random
    Send $S$, $A$ to a sample model, and obtain a sample next reward, $R$, and a sample next state, $S'$
    Apply one-step tabular Q-learning to $S$, $A$, $R$, $S'$:
        $Q(S,A) \leftarrow (1-\alpha)Q(S,A) + \alpha\left(R + \gamma \max_{a \in \mathcal{A}(S)} Q(S',a)\right)$
Until a certain criterion is satisfied
```

Table 16.13: Random sample one-step tabular Q-learning.

Thus the n-step error for Sarsa becomes

$$
\begin{aligned}
G^Q_{t:t+n} - Q_{t+n-1}(S_{t+n}, A_{t+n}) &= \sum_{k=t}^{t+n-1} \gamma^{k-t} \left(R_{k+1} + \gamma Q_{k+1}(S_{k+1}, A_{k+1}) - Q_k(S_k, A_k)\right) \\
&\quad + \left(Q_t(S_t, A_t) - Q_{t+n-1}(S_{t+n}, A_{t+n})\right) + \gamma^n \left(Q_{t+n-1}(S_{t+n}, A_{t+n}) - Q_{t+n}(S_{t+n}, A_{t+n})\right)
\end{aligned}
\tag{16.34}
$$

Again, if Q_t does not change during the episode, (16.34) becomes

$$
G^Q_{t:t+n} - Q(S_{t+n}, A_{t+n}) = \sum_{k=t}^{t+n-1} \gamma^{k-t} \left(R_{k+1} + \gamma Q(S_{k+1}, A_{k+1}) - Q(S_k, A_k)\right) \tag{16.35}
$$

16.6.3 n-step off-policy learning

XXX

16.7 Planning and learning with tabular methods

Table 16.13 describes the random sample one-step tabular Q-learning.

16.7.1 Dyna: integrated planning, acting, and learning

The general Dyna architecture is depicted in Figure 16.2. The tabular Dyna-Q algorithm is described in Table 16.14.

16.8 On-policy Prediction with Approximation

XXX

16.9 On-policy Control with Approximation

XXX

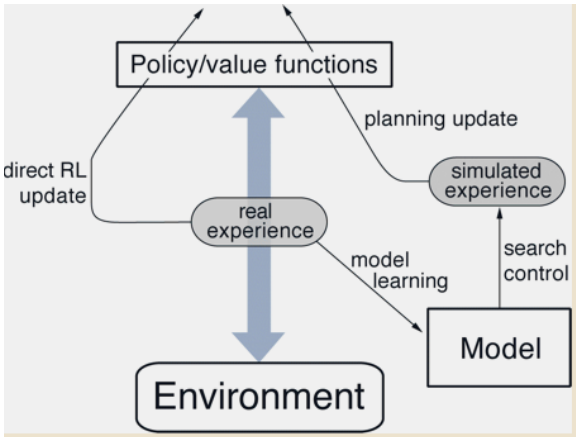

Figure 16.2: The general Dyna Architecture. Real experience, passing back and forth between the environment and the policy, affects policy and value functions in much the same way as does simulated experience generated by the model of the environment.

```
Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(S)$

Loop:
    $S \leftarrow$ current (nonterminal) state
    $A \leftarrow \epsilon$-greedy$(S, Q)$
    Take action $A$; observe reward $R$ and next state $S'$
    Update Q-function: $Q(S,A) \leftarrow (1-\alpha)Q(S,A) + \alpha\left(R + \gamma \max_{a \in \mathcal{A}(S)} Q(S',a)\right)$
    $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
    Loop repeat
    Apply one-step tabular Q-learning to $S$, $A$, $R$, $S'$:
        $S \leftarrow$ random previously observed state
        $A \leftarrow$ random action previously taken in $S$
        $R, S' \leftarrow Model(S,A)$
        Update Q-function: $Q(S,A) \leftarrow (1-\alpha)Q(S,A) + \alpha\left(R + \gamma \max_{a \in \mathcal{A}(S)} Q(S',a)\right)$
Until a certain criterion is satisfied
```

Table 16.14: Tabular Dyna-Q.

