UMA ABORDAGEM OBJETIVA

FUNDAMENTOS MATEMÁTICOS PARA MODELOS DE LINGUAGEM

GISELDO NEO

ALANA NEO

Índice

Capítulo 1

Fundamentos Matemáticos para LLMs

Bem Vindos! Entusiastas dos LLMs.

Baixar PDF

Este livro estabelece as fundações matemáticas necessárias para trabalhar com *Large Language Models* (LLMs).

Grande parte deste livro foi originado a partir das anotações do Professor Enno Nagel, a quem merece todos os créditos. https://konfekt.bitbucket.io/courses/aprendizado/aprendizado.html

Capítulo 2

Conceitos básicos

2.1 Conjuntos

Conjuntos são uma linguagem para organizar objetos e raciocinar sobre coleções. Em Ciência da Computação, eles aparecem ao descrever usuários de um sistema, estados possíveis de um programa, símbolos de um alfabeto, resultados de uma consulta e exemplos de um conjunto de dados. Entender conjuntos ajuda a transformar frases como "selecione os registros válidos" ou "encontre elementos presentes nas duas listas" em operações matemáticas precisas.

2.1.1 Elementos, pertinência e representação

Um **conjunto** é uma coleção bem definida de objetos, chamados **elementos**. Escrevemos seus elementos entre chaves. Por exemplo,

$$A = \{2, 4, 6, 8\}.$$

A expressão $4 \in A$ informa que 4 pertence a A; já $5 \notin A$ informa que 5 não pertence a A. A ordem e a repetição não alteram um conjunto. Portanto,

$$\{1, 2, 3\} = \{3, 2, 1\} = \{1, 1, 2, 3\}.$$

Essa propriedade distingue conjuntos de estruturas como listas e vetores. A lista `[1, 1, 2]` possui três posições e preserva a ordem; o conjunto correspondente possui apenas os elementos $\{1, 2\}$. Linguagens como Python refletem essa diferença: `list` mantém ordem e repetições, enquanto `set` elimina duplicatas.

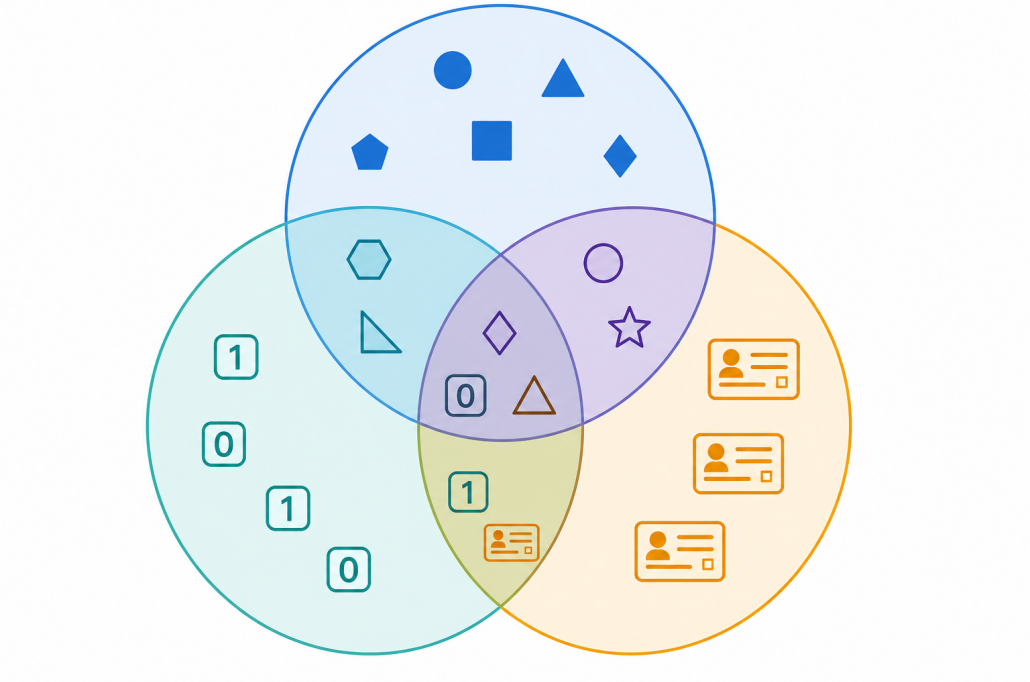


Figura 2.1: Coleções de objetos e regiões sobrepostas ilustram a ideia de conjuntos e elementos compartilhados.

Um conjunto também pode ser descrito por uma propriedade. Em vez de listar todos os valores pares, escrevemos

$$P = \{x \in \mathbb{Z} \mid x \text{ é par}\}.$$

O símbolo "|" significa "tal que". Essa notação é útil quando o conjunto é muito grande ou infinito. O **conjunto vazio**, denotado por $\emptyset$ ou $\{\}$, não possui elemento algum. Atenção: $\emptyset$ é diferente de $\{\emptyset\}$, pois o segundo conjunto tem um elemento — o próprio conjunto vazio.

2.1.2 Conjuntos numéricos

Alguns conjuntos aparecem repetidamente na matemática aplicada à Computação:

- $\mathbb{N} = \{0, 1, 2, 3, ...\}$: números naturais, usados em contagens e índices;
- $\mathbb{Z} = \{..., -2, -1, 0, 1, 2, ...\}$: números inteiros;
- $\mathbb{Q}$: números racionais, que podem ser escritos como p/q, com $p, q \in \mathbb{Z}$ e $q \neq 0$;
- $\mathbb{R}$: números reais, comuns em medições, probabilidades e parâmetros de modelos.

Existe uma cadeia de inclusões:

$$\mathbb{N} \subseteq \mathbb{Z} \subseteq \mathbb{Q} \subseteq \mathbb{R}.$$

Isso significa, por exemplo, que todo número natural também é inteiro e real. Os números irracionais, como $\sqrt{2}$ e π, pertencem a $\mathbb{R} \setminus \mathbb{Q}$.

2.1.3 Subconjuntos e igualdade

Dizemos que A é **subconjunto** de B, e escrevemos $A \subseteq B$, quando todo elemento de A também pertence a B. Se $A \subseteq B$ e $A \neq B$, então A é um subconjunto próprio de B, indicado por $A \subset B$.

Dois conjuntos são iguais quando possuem exatamente os mesmos elementos. Uma maneira comum de provar $A = B$ é demonstrar as duas inclusões: $A \subseteq B$ e $B \subseteq A$. Essa estratégia, chamada **dupla inclusão**, é análoga a verificar duas condições lógicas em vez de comparar apenas a aparência das representações.

2.1.4 Operações entre conjuntos

Considere $A = \{1, 2, 3, 4\}$ e $B = \{3, 4, 5\}$. As principais operações são:

- **união:** $A \cup B = \{1, 2, 3, 4, 5\}$ reúne elementos presentes em A **ou** em B;
- **interseção:** $A \cap B = \{3, 4\}$ mantém elementos presentes em A **e** em B;
- **diferença:** $A \setminus B = \{1, 2\}$ mantém elementos de A que não pertencem a B;
- **diferença simétrica:** $A \triangle B = \{1, 2, 5\}$ mantém elementos que pertencem a apenas um dos conjuntos.

Se $A \cap B = \emptyset$, os conjuntos são chamados **disjuntos**. Em um universo U, o **complemento** de A é $A^c = U \setminus A$: tudo que está no universo considerado, mas não em A. O universo precisa ser declarado, pois o complemento muda conforme o domínio do problema.

Essas operações correspondem diretamente a tarefas computacionais: remover duplicatas, filtrar dados, combinar resultados e localizar itens em comum. Em bancos de dados, `UNION`, `INTERSECT` e `EXCEPT` expressam ideias equivalentes. Em Python, para conjuntos `A` e `B`, podemos usar `A | B`, `A & B` e `A - B`.

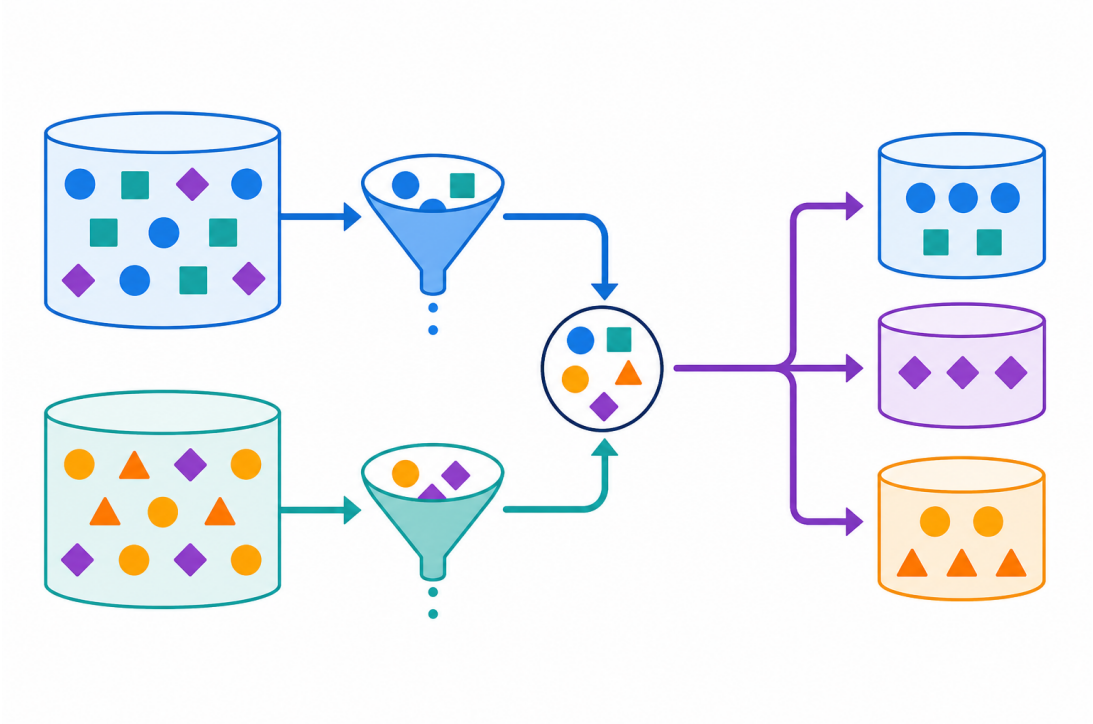


Figura 2.2: Fluxo visual de filtragem, combinação e separação de coleções de dados.

As leis de De Morgan relacionam operações e complementos:

$$(A \cup B)^c = A^c \cap B^c \qquad \text{e} \qquad (A \cap B)^c = A^c \cup B^c.$$

Elas aparecem na simplificação de condições booleanas. Negar "x está em A ou em B" equivale a afirmar que "x não está em A e não está em B".

2.1.5 Cardinalidade e conjunto das partes

A **cardinalidade** $|A|$ é o número de elementos de um conjunto finito. Se $A = \{a, b, c\}$, então $|A| = 3$. Para conjuntos finitos,

$$|A \cup B| = |A| + |B| - |A \cap B|.$$

Subtraímos a interseção porque seus elementos foram contados duas vezes. Essa fórmula é o caso mais simples do princípio da inclusão-exclusão, usado em contagem e análise combinatória.