```
Algorithm parameters:
    step size strategy $\alpha_t \in (0, 1]$
    epsilon strategy: $\epsilon_t > 0$

Inputs:
    Modeling function: $Q : \mathcal{S} \times \mathcal{A} \times \mathbf{R}^n \to \mathbf{R}$

Initialize:
    Initialize $\theta$ arbitrarily (or from the previous learning for transfer learning)

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
        Take action $A$, observe $R$, $S'$:
            $G \leftarrow R + \gamma \max_{a \in \mathcal{A}(S')} Q(S', A; \theta)$
        Update using gradient descent:
            $\theta \leftarrow \theta + \alpha_t \left(G - Q(S, A; \theta)\right) \nabla_\theta Q(S, A; \theta)$
        $S \leftarrow S'$
    until $S$ is terminal
Until a certain criterion is satisfied
```

Table 16.15: Off-policy Q-learning.

16.10 Off-policy Methods with Approximation

16.11 Eligibility Traces

Eligibility traces are one of the basic mechanisms of reinforcement learning. Almost any temporal-difference (TD) method, such as Q-learning or Sarsa, can be combined with eligibility traces to obtain a more general method that may learn more efficiently.

We have already seen one way of unifying TD and Monte Carlo methods: the n-step TD methods §16.6. What eligibility traces offer beyond these is an elegant algorithmic mechanism with significant computational advantages.

The mechanism is a short-term memory vector, the *eligibility trace* $z_t \in \mathbf{R}^d$ that parallels the long-term weight vector $w_t \in \mathbf{R}^d$. The rough idea is that when a component of w_t participates in producing an estimated value, then the corresponding component of z_t is bumped up and then begins to fade away. Learning will then occur in that component of w_t if a nonzero TD error occurs before the trace falls back to zero. The *trace-decay parameter* $\lambda \in [0, 1]$ determines the rate at which the trace falls.

The primary computational advantage of eligibility traces over n-step methods is that only a single trace vector is required rather than a store of the last n feature vectors. Learning also occurs continually and uniformly in time rather than being delayed and then catching up at the end of the episode. In addition learning can occur and affect behavior immediately after a state is encountered rather than being delayed n steps.

16.11.1 The λ-return

In §16.6 we defined an n-step return as the sum of the first n rewards plus the estimated value of the state reached in n steps, each appropriately discounted. The general form of that equation, for any parameterized function approximator, is

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{v}(S_{t+n}, w_{t+n-1}) \tag{16.36}$$

Now we note that a valid update can be done not just toward any n-step return, but toward any average of n-step returns for different ns. An example of such an average can be the average of 3-step return, 5-step return, and 7-step return, *i.e.*,

$$\frac{1}{3}(G_{t:t+3} + G_{t:t+5} + G_{t:t+7}) \tag{16.37}$$

This is one example, but averaging produces a substantial new range of algorithms. For example, we can average across many n-step returns from 1-step return to ∞-step return to obtain another way of interrelating TD and Monte Carlo methods. Nn principle, one could even average experience-based updates with DP updates to get a simple combination of experience-based and model-based methods.

An update that averages simpler component updates is called a compound update. The TD(λ) algorithm can be understood as one particular way of averaging n-step updates. This average contains all the n-step updates, each weighted proportionally to λ^n (where $\lambda \in [0, 1]$), and is normalized (to ensure that the weights sum to 1). The resulting update is toward a return, called the λ-return , defined in its state-based form by

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n} \tag{16.38}$$

where $(1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} = 1$. If the episode terminates at $t = T$, then $G_{t:t+n} = G_{t:T} = G_t$ for all $n \geq T - t$, thus,

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + (1-\lambda)\sum_{n=T-t}^{\infty} \lambda^{n-1} G_{t:t+n} = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t. \tag{16.39}$$

Note that if $\lambda = 1$, G_t^λ becomes the original return G_t. Thus, updating value functions according to λ-return is equivalent to Monte Carlo algorithm. On the other hand, if $\lambda = 0$, G_t^λ becomes $G_{t:t+1}$, hence updating value functions according to λ-return is equivalent to the one-step temporal difference method, *i.e.*, TD(0) method.

Now we define our first learning algorithm based on the λ-return: the *off-line λ-return algorithm.* As an off-line algorithm, it makes no changes to the weight vector during the episode. Then, at the end of the episode, a whole sequence of off-line updates are made according to the semi-gradient method using the λ-return as the target. The update rule of this method is

$$w_{t+1} = w_t + \alpha_t \left(G_t^\lambda - \hat{v}(S_t, w_t) \right) \nabla_w \hat{v}(S_t, w_t) \tag{16.40}$$

for $t = 0, \ldots, T-1$. The λ-return gives us an alternative way of moving smoothly between Monte Carlo and one-step TD methods that can be compared with the n-step bootstrapping.

The above approach is what can be called the *theoretical or forward* view of a learning algorithm. For each state visited, we look forward in time to all the future rewards and decide how best to combine them.

16.11.2 TD(λ)

TD(λ) is one of the oldest and most widely used algorithms in reinforcement learning. It was the first algorithm for which a formal relationship was shown between a more theoretical forward view and a more computationally congenial backward view using eligibility traces. Here we will show empirically that it approximates the off-line λ-return algorithm presented in the previous section.

TD(λ) improves over the off-line λ-return algorithm in three ways.

- It updates the weight vector on every step of an episode rather than only at the end, thus its estimates is updated sooner (and may be better sooner).
- Its computations are equally distributed in time (rather than all at the end of the episode).
- It can be applied to continuing problems rather than just to episodic problems.

The semi-gradient version of TD(λ) with function approximation will be shown below.

The *eligibility trace* is a vector $z_t \in \mathbf{R}^d$ where the number of components of z_t is the same as that of w_t. Whereas the *weight vector*, w_t, is a long-term memory accumulating over the lifetime of the system, the eligibility trace is a short-term memory, which typically lasts shorter than the length of an episode. Eligibility traces assist in the learning process; their only consequence is that they affect the weight vector, and then the weight vector determines the estimated value. At the same time, the eligibility traces plays a critical role since that is what propagates future reward to the current one, or equivalently, current reward back to the past states.

In TD(λ), the eligibility trace vector is initialized to zero at the beginning of the episode, is incremented on each time step by the value gradient, and then fades away by $\gamma\lambda$ additively, *i.e.*,

$$\begin{aligned} z_{-1} &\leftarrow 0 \\ z_t &\leftarrow \gamma\lambda z_{t-1} + \nabla_w \hat{v}(S_t, w_t),\ 0 \le t \le T \end{aligned} \tag{16.41}$$

where γ is the *discount rate* and λ is the *trace-decay parameter*.

The eligibility trace keeps track of which components of the weight vector have contributed, positively or negatively, to recent state valuations, where "recent" is defined in terms of $\gamma\lambda$. The trace indicates the eligibility of each component of the weight vector for undergoing learning changes should a reinforcing event occur where the reinforcing events are the moment-by-moment one-step TD errors. The TD error for state-value prediction is

$$\delta_t \leftarrow R_{t+1} + \gamma\hat{v}(S_{t+1}, w_t) - \hat{v}(S_t, w_t) \tag{16.42}$$

Finally, the weight vector is updated on each step proportionally to all three factors; the step size, the TD error, and the eligibility trace, *i.e.*,

$$w_{t+1} \leftarrow w_t + \alpha_t \delta_t z_t. \tag{16.43}$$

Table 16.16 described the *semi-gradient TD(λ) algorithm* for estimating $\hat{v} \sim v_\pi$ using (16.41), (16.42), and (16.43).

16.11.3 Why TD(λ) approximates the off-line λ-return algorithm?

Here we examine why TD(λ) approximates the off-line λ-return algorithm.

In this section, we drop the subscript t from w_t. Because of this, we can only prove thet TD(λ) is *approximately* the same as the off-line λ-return algorithm. In fact, they are different.