O **conjunto das partes** $\mathcal{P}(A)$ contém todos os subconjuntos de A. Para $A = \{0, 1\}$,

$$\mathcal{P}(A) = \{\emptyset, \{0\}, \{1\}, \{0, 1\}\}.$$

Se $|A| = n$, então $|\mathcal{P}(A)| = 2^n$. Cada elemento pode ser incluído ou não em um subconjunto, exatamente como um bit pode assumir 1 ou 0. Por isso, subconjuntos podem ser representados por máscaras de bits: para $A = \{a, b, c\}$, a máscara `101` representa $\{a, c\}$.

2.1.6 Produto cartesiano

O **produto cartesiano** de A e B é o conjunto de todos os pares ordenados possíveis:

$$A \times B = \{(a, b) \mid a \in A \text{ e } b \in B\}.$$

Se $A = \{1, 2\}$ e $B = \{x, y\}$, então

$$A \times B = \{(1, x), (1, y), (2, x), (2, y)\}.$$

A ordem importa: em geral, $(a, b) \neq (b, a)$. Se A e B são finitos, $|A \times B| = |A|\,|B|$. Em Computação, produtos cartesianos modelam coordenadas, combinações de parâmetros, estados compostos e linhas produzidas por um `CROSS JOIN`. Quando $B = A$, usamos A^2; assim, $\mathbb{R}^2$ representa o plano e $\mathbb{R}^3$ representa o espaço tridimensional.

> Verifique sua compreensão
>
> Considere $U = \{1, 2, 3, 4, 5, 6\}$, $A = \{1, 2, 3, 4\}$ e $B = \{2, 4, 6\}$. Calcule $A \cap B$, $A \cup B$, $A \setminus B$ e B^c em relação a U. Depois, represente $A \cap B$ como um conjunto em Python e teste se `3` pertence ao resultado.

2.2 Funções

Funções são uma das linguagens fundamentais da Matemática e da Ciência da Computação. Elas aparecem quando um programa transforma uma entrada em uma saída, quando um banco de dados associa uma chave a um registro, quando um classificador atribui um rótulo a uma imagem ou quando uma rede neural converte um vetor de atributos em probabilidades. Em todos esses casos existe a mesma ideia: para cada entrada admissível, há uma saída bem determinada.

2.2.1 Definição e notação

Sejam X e Y dois conjuntos. Uma **função** f de X em Y é uma regra que associa a cada elemento $x \in X$ exatamente um elemento $y \in Y$. Escrevemos

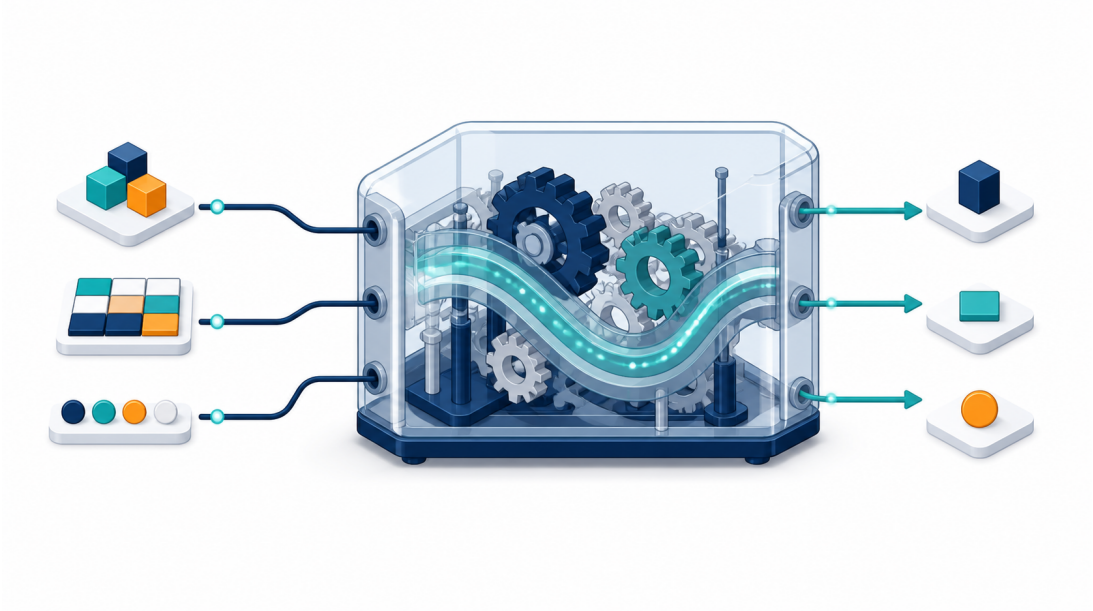


Figura 2.3: Uma função vista como uma máquina determinística: entradas admissíveis são processadas e cada uma produz uma única saída.

$$f\colon X \to Y, \qquad x \mapsto f(x).$$

Nessa notação:

- X é o **domínio**, isto é, o conjunto das entradas permitidas;
- Y é o **contradomínio**, o conjunto no qual declaramos que as saídas estão;
- x é um **argumento** ou uma **entrada**;
- $f(x)$ é o **valor da função** ou a **saída** correspondente a x.

A palavra "exatamente" é essencial. Uma entrada do domínio não pode ficar sem saída e também não pode possuir duas saídas diferentes. Nada impede, entretanto, que duas entradas distintas produzam a mesma saída. Por exemplo, para $f(x) = x^2$, temos $f(2) = f(-2) = 4$.

Na programação, a assinatura de uma função expressa ideia semelhante à notação matemática. A declaração conceitual

```
quadrado : Real -> Real
```

corresponde a $q\colon \mathbb{R} \to \mathbb{R}$, $q(x) = x^2$. O tipo de entrada descreve o domínio esperado, enquanto o tipo de retorno descreve o contradomínio. A analogia não é perfeita: uma função matemática não altera estado, não realiza entrada e saída e sempre associa o mesmo valor à mesma entrada. Em computação, funções com essas propriedades são frequentemente chamadas de **funções puras**.

> **!** Função não é apenas fórmula
>
> Uma função pode ser descrita por uma fórmula, uma tabela, um algoritmo ou uma regra verbal. O que a caracteriza é a associação inequívoca entre entradas e saídas, e não a forma usada para apresentá-la.

2.2.2 Domínio, contradomínio e imagem

A **imagem** de f é o conjunto das saídas que a função realmente produz:

$$\mathrm{im}(f) = \{f(x) \mid x \in X\}.$$

Sempre vale $\mathrm{im}(f) \subseteq Y$, mas a inclusão pode ser estrita. Considere

$$q\colon \mathbb{R} \to \mathbb{R}, \qquad q(x) = x^2.$$

O contradomínio declarado é $\mathbb{R}$, porém nenhum número negativo é produzido. Logo,

$$\mathrm{im}(q) = [0, \infty).$$

Essa distinção importa em computação. Um tipo de retorno pode permitir muitos valores, embora uma implementação produza apenas parte deles. Também importa para decidir se uma função possui inversa.

Quando uma fórmula é apresentada sem domínio explícito, costuma-se adotar o maior subconjunto dos reais em que a expressão faz sentido. Assim,

$$r(x) = \frac{1}{x(x-3)}$$

tem domínio $\mathbb{R} \setminus \{0, 3\}$, pois divisão por zero não está definida. Já $s(x) = \sqrt{x}$, como função real, tem domínio $[0, \infty)$. Em um programa, tentar avaliar essas expressões fora do domínio pode provocar uma exceção, um valor especial como `NaN` ou um resultado pertencente a outro sistema numérico, como os complexos.

2.2.3 Diferentes maneiras de representar uma função

Uma mesma função pode ser apresentada de várias formas, cada uma conveniente para um tipo de problema.

2.2.3.1 Fórmula

Uma fórmula descreve diretamente como calcular a saída. Na queda livre ideal, partindo do repouso, a distância percorrida após t segundos é

$$s(t) = \frac{1}{2} g t^2, \qquad t \geq 0,$$

em que $g \approx 9{,}8\,\mathrm{m/s^2}$. Por exemplo, $s(2) = 19{,}6\,\mathrm{m}$. A fórmula deixa evidente a dependência entre tempo e distância.

2.2.3.2 Tabela ou dicionário

Uma função sobre um conjunto finito pode ser armazenada como tabela. Um sistema pode associar extensões de arquivos aos respectivos tipos:

Entrada	Saída
`.png`	imagem
`.qmd`	documento Quarto
`.py`	código Python

Essa representação é semelhante a um dicionário ou mapa em uma linguagem de programação: uma chave válida determina um único valor.

2.2.3.3 Algoritmo

Algumas funções são descritas mais naturalmente por uma sequência de instruções. A função que informa se um inteiro é par pode ser definida por

```
def eh_par(n: int) -> bool:
    return n % 2 == 0
```

Matematicamente, ela tem a forma

$$\text{par} \colon \mathbb{Z} \to \{\text{falso}, \text{verdadeiro}\}.$$

O código é uma implementação da regra matemática. Diferentes algoritmos podem implementar a mesma função, com custos diferentes de tempo e memória.

2.2.3.4 Gráfico

Para funções reais de uma variável real, o gráfico é o conjunto

$$G_f = \{(x, f(x)) \mid x \in X\}.$$

Cada entrada x determina um ponto de altura $f(x)$. O **teste da reta vertical** fornece uma verificação visual: se alguma reta vertical intercepta a curva em mais de um ponto, a relação desenhada não representa y como função de x.

Um gráfico revela tendências, máximos, mínimos e descontinuidades, mas uma figura amostrada não substitui a definição exata. Em computação gráfica, unir pontos calculados pode esconder oscilações ou descontinuidades existentes entre as amostras.

2.2.4 Funções de várias variáveis

Entradas computacionais frequentemente são vetores, e não números isolados. Uma função que estima a nota final usando prova e projeto pode ser escrita como

$$n\colon [0,10]^2 \to [0,10], \qquad n(p,j) = 0{,}6p + 0{,}4j.$$

O argumento é o par (p, j). De modo geral, modelos de aprendizado de máquina recebem vetores

$$\mathbf{x} = (x_1, x_2, \dots, x_d) \in \mathbb{R}^d$$

e produzem uma saída $f(\mathbf{x})$. Em classificação binária, por exemplo, pode-se ter $f\colon \mathbb{R}^d \to \{-1, +1\}$; em classificação com k classes, uma saída comum é um vetor de k probabilidades.

2.2.5 Funções totais e parciais

Pela definição adotada, $f\colon X \to Y$ precisa fornecer uma saída para todo $x \in X$; dizemos que ela é **total** em X. Em computação, contudo, operações podem falhar ou não terminar. A busca por uma chave inexistente, a divisão por zero ou um algoritmo que entra em laço infinito comportam-se como funções **parciais**.

Há duas maneiras usuais de modelar esse fato. Podemos restringir o domínio às entradas válidas ou ampliar o contradomínio com um valor que represente ausência ou erro. Por exemplo,

$$\text{buscar}\colon K \to V \cup \{\bot\},$$

em que $\bot$ significa "valor não encontrado". Tipos como `Option`, `Maybe` e `Result` tornam essa possibilidade explícita e ajudam o compilador a exigir o tratamento do caso excepcional.

2.2.6 Propriedades importantes

Seja $f\colon X \to Y$.

- f é **injetora** quando entradas diferentes produzem saídas diferentes:

$$x_1 \neq x_2 \Longrightarrow f(x_1) \neq f(x_2).$$

- f é **sobrejetora** quando todo elemento do contradomínio é atingido:

$$\text{im}(f) = Y.$$

- f é **bijetora** quando é simultaneamente injetora e sobrejetora.