```
Inputs:
    $\pi$ to be evaluated
    differential function $\hat{v} : \mathcal{S}' \times \mathbf{R}^d \to \mathbf{R}$ such that $\hat{v}(\text{terminal}, \cdot) = 0$

Algorithm parameters:
    trace-decay parameter, $\lambda \in [0, 1]$, and step size, $\alpha_t > 0$

Initialize:
    Initialize value-function weights $w$ arbitrarily

Loop for each episode:
    Initialize $S$
    $z \leftarrow 0$
    Loop for each step of episode:
        Choose $A \sim \pi(\cdot|S)$
        Take action $A$, observe $R$, $S'$
        $z \leftarrow \gamma\lambda z + \nabla_w \hat{v}(S, w)$
        $\delta \leftarrow R + \gamma\hat{v}(S', w) - \hat{v}(S, w)$
        $w \leftarrow w + \alpha_t \delta z$
        $S \leftarrow S'$
    until $S'$ is terminal
```

Table 16.16: Semi-gradient TD(λ) algorithm for estimating $\hat{v} \sim v_\pi$.

First, the n-step return at t can be rewritten in terms of that at $t+1$ as follows.

$$\begin{aligned}
G_{t:t+n} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{v}(S_{t+n}, w) \\
&= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+n} \\
&\quad + \gamma \hat{v}(S_{t+1}, w) - \hat{v}(S_t, w) \\
&\quad + \gamma(\gamma \hat{v}(S_{t+2}, w) - \hat{v}(S_{t+1}, w)) \\
&\quad + \cdots \\
&\quad + \gamma^{n-1}(\gamma \hat{v}(S_{t+n}, w) - \hat{v}(S_{t+n-1}, w)) \\
&\quad + \hat{v}(S_t, w) \\
&= R_{t+1} + \gamma \hat{v}(S_{t+1}, w) - \hat{v}(S_t, w) \\
&\quad + \gamma(R_{t+2} + \gamma \hat{v}(S_{t+2}, w) - \hat{v}(S_{t+1}, w)) \\
&\quad + \cdots \\
&\quad + \gamma^{n-1}(R_{t+n} + \gamma \hat{v}(S_{t+n}, w) - \hat{v}(S_{t+n-1}, w)) \\
&\quad + \hat{v}(S_t, w),
\end{aligned}$$

thus, (16.42) implies

$$G_{t:t+n} = \delta_t + \gamma \delta_{t+1} + \cdots + \gamma^{n-1} \delta_{t+n-1} + \hat{v}(S_t, w) \tag{16.44}$$

Now (16.44) together with (16.38) implies that

$$\begin{aligned}
G_t^\lambda &= (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n} \\
&= (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \left(\delta_t + \gamma \delta_{t+1} + \cdots + \gamma^{n-1} \delta_{t+n-1} + \hat{v}(S_t, w) \right) \\
&= (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \left(\sum_{k=0}^{n-1} \gamma^k \delta_{t+k} + \hat{v}(S_t, w) \right) \\
&= (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \left(\sum_{k=0}^{n-1} \gamma^k \delta_{t+k} \right) + (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \hat{v}(S_t, w) \\
&= (1-\lambda) \sum_{n=1}^{\infty} \sum_{k=0}^{n-1} \lambda^{n-1} \gamma^k \delta_{t+k} + (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \hat{v}(S_t, w) \\
&= (1-\lambda) \sum_{k=0}^{\infty} \sum_{n=k+1}^{\infty} \lambda^{n-1} \gamma^k \delta_{t+k} + (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \hat{v}(S_t, w) \\
&= (1-\lambda) \sum_{k=0}^{\infty} \gamma^k \delta_{t+k} \sum_{n=k+1}^{\infty} \lambda^{n-1} + \hat{v}(S_t, w) \\
&= \sum_{k=0}^{\infty} (\gamma\lambda)^k \delta_{t+k} + \hat{v}(S_t, w)
\end{aligned}$$

since $(1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}=1$, thus

$$G_t^{\lambda}-\hat{v}(S_t,w)=\sum_{k=0}^{\infty}(\gamma\lambda)^k\delta_{t+k}. \tag{16.45}$$

If an episode ends at $t=T$, then $tderrort=0$ for $t\geq T$, hence

$$G_t^{\lambda}-\hat{v}(S_t,w)=\sum_{k=0}^{T-t-1}(\gamma\lambda)^k\delta_{t+k}. \tag{16.46}$$

Now we derived a formula for z_{t+n} using summation notation from the recursive formula (16.41).

$$\begin{aligned}
z_{t+n} &= \gamma\lambda z_{t+n-1}+\nabla_w\hat{v}(S_{t+n},w)\\
&= \gamma\lambda\left(\gamma\lambda z_{t+n-2}+\nabla_w\hat{v}(S_{t+n-1},w)\right)+\nabla_w\hat{v}(S_{t+n},w)\\
&= (\gamma\lambda)^2 z_{t+n-2}+(\gamma\lambda)\nabla_w\hat{v}(S_{t+n-1},w)+\nabla_w\hat{v}(S_{t+n},w)\\
&\vdots\\
&= (\gamma\lambda)^n\nabla_w\hat{v}(S_t,w)+(\gamma\lambda)^{n-1}\nabla_w\hat{v}(S_{t+1},w)+\cdots+(\gamma\lambda)\nabla_w\hat{v}(S_{t+n-1},w)+\nabla_w\hat{v}(S_{t+n},w),
\end{aligned}$$

thus,

$$z_{t+n}=\sum_{k=0}^{n}(\gamma\lambda)^{n-k}\nabla_w\hat{v}(S_{t+k},w). \tag{16.47}$$

(We can prove (16.47) mathematically strictly using the mathematical induction.)

We also derive a formula for w_{t+n} using summation notation from the recursive formula (16.43) and (16.42).

$$\begin{aligned}
w_{t+n} &= \alpha\delta_{t+n-1}z_{t+n-1}+w_{t+n-1}\\
&= \alpha\left(\delta_{t+n-1}z_{t+n-1}+\delta_{t+n-2}z_{t+n-2}\right)+w_{t+n-2}\\
&= \alpha\left(\delta_{t+n-1}z_{t+n-1}+\delta_{t+n-2}z_{t+n-2}+\cdots+\delta_{t+1}z_{t+1}+\delta_t z_t\right)+w_t
\end{aligned}$$

where we drop the subscript from the step size α_t. Thus,

$$w_{t+n}=\alpha\sum_{j=0}^{n-1}\delta_{t+j}z_{t+j}+w_t. \tag{16.48}$$

Now combining (16.47) and (16.48) yields

$$\begin{aligned}
w_{t+n} - w_t &= \alpha \sum_{j=0}^{n-1} \delta_{t+j} z_{t+j} \\
&= \alpha \sum_{j=0}^{n-1} \delta_{t+j} \sum_{k=0}^{j} (\gamma\lambda)^{j-k} \nabla_w \hat{v}(S_{t+k}, w). \\
&= \alpha \sum_{j=0}^{n-1} \sum_{k=0}^{j} (\gamma\lambda)^{j-k} \delta_{t+j} \nabla_w \hat{v}(S_{t+k}, w). \\
&= \alpha \sum_{k=0}^{n-1} \sum_{j=k}^{n-1} (\gamma\lambda)^{j-k} \delta_{t+j} \nabla_w \hat{v}(S_{t+k}, w) \\
&= \alpha \sum_{k=0}^{n-1} \nabla_w \hat{v}(S_{t+k}, w) \sum_{j=k}^{n-1} (\gamma\lambda)^{j-k} \delta_{t+j} \\
&= \alpha \sum_{k=0}^{n-1} \nabla_w \hat{v}(S_{t+k}, w) \sum_{j=0}^{n-k-1} (\gamma\lambda)^{j} \delta_{t+k+j} \\
&= \alpha \sum_{k=0}^{n-1} \nabla_w \hat{v}(S_{t+k}, w) \sum_{j=0}^{n+t-(t+k)-1} (\gamma\lambda)^{j} \delta_{(t+k)+j}. \qquad (16.49)
\end{aligned}$$

Now if we substitute $T - t$ for n in (16.49), (16.46) implies

$$\begin{aligned}
w_T - w_t &= \alpha \sum_{k=0}^{T-t-1} \nabla_w \hat{v}(S_{t+k}, w) \sum_{j=0}^{T-(t+k)-1} (\gamma\lambda)^{j} \delta_{(t+k)+j} \\
&= \alpha \sum_{k=0}^{T-t-1} \nabla_w \hat{v}(S_{t+k}, w) \left(G_{t+k}^{\lambda} - \hat{v}(S_{t+k}, w)\right) \\
&= \alpha \sum_{k=0}^{T-t-1} \left(G_{t+k}^{\lambda} - \hat{v}(S_{t+k}, w)\right) \nabla_w \hat{v}(S_{t+k}, w). \\