Uma função hash comum ilustra a ausência de injetividade: o conjunto de entradas possíveis costuma ser muito maior do que o conjunto de códigos produzidos, portanto colisões são inevitáveis. Já uma codificação reversível precisa preservar informação suficiente para que entradas distintas não se confundam.

2.2.7 Composição de funções

Considere funções compatíveis

$$f\colon X \to Y \qquad \text{e} \qquad g\colon Y \to Z.$$

A **composição** de g com f é a função

$$g \circ f\colon X \to Z, \qquad (g \circ f)(x) = g(f(x)).$$

Primeiro aplicamos f e, depois, aplicamos g ao resultado. A ordem de leitura da expressão, portanto, é da direita para a esquerda.

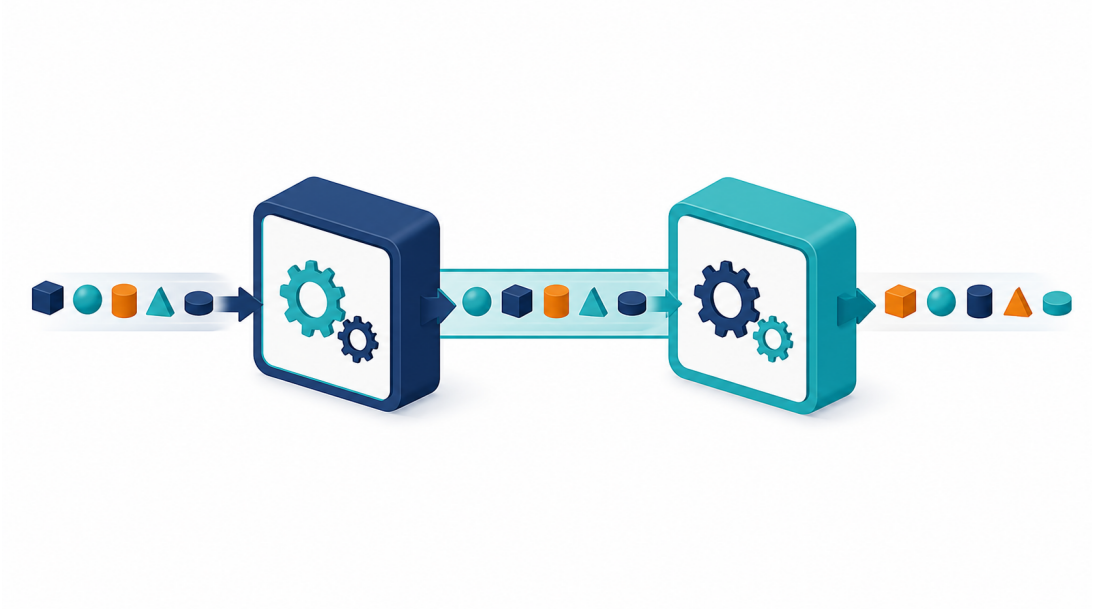


Figura 2.4: A composição conecta duas transformações: a saída da primeira função torna-se a entrada da segunda.

Por exemplo, sejam $f(x) = 2x + 1$ e $g(y) = y^2$. Então

$$(g \circ f)(x) = g(2x + 1) = (2x + 1)^2,$$

enquanto

$$(f \circ g)(x) = f(x^2) = 2x^2 + 1.$$

Em geral, $g \circ f \neq f \circ g$. A composição também só faz sentido quando as saídas da primeira etapa são entradas válidas para a segunda. Essa verificação corresponde, em programação, à compatibilidade entre tipos em um pipeline.

```
def normalizar(texto: str) -> str:
    return texto.strip().lower()

def comprimento(texto: str) -> int:
    return len(texto)

def tamanho_normalizado(texto: str) -> int:
    return comprimento(normalizar(texto))
```

Nesse exemplo, `normalizar` produz uma cadeia de caracteres, exatamente o tipo

aceito por `comprimento`. A função `tamanho_normalizado` implementa a composição comprimento $\circ$ normalizar.

A composição é **associativa**: quando os tipos são compatíveis,

$$h \circ (g \circ f) = (h \circ g) \circ f.$$

Essa propriedade permite construir sistemas complexos a partir de módulos menores, como ocorre em compiladores, fluxos de processamento de dados e camadas de redes neurais.

2.2.8 Função identidade

Para qualquer conjunto X, a função identidade é definida por

$$\mathrm{id}_X \colon X \to X, \qquad \mathrm{id}_X(x) = x.$$

Ela não modifica a entrada e funciona como elemento neutro da composição:

$$f \circ \mathrm{id}_X = f \qquad \text{e} \qquad \mathrm{id}_Y \circ f = f.$$

Embora pareça trivial, a identidade é útil para expressar interfaces uniformes, etapas opcionais de pipelines e conexões residuais em redes neurais.

2.2.9 Função inversa

Uma função $f \colon X \to Y$ possui uma **inversa** $f^{-1} \colon Y \to X$ quando podemos desfazer sua ação sem ambiguidade:

$$f^{-1}(f(x)) = x \quad \text{para todo } x \in X$$

e

$$f(f^{-1}(y)) = y \quad \text{para todo } y \in Y.$$

Isso acontece exatamente quando f é bijetora. A injetividade garante que a saída identifica uma única entrada; a sobrejetividade garante que toda saída declarada pode ser invertida.

Por exemplo,

$$f\colon \mathbb{R} \to \mathbb{R}, \qquad f(x) = 2x + 1$$

é bijetora. Para encontrar a inversa, escrevemos $y = 2x + 1$ e isolamos x:

$$x = \frac{y - 1}{2}.$$

Logo,

$$f^{-1}(y) = \frac{y - 1}{2}.$$

Podemos conferir compondo as duas funções:

$$f^{-1}(f(x)) = \frac{(2x + 1) - 1}{2} = x.$$

Já $q(x) = x^2$ não é injetora em $\mathbb{R}$, porque $q(x) = q(-x)$. Ao restringir o domínio a $[0, \infty)$ e usar o contradomínio $[0, \infty)$, obtemos uma bijeção cuja inversa é $q^{-1}(y) = \sqrt{y}$. Portanto, a invertibilidade não depende apenas da fórmula; depende também do domínio e do contradomínio.

Geometricamente, os gráficos de f e f^{-1} são reflexos um do outro em relação à reta $y = x$. Computacionalmente, inverter uma transformação significa recuperar a entrada a partir da saída. Criptografia reversível, serialização sem perdas e mudanças de coordenadas exploram essa ideia; já compressão com perdas e funções hash descartam informação e, por isso, não admitem inversão perfeita.

> **Não confunda**
>
> f^{-1} indica a função inversa, enquanto $1/f$ indica a função recíproca, $(1/f)(x) = 1/f(x)$. São conceitos diferentes. Por exemplo, se $f(x) = 2x$, então $f^{-1}(x) = x/2$, mas $(1/f)(x) = 1/(2x)$.

2.2.10 Funções em aprendizado de máquina

Um modelo preditivo é uma função parametrizada. Escrevemos

$$h_{\mathbf{w}}\colon X \to Y,$$

em que **w** representa os parâmetros aprendidos a partir dos dados. Durante o treinamento, um algoritmo modifica **w**; depois do treinamento, o modelo recebe uma entrada e calcula uma saída.

Uma camada linear, por exemplo, transforma um vetor $\mathbf{x} \in \mathbb{R}^d$ em

$$f(\mathbf{x}) = W\mathbf{x} + \mathbf{b}.$$

Uma rede neural é obtida pela composição de várias transformações. Em uma arquitetura simples,

$$h = f_3 \circ f_2 \circ f_1.$$

A entrada passa primeiro por f_1, depois por f_2 e finalmente por f_3. Assim, domínio, contradomínio, composição e imagem não são apenas abstrações: eles descrevem diretamente como dados percorrem um sistema computacional.

2.2.11 Exercícios de fixação

1. Para $f\colon \mathbb{Z} \to \mathbb{Z}$, $f(n) = 2n + 1$, determine $f(0)$, $f(-3)$ e a imagem de f. A função é sobrejetora no contradomínio declarado?
2. Explique por que a relação que associa uma pessoa a todos os seus números de telefone não é, sem uma convenção adicional, uma função com contradomínio formado por números individuais. Como você alteraria o contradomínio?
3. Sejam $f(x) = x - 1$ e $g(x) = 3x$. Calcule $(g \circ f)(2)$ e $(f \circ g)(2)$.
4. Determine um domínio e um contradomínio para os quais $f(x) = 1/x$ seja bijetora e encontre sua inversa.
5. Dê um exemplo de operação de programação que seja parcial. Modele-a como uma função total acrescentando ao contradomínio um valor de erro.
6. Considere um modelo $h\colon \mathbb{R}^{784} \to \mathbb{R}^{10}$. Interprete as dimensões de entrada e saída no contexto da classificação de imagens de algarismos com 28×28 pixels.

2.3 Algarismos

Números são objetos abstratos; **algarismos** são símbolos usados para representá-los. O número doze, por exemplo, pode ser escrito como 12 no sistema decimal, 1100 no sistema binário ou C no sistema hexadecimal. As três escritas representam a mesma quantidade, assim como as palavras “doze” e “twelve” expressam a mesma ideia em idiomas diferentes.

Para um estudante de Ciência da Computação, distinguir o número de sua representação é

fundamental. Computadores armazenam sequências de bits, mas programas podem exibir essas sequências em decimal, hexadecimal, como caracteres, cores ou instruções. O significado depende da interpretação escolhida.

2.3.1 Sistemas de numeração posicionais

Um sistema de numeração de base b utiliza b algarismos diferentes, com valores entre 0 e $b-1$. A posição de cada algarismo determina por qual potência da base ele será multiplicado. Para uma sequência inteira

$$(a_n a_{n-1} \dots a_1 a_0)_b,$$

seu valor é

$$\sum_{i=0}^{n} a_i b^i, \qquad 0 \le a_i < b.$$

O algarismo mais à direita ocupa a posição 0 e tem peso $b^0 = 1$. À medida que caminhamos para a esquerda, os pesos são multiplicados sucessivamente pela base.

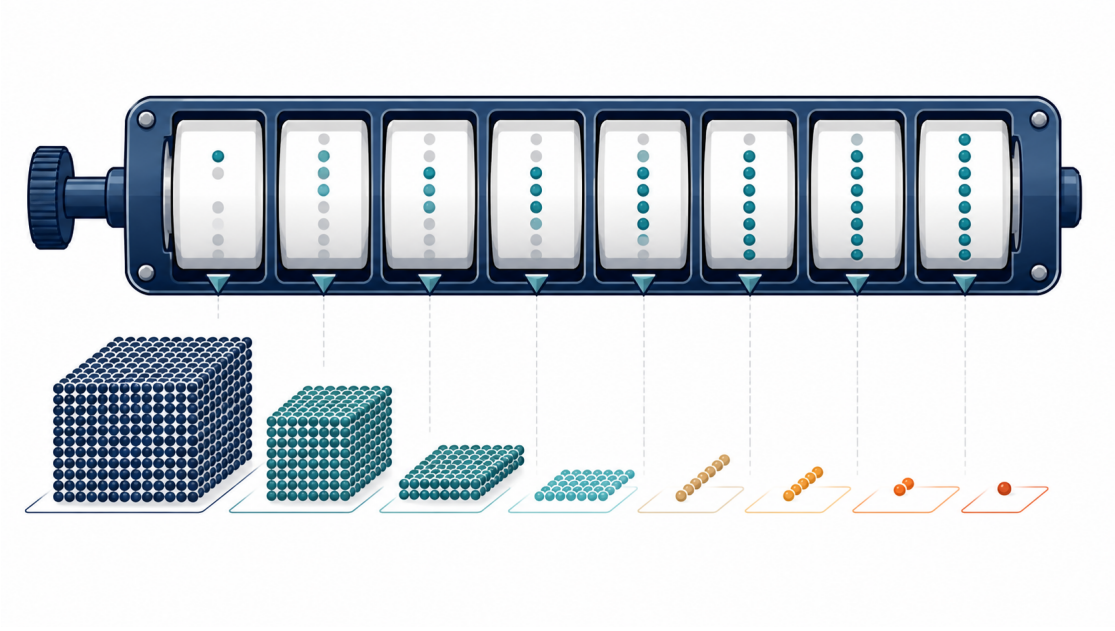


Figura 2.5: Em uma representação posicional, cada casa possui um peso; as casas à esquerda representam agrupamentos progressivamente maiores.