&= \alpha \sum_{k=t}^{T-1} \left(G_{k}^{\lambda} - \hat{v}(S_{k}, w)\right) \nabla_w \hat{v}(S_{k}, w). \qquad (16.50)
\end{aligned}$$

Now observer that the semi-gradient update rule for off-line λ-return algorithm (16.40) implies that

$$\begin{aligned}
w_T &= \alpha\left(G_{T-1}^{\lambda} - \hat{v}(S_{T-1},w)\right)\nabla_w \hat{v}(S_{T-1},w) + w_{T-1} \\
&= \alpha\left(G_{T-1}^{\lambda} - \hat{v}(S_{T-1},w)\right)\nabla_w \hat{v}(S_{T-1},w) + \alpha\left(G_{T-2}^{\lambda} - \hat{v}(S_{T-2},w)\right)\nabla_w \hat{v}(S_{T-2},w) + w_{T-2} \\
&\vdots \\
&= \alpha\left(G_{T-1}^{\lambda} - \hat{v}(S_{T-1},w)\right)\nabla_w \hat{v}(S_{T-1},w) + \cdots + \alpha\left(G_{t}^{\lambda} - \hat{v}(S_{t},w)\right)\nabla_w \hat{v}(S_{t},w) + w_t \\
&= \alpha\sum_{k=t}^{T-1}\left(G_{k}^{\lambda} - \hat{v}(S_{k},w)\right)\nabla_w \hat{v}(S_{k},w) + w_t \qquad (16.51)
\end{aligned}$$

assuming that $\alpha_t = \alpha$ and $w_t = w$ for all t. (We can prove (16.51) in a mathematically strict way using the mathematical induction.) Comparing (16.50) with (16.51) implies that the off-line λ-return algorithm and TD(λ) are equivalent when we drop t from w_t and α_t. Therefore we have shown that TD(λ) approximates the off-line λ-return algorithm.

16.11.4 Sarsa(λ)

16.11.5 Tabular methods using eligibility traces

In the previous sections, we describe TD(λ) for function approximation cases. Since the tabular case is a special case of the function approximation, we can readily derive TD(λ) algorithm for the tabular case. Here we show the process of deriving TD(λ) algorithm for the tabular case and the final algorithm description.

Suppose that the set of all the states, $\mathcal{S}$, is finite and there is a one-to-one mapping $\phi : \{1,\ldots,|\mathcal{S}|\} \to \mathcal{S}$ which maps each integer index to a state. Let $w \in \mathbf{R}^{|\mathcal{S}|}$ be the weight vector such that $w_i = V(\phi(i))$, *i.e.*, w_i represents the value of the state value function for the ith state, *i.e.*, the state S such with $\phi(i) = S \Leftrightarrow \phi^{-1}(S) = i$. Then we have

$$\hat{v}(S,w) = w_{\phi^{-1}(S)}. \qquad (16.52)$$

Now we apply (16.52) to each of the three update rule for TD(λ); (16.41), (16.42), and (16.43). We first derived the gradient of $\hat{v}(S,w)$ with respect to w for each $S \in \mathcal{S}$. Note that

$$\frac{\partial}{\partial w_i}\hat{v}(S,w) = \delta_{\phi^{-1}(S),i} = \begin{cases} 1 & \text{if } \phi(i) = S \\ 0 & \text{otherwise} \end{cases} \qquad (16.53)$$

where $\delta_{\cdot,\cdot}$ is the Kronecker delta function defined by

$$\delta_{i,j} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases} \qquad (16.54)$$

Therefore the gradient is

$$\nabla_w \hat{v}(S,w) = e_{\phi^{-1}(S)} \qquad (16.55)$$

where $e_i \in \mathbf{R}^{|\mathcal{S}|}$ is the ith unit vector whose entries are all zero except the ith entry which is 1.