No sistema decimal, a base é 10 e os algarismos disponíveis são $0, 1, \dots, 9$. Assim,

$$(321)_{10} = 3 \cdot 10^2 + 2 \cdot 10^1 + 1 \cdot 10^0.$$

Os parênteses e o índice indicam explicitamente a base. Quando não houver risco de ambiguidade, omitiremos o índice decimal.

O zero exerce dois papéis importantes: pode representar a quantidade nula e também reservar uma posição. Em 205, o zero informa que não há dezenas; sem ele, 25 representaria outro número.

> **Uma analogia útil**
>
> Um hodômetro é um contador posicional. Quando uma roda completa todos os símbolos disponíveis, ela retorna a zero e produz um "vai um" para a roda à esquerda. Em base 10, isso ocorre após dez estados; em base 2, após apenas dois.

2.3.2 Parte fracionária

Posições à direita do separador representam potências negativas da base. Em base b,

$$(a_n a_{n-1} \ldots a_0, a_{-1} a_{-2} \ldots)_b = \sum_{i=-m}^{n} a_i b^i.$$

Por exemplo,

$$(321,34)_{10} = 3 \cdot 10^2 + 2 \cdot 10^1 + 1 \cdot 10^0 + 3 \cdot 10^{-1} + 4 \cdot 10^{-2}.$$

Em notação brasileira usamos vírgula como separador decimal. Linguagens de programação normalmente exigem ponto, como em `321.34`.

2.3.3 Base binária

O sistema **binário** tem base 2 e utiliza somente os algarismos 0 e 1. Cada algarismo binário é chamado de **bit**, contração de *binary digit*. Os pesos das posições inteiras são

$$\ldots, 2^5, 2^4, 2^3, 2^2, 2^1, 2^0 = \ldots, 32, 16, 8, 4, 2, 1.$$

Para converter uma representação binária em decimal, somamos os pesos das posições que contêm 1. Por exemplo,

$$(1101)_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 8 + 4 + 0 + 1 = (13)_{10}.$$

O binário é apropriado para circuitos digitais porque dois estados físicos distinguíveis podem representar 0 e 1: tensão baixa e alta, chave aberta e fechada ou ausência e presença de carga. Isso não significa que o computador "compreenda" números binários; significa apenas que convencionamos interpretar estados físicos dessa maneira.

2.3.4 Conversão de decimal para binário

Para converter um inteiro decimal não negativo em binário, podemos dividi-lo repetidamente por 2 e registrar os restos. Os restos devem ser lidos da última divisão para a primeira.

Convertamos $(45)_{10}$:

Divisão	Quociente	Resto
$45 \div 2$	22	1
$22 \div 2$	11	0
$11 \div 2$	5	1
$5 \div 2$	2	1
$2 \div 2$	1	0
$1 \div 2$	0	1

Lendo os restos de baixo para cima, obtemos

$$(45)_{10} = (101101)_2.$$

Podemos verificar: $32 + 8 + 4 + 1 = 45$.

Em Python, `bin(45)` devolve a cadeia `'0b101101'`. O prefixo `0b` informa que a representação está em base 2; ele não faz parte dos algarismos do número.

2.3.5 Base hexadecimal

Representações binárias longas são difíceis de ler. O sistema **hexadecimal**, de base 16, oferece uma escrita compacta e alinhada aos bits. Ele usa os algarismos

$$0, 1, \ldots, 9, \mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}, \mathrm{F},$$

em que A a F representam os valores decimais 10 a 15. Por exemplo,

$$(\text{A02F})_{16} = 10 \cdot 16^3 + 0 \cdot 16^2 + 2 \cdot 16^1 + 15 \cdot 16^0 = (41007)_{10}.$$

Como $16 = 2^4$, cada algarismo hexadecimal corresponde exatamente a quatro bits:

Binário	Hexadecimal	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Para converter binário em hexadecimal, agrupamos os bits de quatro em quatro, começando pela direita. Se necessário, completamos o grupo da esquerda com zeros:

$$(10110110)_2 = (1011\ 0110)_2 = (\text{B6})_{16}.$$

O processo inverso consiste em substituir cada algarismo hexadecimal por seus quatro bits. Assim,

$$(\text{3A})_{16} = (0011\ 1010)_2.$$

Em programas, o prefixo `0x` costuma indicar hexadecimal: `0x3A`. Endereços de memória, máscaras de bits, códigos de cor e ferramentas de depuração frequentemente usam essa base.

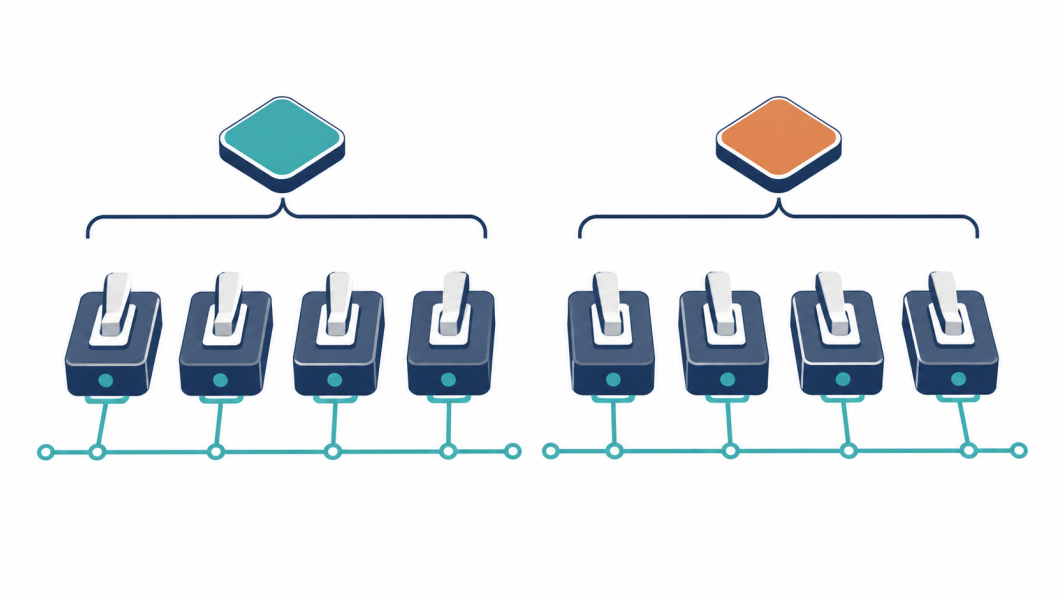


Figura 2.6: Um byte contém oito bits e pode ser separado em dois grupos de quatro; cada grupo corresponde a um algarismo hexadecimal.

2.3.6 Bits, nibbles, bytes e capacidade

Um grupo de quatro bits é informalmente chamado de **nibble**. Um grupo de oito bits é um **byte**. Com n bits existem

$$2^n$$

sequências distintas. Um byte, portanto, admite $2^8 = 256$ padrões, de `00000000` a `11111111`. Se interpretados como inteiros sem sinal, esses padrões representam os valores de 0 a 255.

Em geral, n bits representam inteiros sem sinal no intervalo

$$0 \leq x \leq 2^n - 1.$$

O limite superior contém -1 porque uma das 2^n combinações é usada para representar o zero.

Quantidade de bits	Número de padrões	Intervalo sem sinal
4	16	0 a 15
8	256	0 a 255
16	65 536	0 a 65 535
32	4 294 967 296	0 a 4 294 967 295

2.3.7 Inteiros com sinal e complemento de dois

Para representar números negativos, computadores modernos geralmente usam **complemento de dois**. Com n bits, o intervalo passa a ser

$$-2^{n-1} \leq x \leq 2^{n-1} - 1.$$

Em oito bits, isso corresponde a -128 até 127. O bit mais à esquerda possui peso -2^{n-1}, enquanto os demais mantêm pesos positivos. Por exemplo,

$$(11111101)_2 = -2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 0 \cdot 2^1 + 2^0 = -3.$$

Uma maneira prática de obter a representação de $-x$ é inverter todos os bits da representação de x e somar 1. Em oito bits:

$$3 = 00000011 \longrightarrow 11111100 \longrightarrow 11111101 = -3.$$

Na etapa intermediária, todos os bits foram invertidos; na etapa final, somamos `1`.

> ⚠ Overflow
>
> Uma quantidade fixa de bits possui capacidade finita. Em um inteiro sem sinal de oito bits, $255 + 1$ produz `00000000` se o cálculo descartar o transporte final. Esse retorno ao início é chamado de **overflow** e pode causar erros graves quando não é previsto.

2.3.8 Frações binárias e representação inexata

As posições à direita do separador binário têm pesos $2^{-1}, 2^{-2}, 2^{-3}, \ldots$. Portanto,

$$(10,101)_2 = 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = (2,625)_{10}.$$

Nem toda fração decimal possui representação binária finita. O número decimal $0,1$, por exemplo, torna-se uma expansão binária infinita. Como a memória é finita, o computador armazena uma aproximação. Isso explica por que, em muitas linguagens,

```
0.1 + 0.2 == 0.3
```

resulta em `False`.

Números de ponto flutuante normalmente são armazenados em uma forma análoga à notação científica, com sinal, significando e expoente. O padrão IEEE 754 especifica formatos amplamente usados. Para cálculos financeiros, em que centavos precisam ser exatos, pode ser melhor usar inteiros ou tipos decimais apropriados.

2.3.9 Bits não possuem significado isolado

A sequência `01000001` pode ser interpretada de várias maneiras. Como inteiro sem sinal, vale 65. Segundo a codificação ASCII, representa o caractere `A`. Também pode ser parte de uma instrução, de um pixel ou de um número em ponto flutuante. Os bits são os mesmos; o **tipo** e o contexto determinam a interpretação.

Essa observação conecta sistemas de numeração a um tema central da computação: dados precisam de uma convenção de codificação. Um arquivo não contém "uma imagem" ou "um texto" de maneira intrínseca; contém bytes interpretados segundo um formato.

2.3.10 Operações bit a bit

Linguagens de programação oferecem operações que atuam diretamente sobre os bits:

- `AND` preserva uma posição somente quando os dois bits são 1;
- `OR` produz 1 quando pelo menos um dos bits é 1;
- `XOR` produz 1 quando os bits são diferentes;
- `NOT` inverte cada bit;
- deslocamentos movem os bits para a esquerda ou para a direita.

Por exemplo,

$$1010_2 \ \text{AND} \ 1100_2 = 1000_2.$$

Deslocar um inteiro não negativo uma posição para a esquerda corresponde, quando não há overflow, a multiplicá-lo por 2. Deslocar para a direita corresponde a uma divisão inteira por 2. Essas operações aparecem em máscaras de permissões, protocolos de comunicação, compressão e programação de baixo nível.

2.3.11 Exemplo em Python

O código seguinte mostra diferentes representações do mesmo inteiro:

```
n = 45

print(bin(n))  # 0b101101
print(oct(n))  # 0o55
print(hex(n))  # 0x2d
print(int("101101", 2))  # 45
print(f"{n:08b}")        # 00101101
```

`bin`, `oct` e `hex` devolvem textos. O valor armazenado em `n` não "está em decimal" ou "está em hexadecimal"; essas bases dizem respeito à maneira escolhida para escrever ou exibir o valor.

2.3.12 Estratégias para evitar erros

Ao trabalhar com representações numéricas, convém seguir algumas práticas:

1. Indique a base quando ela não estiver evidente: $(101)_2$, $(101)_{10}$ e $(101)_{16}$ são números diferentes.
2. Verifique o intervalo permitido pelo tipo antes de realizar operações.
3. Não compare números de ponto flutuante esperando igualdade exata; use uma tolerância adequada.
4. Diferencie valor numérico de texto. A cadeia `"10"` possui dois caracteres; o inteiro `10` é um valor numérico.
5. Use hexadecimal para inspecionar grupos de bits, mas converta cada algarismo com atenção.

2.3.13 Exercícios de fixação

1. Expanda $(5072)_{10}$ como soma de potências de 10.
2. Converta $(101011)_2$ para decimal e confirme o resultado somando os pesos das posições.
3. Converta $(73)_{10}$ para binário pelo método das divisões sucessivas.
4. Converta $(11011110)_2$ para hexadecimal e $(7\mathrm{A})_{16}$ para binário.
5. Qual é o maior inteiro sem sinal representável com 12 bits? Quantos padrões diferentes existem?
6. Determine o intervalo de inteiros com sinal em complemento de dois para 16 bits.
7. Interprete `01000001` como inteiro sem sinal e pesquise qual caractere ASCII possui esse código.
8. Explique, com suas palavras, por que $0,1$ não pode ser representado exatamente por um número binário finito.

9. Calcule `10110100 AND 11110000` e explique como essa operação pode servir para selecionar os quatro bits mais significativos de um byte.

Capítulo 3

O que é um problema de aprendizado?

Programar um computador significa especificar um procedimento capaz de transformar entradas em saídas. Em muitos problemas, conhecemos bem esse procedimento. Para calcular a área de um círculo, por exemplo, fornecemos ao computador a fórmula $A(r) = \pi r^2$. Para ordenar uma lista, implementamos um algoritmo de ordenação. As regras são escritas por uma pessoa e executadas pela máquina.

Há, porém, tarefas para as quais é difícil formular todas as regras explicitamente. Como escrever uma lista completa de instruções para reconhecer um rosto, detectar uma tentativa de fraude ou estimar o preço de uma residência? Uma pessoa realiza algumas dessas tarefas pela experiência, mas pode não conseguir descrever cada decisão como uma sequência rígida de condições. Nesses casos, podemos fornecer **exemplos** e fazer um algoritmo ajustar uma função que reproduza, tão bem quanto possível, os padrões observados.

Esse processo é chamado de **aprendizado de máquina**. O computador não aprende no mesmo sentido que uma pessoa, nem descobre necessariamente uma lei verdadeira da natureza. Ele executa um algoritmo de otimização que escolhe, dentro de uma família de funções possíveis, aquela que obtém um bom desempenho segundo um critério matemático.

3.1 Programação convencional e aprendizado de máquina

Na programação convencional, dados e regras são fornecidos ao computador, que calcula uma saída:

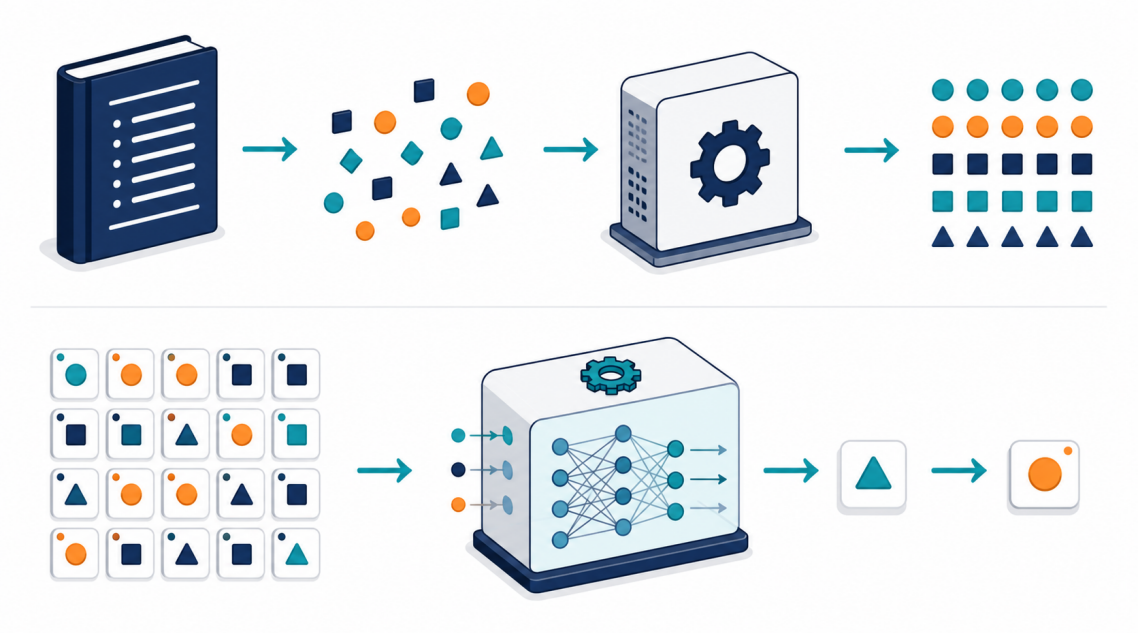
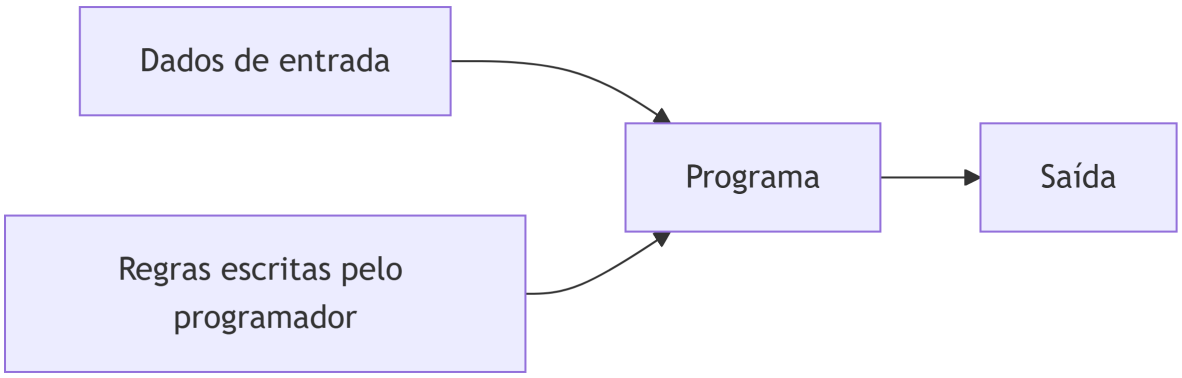
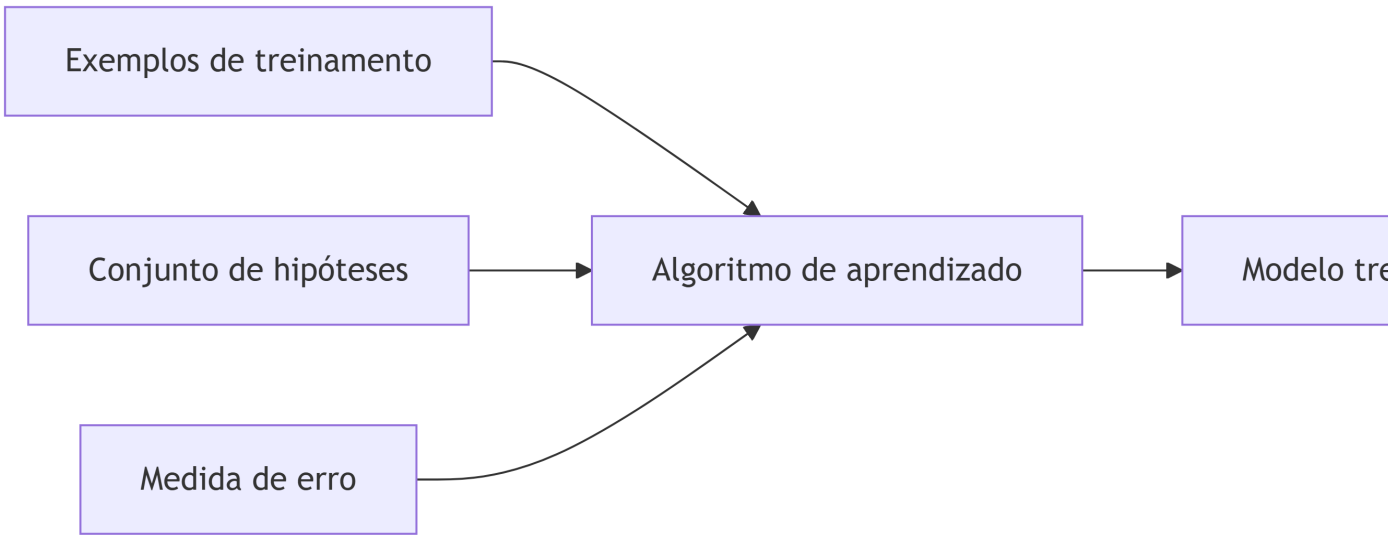


Figura 3.1: Na programação convencional, regras orientam o processamento dos dados; no aprendizado de máquina, exemplos são usados para ajustar um modelo que fará previsões sobre novas entradas.

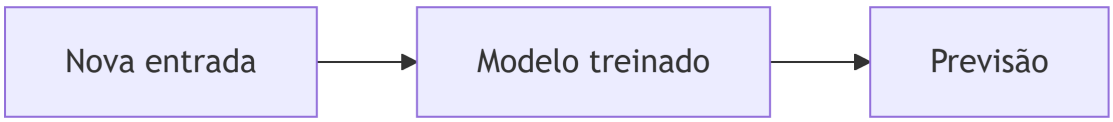


Um programa que calcula o imposto devido é um exemplo típico. As alíquotas e condições são conhecidas, codificadas e aplicadas aos dados do contribuinte. Se a legislação mudar, alguém deve alterar as regras do programa.

No aprendizado supervisionado, os dados de treinamento já incluem exemplos de entradas e das saídas desejadas. Um algoritmo utiliza esses pares para produzir um modelo:



Depois do treinamento, o modelo é usado para calcular previsões:



É importante separar essas duas fases. **Treinar** significa escolher ou ajustar o modelo usando dados conhecidos. **Inferir** ou **predizer** significa aplicar o modelo já treinado a uma nova entrada. O treinamento pode exigir milhões de exemplos e grande capacidade computacional; uma inferência individual pode ser muito mais barata.