```
Inputs:
    $\pi$ to be evaluated

Algorithm parameters:
    trace-decay parameter, $\lambda \in [0, 1]$, and step size, $\alpha_t > 0$

Initialize:
    Initialize $V(S)$ for all $S \in \mathcal{S}$

Loop for each episode:
    Initialize $S$
    $z(S) \leftarrow 0$ for all $S \in \mathcal{S}$
    Loop for each step of episode:
        Choose $A \sim \pi(\cdot|S)$
        Take action $A$, observe $R$, $S'$
        $\delta \leftarrow R + \gamma V(S') - V(S)$
        $z(S) \leftarrow z(S) + 1$
        For all $\tilde{S} \in \mathcal{S}$:
            $V(\tilde{S}) \leftarrow V(\tilde{S}) + \alpha_t \delta z(\tilde{S})$
            $z(\tilde{S}) \leftarrow \gamma\lambda z(\tilde{S})$
        $S \leftarrow S'$
    until $S'$ is terminal
```

Table 16.17: Tabular TD(λ) algorithm for estimating $\hat{v} \sim v_\pi$.

Now let us apply (16.55) to (16.41), (16.42), and (16.43). First, the eligibility trace update (16.41) becomes

$$\begin{aligned} z_{-1} &\leftarrow 0 \in \mathbf{R}^{|\mathcal{S}|} \\ z_t &\leftarrow \gamma\lambda z_{t-1} + e_{\phi^{-1}(S_t)} \; 0 \le t \le T \end{aligned} \tag{16.56}$$

and (16.42) becomes

$$\delta_t \leftarrow R_{t+1} + \gamma w_{\phi^{-1}(S_{t+1})} - w_{\phi^{-1}(S_t)}. \tag{16.57}$$

However, (16.43) remains the same.

$$w_{t+1} \leftarrow w_t + \alpha_t \delta_t z_t. \tag{16.58}$$

Now we note that we would better express this in terms of $V_t : \mathcal{S} \to \mathbf{R}$ and $z_t : \mathcal{S} \to \mathbf{R}$. Thus, if we plug (16.56), (16.57), and (16.58) in Table 16.16 using V_t and z_t, we have Table 16.17.

```
Algorithm parameters:
    trace-decay parameter, $\lambda \in [0, 1]$, and step size, $\alpha_t > 0$

Initialize:
    Initialize $Q(S, A)$ for all $S \in \mathcal{S}$, $A \in \mathcal{A}$

Loop for each episode:
    $z(S, A) \leftarrow 0$ for all $S \in \mathcal{S}$, $A \in \mathcal{A}$
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
    Loop for each step of episode:
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
        $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$
        $z(S, A) \leftarrow z(S, A) + 1$
        For all $\tilde{S} \in \mathcal{S}$ and $\tilde{A} \in \mathcal{A}$:
            $Q(\tilde{S}, \tilde{A}) \leftarrow Q(\tilde{S}, \tilde{A}) + \alpha_t \delta z(\tilde{S}, \tilde{A})$
            $z(\tilde{S}, \tilde{A}) \leftarrow \gamma \lambda z(\tilde{S}, \tilde{A})$
        $S \leftarrow S'$, $A \leftarrow A'$
    until $S'$ is terminal
```

Table 16.18: Tabular SARSA(λ) algorithm for estimating $Q \sim q_*$.

16.12 Appendix: conditional probability and expected value

Suppose that we have a sequence of random variables, X, Y, and Z with supports $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$.

Note that the definition of the conditional probability implies that for all $x \in \mathcal{X}$, $y \in \mathcal{Y}$, $z \in \mathcal{Z}$ such that $p_Y(y) \neq 0$ and $p_Z(z) \neq 0$,

$$p_{X|Y}(x|y) = \frac{p_{X,Y}(x, y)}{p_Y(y)} \Leftrightarrow p_{X,Y}(x, y) = p_{X|Y}(x|y) p_Y(y). \tag{16.59}$$

$$p(x, y|z) = \frac{p(x, y, z)}{p(z)} = \frac{p(x, y, z)}{p(y, z)} \frac{p(y, z)}{p(z)} = p(x|y, z) p(y|z). \tag{16.60}$$