3.2 Elementos de um problema de aprendizado

Um problema de aprendizado supervisionado pode ser descrito inicialmente por cinco componentes.

1. **Entradas:** objetos sobre os quais queremos tomar uma decisão. Representamos o conjunto de entradas possíveis por X.
2. **Saídas:** respostas que desejamos produzir. O conjunto de saídas possíveis é denotado por Y.
3. **Dados:** exemplos observados, frequentemente pares (x_i, y_i) com $x_i \in X$ e $y_i \in Y$.
4. **Modelo:** uma função $h \colon X \to Y$, também chamada de hipótese, usada para produzir previsões.
5. **Critério de qualidade:** uma medida que quantifica o quanto as previsões de h diferem das respostas observadas.

O algoritmo de aprendizado recebe os dados e procura uma hipótese que tenha erro pequeno.

Em notação compacta, uma amostra supervisionada com N exemplos é

$$D = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}.$$

O objetivo não é apenas memorizar esses pares. Queremos que a função aprendida produza boas respostas para entradas que não estavam na amostra. Essa capacidade é chamada de **generalização**.

> **!** Aprender não é memorizar
>
> Um modelo que acerta todos os exemplos de treinamento pode ainda falhar diante de novos dados. O sucesso de um sistema de aprendizado é medido principalmente pelo desempenho fora da amostra usada para ajustá-lo.

3.3 Um exemplo: filtragem de spam

Considere o problema de classificar mensagens como `spam` ou `não spam`. O espaço de entrada X é formado pelas mensagens possíveis, e o espaço de saída é

$$Y = \{0, 1\},$$

em que podemos convencionar 1 para `spam` e 0 para `não spam`.

O computador não manipula diretamente o significado linguístico de uma mensagem. Primeiro, precisamos transformá-la em atributos numéricos, como número de links, frequência de certas palavras, tamanho do assunto ou identidade do remetente. Essa transformação produz um vetor

$$\mathbf{x} = (x_1, x_2, \dots, x_d) \in \mathbb{R}^d.$$

Um conjunto de mensagens previamente rotuladas forma os dados de treinamento. O algoritmo ajusta um classificador $h \colon \mathbb{R}^d \to \{0, 1\}$. Para uma nova mensagem, calculamos $h(\mathbf{x})$ e usamos o resultado como previsão.

Mesmo nesse exemplo simples surgem decisões importantes:

- os e-mails rotulados representam adequadamente as mensagens futuras?
- os rótulos estão corretos?
- quais atributos devem ser extraídos?
- qual tipo de função será usado como hipótese?

- qual é o custo de bloquear uma mensagem legítima em comparação com deixar passar um spam?

Essas perguntas mostram que aprendizado de máquina não consiste apenas em executar um algoritmo. É necessário formular o problema, obter dados adequados, escolher representações e avaliar riscos.

3.4 Tipos de saída

A natureza de Y ajuda a identificar o tipo de tarefa.

- Na **classificação**, a saída pertence a um conjunto finito de categorias. Exemplos: `spam` ou `não spam`; espécie de uma planta; algarismo presente em uma imagem.
- Na **regressão**, a saída é numérica. Exemplos: preço de uma casa, temperatura futura ou tempo necessário para concluir uma tarefa.
- No **ranqueamento**, o objetivo é ordenar itens por relevância, como resultados de uma busca.
- Na **geração**, a saída pode ser uma sequência complexa, como texto, áudio, imagem ou código.

Um mesmo contexto pode originar formulações diferentes. Para analisar risco de crédito, podemos prever `pagará` ou `não pagará` (classificação), estimar a probabilidade de pagamento (regressão probabilística) ou ordenar clientes do menor para o maior risco (ranqueamento).

3.5 De onde vêm os exemplos?

Dados não aparecem prontos e neutros. Eles são coletados por sensores, formulários, registros de sistemas, experimentos ou pessoas responsáveis por rotular exemplos. Cada mecanismo de coleta introduz escolhas e possíveis erros.

Uma amostra pode ser **enviesada** quando representa alguns grupos ou situações melhor do que outros. Dados históricos também podem preservar decisões injustas do passado. Além disso, o ambiente pode mudar: padrões de fraude, linguagem usada em spam e preferências de usuários não permanecem constantes para sempre.

Por isso, a origem dos dados deve ser documentada. Devemos perguntar quem ou o que ficou de fora, como os rótulos foram definidos, quais erros de medição existem e se o uso pretendido é compatível com o consentimento e a privacidade das pessoas envolvidas.

3.6 Hipótese, parâmetros e algoritmo

Três conceitos próximos precisam ser distinguidos:

- a **hipótese** é uma função candidata $h \colon X \to Y$;
- os **parâmetros** são valores internos que determinam uma hipótese específica;
- o **algoritmo de aprendizado** é o procedimento que escolhe os parâmetros a partir dos dados.

Por exemplo, uma função linear pode ser escrita como

$$h_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w}^{\mathsf{T}}\mathbf{x} + b.$$

Os valores de $\mathbf{w}$ e b são parâmetros. O conjunto de todas as funções obtidas ao variar esses parâmetros é uma família de hipóteses. Um algoritmo de treinamento procura valores que reduzam o erro nos exemplos disponíveis.

Em uma rede neural, há muitos parâmetros organizados em matrizes e vetores. Embora a arquitetura seja especificada pelo programador, os valores desses parâmetros são ajustados durante o treinamento.

3.7 Erro e objetivo de treinamento

Para comparar uma previsão $h(x_i)$ com a resposta correta y_i, usamos uma **função de perda**. Em regressão, uma escolha simples é o erro quadrático:

$$\ell(h(x_i), y_i) = (h(x_i) - y_i)^2.$$

O erro médio na amostra é então

$$E_D(h) = \frac{1}{N} \sum_{i=1}^{N} \ell(h(x_i), y_i).$$

Treinar pode ser visto como procurar uma hipótese com valor pequeno de $E_D(h)$. Entretanto, minimizar apenas o erro de treinamento pode favorecer modelos que memorizam detalhes acidentais. Mais adiante discutiremos separação entre treino e teste, regularização e limites de generalização.

3.8 Quando usar aprendizado de máquina?

Aprendizado de máquina tende a ser útil quando há dados representativos, uma meta mensurável e uma relação entre entradas e saídas que pode ser explorada. Também é importante que o padrão procurado seja relativamente estável e que os erros possam ser monitorados.

Nem todo problema precisa de aprendizado. Uma regra explícita é geralmente preferível quando é conhecida, curta, confiável e fácil de manter. Não faz sentido treinar um modelo para converter graus Celsius em Fahrenheit ou verificar se um número inteiro é par. Nesses casos, a solução determinística é mais transparente e exata.

Também devemos ser cautelosos quando há poucos dados, quando o custo de um erro é muito alto ou quando não existe uma forma responsável de avaliar o sistema. Em aplicações médicas, jurídicas, financeiras ou de segurança, previsões precisam de validação rigorosa, supervisão humana e análise das consequências.

3.9 Um roteiro para formular o problema

Antes de escolher um algoritmo, é útil responder às seguintes perguntas:

1. Qual decisão ou previsão queremos produzir?
2. O que constitui uma entrada e quais informações estarão disponíveis no momento da previsão?
3. Qual é o espaço de saídas?
4. De onde virão os exemplos e como serão rotulados?
5. Como mediremos um erro e quais tipos de erro são mais graves?
6. Como verificaremos o desempenho em dados ainda não vistos?
7. Como o sistema será monitorado após entrar em uso?

Esse roteiro evita um erro comum: começar pelo modelo antes de definir claramente o problema. Um sistema tecnicamente sofisticado não compensa uma pergunta mal formulada ou dados inadequados.

3.10 Exercícios de fixação

1. Compare programação convencional e aprendizado de máquina usando como exemplo a detecção de mensagens de spam.
2. Para um sistema que estima o tempo de entrega de uma encomenda, identifique X, Y, possíveis atributos e o tipo de tarefa.
3. Explique a diferença entre treinamento e inferência.

4. Dê um exemplo de modelo que memoriza os dados de treinamento, mas não generaliza.
5. Proponha uma tarefa para a qual uma regra explícita seja melhor do que aprendizado de máquina e justifique.
6. Em um diagnóstico assistido por computador, cite dois tipos de erro com consequências diferentes. Como essa diferença deveria influenciar a avaliação?
7. Explique a diferença entre hipótese, parâmetros e algoritmo de aprendizado no caso de uma função linear.

3.11 Categorias de Aprendizado

Problemas de aprendizado podem ser organizados de acordo com o tipo de informação disponível durante o treinamento e com a forma pela qual o sistema recebe retorno sobre suas decisões. Essa classificação orienta a preparação dos dados, a escolha dos algoritmos e a maneira de avaliar os resultados.

As categorias não são caixas absolutamente isoladas. Um sistema real pode combinar várias delas: um modelo pode ser pré-treinado de forma autossupervisionada, refinado com exemplos rotulados e depois ajustado com feedback humano por reforço. Ainda assim, compreender os paradigmas separadamente fornece um vocabulário útil para projetar soluções.

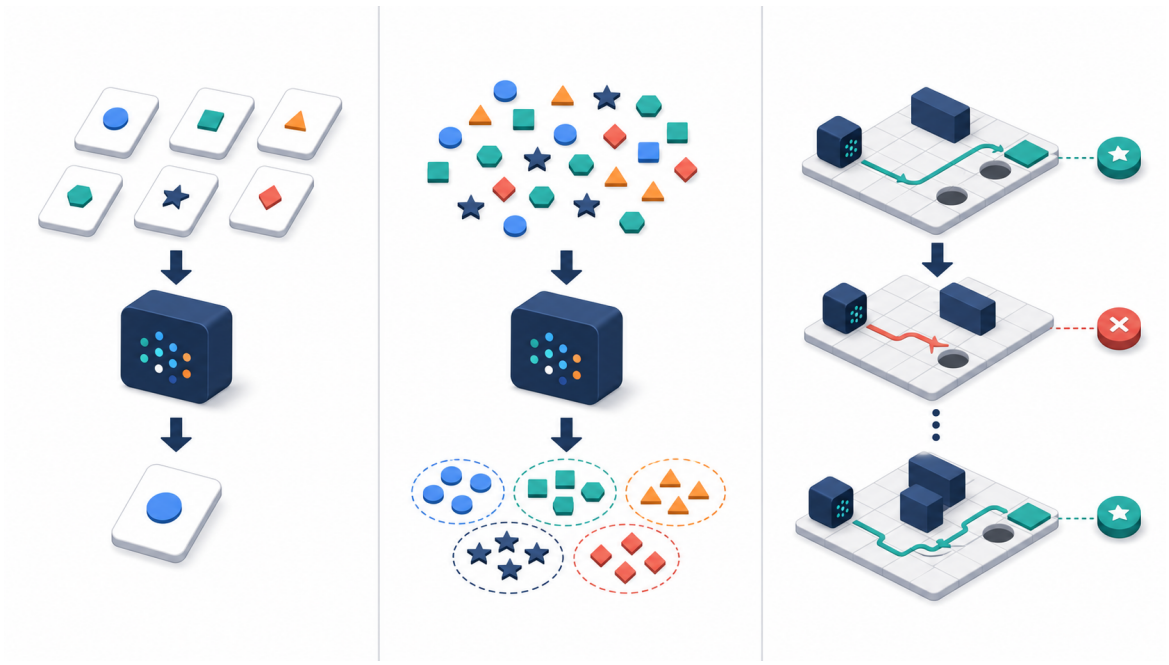


Figura 3.2: Três paradigmas: exemplos rotulados no aprendizado supervisionado, descoberta de estrutura no não supervisionado e interação com recompensas no aprendizado por reforço.