```
Algorithm parameters:
    trace-decay parameter, $\lambda \in [0,1]$, and step size, $\alpha_t > 0$

Initialize:
    Initialize $Q(S,A)$ for all $S \in \mathcal{S}$, $A \in \mathcal{A}$

Loop for each episode:
    $z(S,A) \leftarrow 0$ for all $S \in \mathcal{S}$, $A \in \mathcal{A}$
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
    Loop for each step of episode:
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
        $A^* \leftarrow \text{argmin}_{\tilde{A}}\, Q(S', \tilde{A})$
        $\delta \leftarrow R + \gamma Q(S', A^*) - Q(S,A)$
        $z(S,A) \leftarrow z(S,A) + 1$
        For all $\tilde{S} \in \mathcal{S}$ and $\tilde{A} \in \mathcal{A}$:
            $Q(\tilde{S}, \tilde{A}) \leftarrow Q(\tilde{S}, \tilde{A}) + \alpha_t \delta z(\tilde{S}, \tilde{A})$
            If $\tilde{A} = A^*$:
                $z(\tilde{S}, \tilde{A}) \leftarrow \gamma \lambda z(\tilde{S}, \tilde{A})$
            else:
                $z(\tilde{S}, \tilde{A}) \leftarrow 0$
        $S \leftarrow S'$, $A \leftarrow A'$
    until $S'$ is terminal
```

Table 16.19: Tabular version of Watkins's $Q(\lambda)$ algorithm for estimating $Q \sim q_*$.

Then (16.59) implies

$$\begin{aligned}
\mathbf{E}(X) &= \int_{\mathcal{X}} x p_X(x) dx \\
&= \int_{\mathcal{X}} x \left(\int_{\mathcal{Y}} p_{X,Y}(x,y) dy \right) dx = \int_{\mathcal{X}} x \left(\int_{\mathcal{Y}} p_{X|Y}(x|y) p_Y(y) dy \right) dx \\
&= \int_{\mathcal{Y}} \int_{\mathcal{X}} x p_{X|Y}(x|y) p_Y(y) dx dy = \int_{\mathcal{Y}} \left(\int_{\mathcal{X}} x p_{X|Y}(x|y) dx \right) p_Y(y) dy \\
&= \int_{\mathcal{Y}} \mathbf{E}(X|Y=y) p_Y(y) dy = \mathbf{E}_Y \mathbf{E}_{X|Y}(X|Y),
\end{aligned}$$

$$\mathbf{E}(X) = \mathbf{E}_{X|Y}(X|Y), \tag{16.61}$$

and (16.60) implies

$$\begin{aligned}
\mathbf{E}(X|Z=z) &= \int_{\mathcal{X}} x p_{X|Z}(x|z) dx \\
&= \int_{\mathcal{X}} x \left(\int_{\mathcal{Y}} p_{X,Y|Z}(x,y|z) dy \right) dx = \int_{\mathcal{X}} x \left(\int_{\mathcal{Y}} p_{X|Y,Z}(x|y,z) p_{Y|Z}(y|z) dy \right) dx \\
&= \int_{\mathcal{Y}} \int_{\mathcal{X}} x p_{X|Y,Z}(x|y,z) p_{Y|Z}(y|z) dx dy = \int_{\mathcal{Y}} \left(\int_{\mathcal{X}} x p_{X|Y,Z}(x|y,z) dx \right) p_{Y|Z}(y|z) dy \\
&= \int_{\mathcal{Y}} \mathbf{E}(X|Y=y, Z=z) p_{Y|Z}(y|z) dy = \mathbf{E}_{Y|Z=z} \mathbf{E}(X|Y, Z=z),
\end{aligned}$$

i.e.,

$$\mathbf{E}(X|Z=z) = \mathbf{E}_{Y|Z=z} \mathbf{E}(X|Y, Z=z). \tag{16.62}$$