3.11.1 Especificar regras ou aprender com dados?

Antes de comparar as categorias, convém distinguir duas formas de construir um sistema. Na **especificação direta**, o programador determina explicitamente as regras. Para verificar se uma senha possui pelo menos doze caracteres, basta escrever uma condição. A regra é conhecida, verificável e não precisa ser estimada.

No aprendizado de máquina, o programador especifica a representação dos dados, a família de modelos, a medida de erro e o procedimento de treinamento. As decisões internas do modelo são ajustadas usando exemplos. Portanto, o sistema não elimina a especificação humana; ele desloca parte dela das regras finais para o **processo pelo qual as regras paramétricas serão ajustadas**.

Considere a identificação de moedas por diâmetro e massa. Se o fabricante fornece intervalos exatos e estáveis para cada moeda, podemos codificar esses limites diretamente. Se as medições variam por desgaste, ruído do sensor ou diferenças de fabricação, podemos coletar exemplos e ajustar fronteiras de decisão. A escolha entre regras e aprendizado depende do conhecimento disponível, da variabilidade do problema e do custo dos erros.

> **Uma distinção prática**
>
> Regras definidas antes de observar exemplos constituem uma especificação. Parâmetros ajustados a partir dos exemplos constituem aprendizado. Na prática, sistemas frequentemente combinam ambos: regras garantem restrições obrigatórias, enquanto modelos tratam padrões difíceis de descrever.

3.11.2 Aprendizado supervisionado

No **aprendizado supervisionado**, cada exemplo de treinamento contém uma entrada e uma saída desejada:

$$D = \{(x_i, y_i)\}_{i=1}^{N}.$$

O termo “supervisionado” não significa necessariamente que uma pessoa acompanha cada etapa do treinamento. Significa que existe um **alvo** y_i associado a cada entrada x_i. Esse alvo pode ter sido fornecido por especialistas, produzido por sensores, extraído de registros históricos ou calculado por outro processo.

O algoritmo procura uma função $h\colon X \to Y$ que aproxime a relação observada. Há dois subtipos centrais:

- **classificação**, quando Y contém categorias discretas, como `spam` e `não spam`;

- **regressão**, quando Y é numérico, como preço, temperatura ou duração.

Por exemplo, para classificar e-mails, uma amostra pode conter vetores de atributos e seus rótulos:

$$(\mathbf{x}_i, y_i), \qquad \mathbf{x}_i \in \mathbb{R}^d, \quad y_i \in \{0, 1\}.$$

O treinamento usa os rótulos para calcular o erro e corrigir o modelo. Depois, o classificador recebe uma mensagem ainda não rotulada e estima sua classe.

Exemplos de aplicações supervisionadas incluem reconhecimento de caracteres, previsão de inadimplência, diagnóstico assistido, detecção de objetos e estimativa de demanda.

3.11.2.1 Qualidade dos rótulos

Rótulos são medições e podem conter erros. Dois especialistas podem discordar sobre um diagnóstico; usuários podem marcar mensagens legítimas como spam; registros históricos podem refletir decisões enviesadas. Um modelo não possui acesso direto à "verdade": ele aprende a partir dos alvos fornecidos.

Também pode ocorrer **vazamento de dados** quando um atributo contém, direta ou indiretamente, informação que só estaria disponível depois da resposta que queremos prever. Um modelo de risco hospitalar não deve usar um procedimento realizado após o diagnóstico como entrada para prever esse mesmo diagnóstico.

3.11.3 Aprendizado não supervisionado

No **aprendizado não supervisionado**, a amostra contém entradas sem uma saída-alvo explícita:

$$D = \{x_i\}_{i=1}^{N}.$$

O objetivo é encontrar estrutura nos dados: grupos, direções de maior variação, representações compactas ou exemplos atípicos. Como não há rótulos corretos disponíveis durante o treinamento, a avaliação exige critérios diferentes daqueles usados na classificação supervisionada.

As tarefas mais comuns incluem:

- **agrupamento (*clustering*):** reunir exemplos semelhantes;
- **redução de dimensionalidade:** representar os dados usando menos variáveis;
- **detecção de anomalias:** localizar observações que diferem do padrão predominante;
- **estimativa de densidade:** modelar regiões mais e menos prováveis do espaço de dados.

No exemplo das moedas, podemos representar cada moeda por massa e diâmetro. Um algoritmo de agrupamento pode separar concentrações de pontos sem conhecer os valores monetários. Contudo, os grupos encontrados não vêm automaticamente acompanhados dos nomes “dez centavos” ou “um real”. Além disso, dependendo da medida de distância e do algoritmo, os agrupamentos podem refletir tamanho, material, estado de conservação ou simplesmente ruído.

> ⚠ Agrupamentos não são verdades ocultas
>
> Um algoritmo não supervisionado encontra estrutura segundo hipóteses matemáticas escolhidas por nós. Mudar a escala dos atributos, a noção de similaridade ou o número de grupos pode alterar completamente o resultado.

3.11.4 Aprendizado autossupervisionado

No **aprendizado autossupervisionado**, os alvos são construídos a partir dos próprios dados, sem a necessidade de rotulação manual para cada exemplo. Escondemos ou transformamos parte da entrada e treinamos o modelo para recuperá-la.

Em modelos de linguagem, uma tarefa comum é prever o próximo token:

$$(w_1, w_2, \dots, w_t) \longmapsto w_{t+1}.$$

O texto fornece tanto a entrada quanto o alvo. Em visão computacional, podemos ocultar regiões de uma imagem e pedir ao modelo que reconstrua o conteúdo ausente. Essas tarefas permitem aprender representações gerais usando grandes volumes de dados não rotulados manualmente.

Embora os alvos sejam gerados automaticamente, o treinamento continua matematicamente semelhante ao supervisionado: existe uma resposta esperada e uma função de perda. A diferença está na origem da supervisão.

3.11.5 Aprendizado semissupervisionado

Rotular dados pode ser caro, enquanto coletar entradas sem rótulo pode ser fácil. O **aprendizado semissupervisionado** combina um pequeno conjunto rotulado

$$D_L = \{(x_i, y_i)\}$$

com um conjunto maior sem rótulos

$$D_U = \{x_j\}.$$

O método tenta aproveitar a estrutura de D_U para melhorar a previsão, sem tratar automaticamente toda predição do próprio modelo como verdade. Estratégias incluem pseudorrótulos, regularização por consistência e propagação de rótulos em grafos.

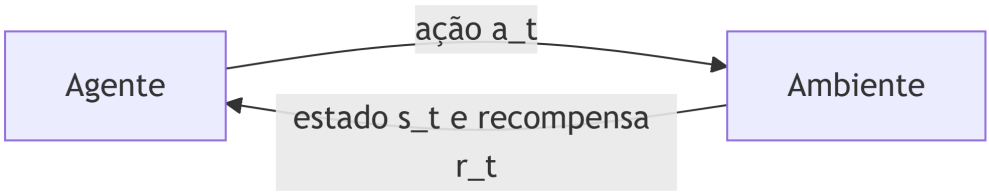
Essa abordagem é útil em imagens médicas, áudio e documentos, nos quais especialistas podem rotular apenas uma fração do material disponível.

3.11.6 Aprendizado por reforço

No **aprendizado por reforço**, um **agente** interage repetidamente com um **ambiente**. Em cada instante t, ele observa um estado s_t, escolhe uma ação a_t, recebe uma recompensa r_{t+1} e chega a um novo estado s_{t+1}:

$$s_t \xrightarrow{a_t} (r_{t+1}, s_{t+1}).$$

O objetivo não é simplesmente prever um rótulo, mas aprender uma **política** π que escolha ações capazes de maximizar a recompensa acumulada ao longo do tempo.



Uma recompensa imediata pode não revelar a qualidade de uma ação. Em um jogo de xadrez, sacrificar uma peça reduz a vantagem material naquele momento, mas pode contribuir para a vitória muitos lances depois. Esse é o problema de **atribuição temporal de crédito**: determinar quais decisões anteriores foram responsáveis pelo resultado futuro.

3.11.6.1 Exploração e aproveitamento

O agente enfrenta um dilema:

- **aproveitar** (*exploitation*) ações que já parecem boas;
- **explorar** ações menos conhecidas que podem revelar estratégias melhores.

Se apenas aproveitar o conhecimento atual, pode permanecer preso a uma estratégia mediana. Se explorar o tempo todo, deixa de usar o que aprendeu. Algoritmos de reforço equilibram esses comportamentos de diferentes maneiras.

Aplicações incluem controle robótico, jogos, alocação de recursos e sistemas de recomendação sequencial. O paradigma exige cuidado: uma recompensa mal definida pode induzir comportamentos que maximizam a pontuação formal sem cumprir a intenção humana.

3.11.7 Aprendizado on-line e em lote

Outra distinção diz respeito ao momento em que os dados chegam.

No **aprendizado em lote** (*batch*), o modelo é treinado usando um conjunto previamente reunido. Novos dados só influenciam o sistema em um treinamento posterior. No **aprendizado on-line**, os parâmetros são atualizados à medida que observações chegam.

Aprendizado on-line é útil para fluxos contínuos e ambientes que mudam, mas requer mecanismos contra ruído, ataques e esquecimento de padrões anteriores. "On-line" nesse contexto não significa necessariamente conectado à internet; significa atualização incremental.

3.11.8 Comparação entre paradigmas

Paradigma	Informação de treinamento	Objetivo típico	Exemplo
Supervisionado	pares (x, y)	prever y para nova entrada	classificar spam
Não supervisionado	entradas x	descobrir estrutura	agrupar clientes
Autossupervisionado	alvos derivados dos dados	aprender representações	prever próximo token
Semissupervisionado	poucos rótulos e muitas entradas	melhorar previsão com dados não rotulados	classificar imagens médicas
Por reforço	estados, ações e recompensas	maximizar retorno acumulado	aprender uma estratégia de jogo

Essa tabela descreve a fonte do sinal de aprendizado, e não o algoritmo específico. Redes neurais, por exemplo, podem ser usadas em todos esses paradigmas.

3.11.9 Como escolher a categoria?

A escolha começa pela informação disponível e pela pergunta que desejamos responder:

1. Há uma saída correta conhecida para cada exemplo? O problema pode ser supervisionado.

2. Há muitos dados, mas poucos rótulos? Considere aprendizado semissupervisionado ou autossupervisionado.
3. Queremos explorar a estrutura dos dados sem prever um alvo conhecido? Trata-se de uma tarefa não supervisionada.
4. As decisões alteram estados futuros e o retorno ocorre ao longo do tempo? O paradigma de reforço pode ser apropriado.

Não devemos escolher uma categoria apenas porque está em evidência. Se existe uma regra exata, ela pode ser mais adequada do que qualquer modelo aprendido. Se não há dados capazes de medir o objetivo, mudar o algoritmo não resolve a formulação incompleta.

3.11.10 Exercícios de fixação

1. Classifique como supervisionada, não supervisionada ou por reforço: prever o preço de uma casa; agrupar notícias por assunto; aprender a controlar um robô por recompensas.
2. Explique por que prever o próximo token é chamado de aprendizado autossupervisionado.
3. Um agrupamento separou clientes por região, mas a equipe esperava grupos por comportamento de compra. O resultado está "errado"? Discuta o papel dos atributos e da métrica de similaridade.
4. Dê um exemplo do dilema entre exploração e aproveitamento fora de jogos.
5. Explique a diferença entre aprendizado on-line e um serviço disponível na internet.
6. Proponha uma tarefa que combine pré-treinamento autossupervisionado e ajuste supervisionado.
7. Para identificação de moedas, descreva uma solução por regras e uma solução aprendida. Em que condições você escolheria cada uma?

3.12 Exemplos de Aprendizado Supervisionado

O aprendizado supervisionado aparece em contextos muito diferentes, mas todos compartilham a mesma estrutura: dispomos de entradas acompanhadas por respostas conhecidas e queremos construir uma função que produza respostas adequadas para novas entradas. O domínio da aplicação muda; a formulação matemática permanece semelhante.

Para analisar uma aplicação, não basta dizer que "usaremos inteligência artificial". Precisamos identificar claramente:

- qual objeto constitui uma entrada x;
- qual resposta será usada como alvo y;
- se a tarefa é classificação ou regressão;

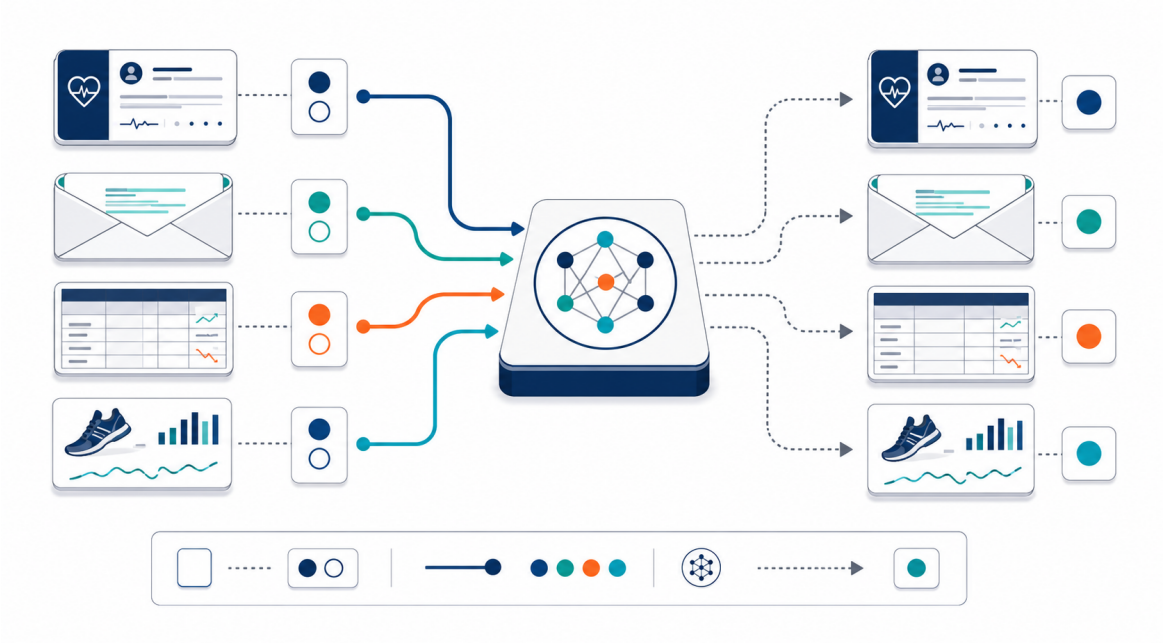


Figura 3.3: Diferentes domínios podem usar o mesmo padrão supervisionado: exemplos acompanhados de alvos treinam um modelo que fará previsões para novas entradas.

- como os exemplos serão coletados;
- qual métrica traduz a qualidade desejada;
- quais são as consequências de cada tipo de erro.

Os estudos de caso seguintes aplicam esse roteiro.

3.12.1 Diagnóstico assistido

Considere um sistema que auxilia profissionais de saúde na identificação de uma doença. Uma entrada pode combinar idade, sintomas, resultados laboratoriais e medições obtidas por equipamentos:

$$\mathbf{x} = (x_1, x_2, \dots, x_d) \in X.$$

Se o objetivo for indicar presença ou ausência de uma condição, podemos usar

$$Y = \{0, 1\}.$$

Os exemplos de treinamento seriam registros clínicos cujo diagnóstico foi estabelecido por um procedimento de referência. O modelo aprenderia uma função $h \colon X \to \{0, 1\}$ ou, preferencialmente, uma probabilidade estimada

$$h(\mathbf{x}) \approx P(Y = 1 \mid X = \mathbf{x}).$$

Nesse domínio, dois erros merecem atenção distinta. Um **falso negativo** ocorre quando uma pessoa doente é classificada como saudável; um **falso positivo** ocorre quando uma pessoa saudável é sinalizada como doente. A importância relativa depende da condição, do tratamento e do uso do sistema. Por isso, apenas a porcentagem total de acertos pode ser insuficiente.

Também é necessário separar correlação de causalidade. Um modelo pode descobrir que certo procedimento está associado à doença porque o procedimento costuma ser solicitado depois que médicos já suspeitam do diagnóstico. Usar essa variável pode produzir bom resultado retrospectivo, mas não uma ferramenta válida para detecção precoce.

> ⚠ Apoio não é decisão automática
>
> Em aplicações clínicas, o modelo deve ser validado na população e no contexto de uso. Uma previsão estatística não substitui avaliação profissional, protocolos de segurança nem responsabilidade institucional.

3.12.2 Filtragem de mensagens indesejadas

Na filtragem de e-mails, a entrada original é uma mensagem e o alvo indica `spam` ou `não spam`. Para processar o texto, extraímos atributos como frequência de termos, remetente, número de links e características do cabeçalho.

Uma representação simples é o **vetor de contagens**. Escolhido um vocabulário com d termos, a coordenada x_j registra quantas vezes o termo j aparece:

$$\mathbf{x} \in \mathbb{N}^d, \qquad y \in \{0, 1\}.$$

O classificador é treinado com mensagens previamente rotuladas. Entretanto, os padrões mudam: remetentes maliciosos adaptam sua escrita para evitar filtros, enquanto mensagens legítimas passam a usar expressões antes associadas a spam. Esse fenômeno é um exemplo de **mudança de distribuição**.

A escolha da métrica deve refletir a experiência do usuário. Bloquear um e-mail legítimo pode ser mais grave do que deixar uma propaganda chegar à caixa de entrada. Por isso, precisão, revocação e taxa de falsos positivos devem ser examinadas separadamente.

3.12.3 Avaliação de risco de crédito

Em uma análise de crédito, a entrada pode conter renda, comprometimento mensal, histórico de pagamentos e características do contrato. Há várias perguntas possíveis, e cada uma define um problema diferente:

- prever se ocorrerá inadimplência: classificação binária;
- estimar a probabilidade de inadimplência: regressão probabilística;
- estimar o prejuízo esperado: regressão;
- ordenar propostas por risco: ranqueamento.

Para classificação, podemos escrever

$$h\colon X \to \{0,1\},$$

mas a decisão de conceder crédito não precisa coincidir diretamente com $h(\mathbf{x})$. Uma política posterior pode considerar probabilidade estimada, valor solicitado, garantias e limites regulatórios.

Dados históricos exigem análise cuidadosa. Observamos o pagamento apenas para pessoas que receberam crédito no passado. Não sabemos diretamente o que teria acontecido com propostas recusadas. Além disso, decisões anteriores podem conter discriminação. Um modelo que imita esses dados pode reproduzir ou ampliar desigualdades.

3.12.4 Previsão de desempenho esportivo

Em esportes, entradas podem reunir velocidade, distância percorrida, precisão, histórico de lesões e estatísticas de partidas. O alvo deve ser definido de forma mensurável. “Ser convocado para a seleção” é um rótulo possível, mas mistura desempenho com decisões táticas, disponibilidade, posição e escolhas da comissão técnica.

Uma formulação mais específica poderia prever a quantidade de ações bem-sucedidas em uma partida futura:

$$h\colon \mathbb{R}^d \to \mathbb{R}_{\geq 0}.$$

Outra poderia estimar a probabilidade de lesão em determinado intervalo. Em ambos os casos, é importante evitar informações posteriores ao evento previsto e respeitar a privacidade dos atletas.

O exemplo evidencia que o alvo disponível nem sempre coincide com a qualidade que realmente queremos medir. Aprender a reproduzir convocações passadas não significa necessariamente aprender “mérito esportivo”.

3.12.5 Reconhecimento de imagens

Considere a classificação de imagens de algarismos manuscritos. Uma imagem em tons de cinza com 28×28 pixels pode ser transformada em um vetor com 784 intensidades:

$$\mathbf{x} \in [0,1]^{784}.$$

O rótulo pertence ao conjunto

$$Y = \{0, 1, 2, \dots, 9\}.$$

O modelo pode produzir dez pontuações, uma para cada classe. Após uma normalização, essas pontuações podem ser interpretadas como probabilidades estimadas. A classe prevista é frequentemente a de maior valor:

$$\hat{y} = \operatorname*{argmax}_{k \in Y} h_k(\mathbf{x}).$$

Mesmo aqui existem dificuldades: caligrafias variam, algumas imagens são ambíguas e dados de treinamento podem não representar dispositivos ou populações futuras.

3.12.6 Previsão de demanda

Uma empresa pode querer estimar quantas unidades de um produto serão solicitadas na próxima semana. Entradas possíveis incluem vendas anteriores, preço, promoções, feriados, região e disponibilidade em estoque. A saída é uma quantidade numérica:

$$h \colon X \to \mathbb{R}_{\geq 0}.$$

Esse é um problema de regressão. Erros para cima e para baixo podem ter custos diferentes: superestimar gera estoque parado; subestimar provoca falta do produto. Uma função de perda assimétrica pode representar essa diferença melhor do que o erro quadrático usual.

Séries temporais exigem uma divisão de treino e teste que respeite o tempo. Treinar com dados futuros para avaliar previsões do passado causaria vazamento e produziria uma estimativa irrealista de desempenho.

3.12.7 Comparando os exemplos

Aplicação	Entrada	Alvo	Tarefa	Risco relevante
Diagnóstico assistido	dados clínicos	condição presente ou ausente	classificação	falsos negativos e positivos
Filtro de spam	conteúdo e metadados	spam ou não spam	classificação	bloquear mensagem legítima
Crédito	atributos e contrato	inadimplência ou prejuízo	classificação ou regressão	viés e decisões seletivas passadas
Esporte	medições de desempenho	resultado futuro	classificação ou regressão	alvo representar mal o objetivo
Algarismos	pixels	classe de 0 a 9	classificação multiclasse	variação e ambiguidade visual
Demanda	histórico e contexto	quantidade futura	regressão	estoque excessivo ou insuficiente

A tabela mostra que a mesma técnica pode aparecer em domínios distintos, mas os riscos e critérios de sucesso não são intercambiáveis. Uma acurácia considerada adequada em recomendação de conteúdo pode ser inaceitável em uma triagem médica.

3.12.8 Da pergunta ao conjunto de dados

Depois de definir entrada e alvo, precisamos construir uma amostra. Um procedimento cuidadoso inclui:

1. determinar a população sobre a qual o sistema será usado;
2. estabelecer como entradas e rótulos serão medidos;
3. documentar dados ausentes e critérios de exclusão;
4. separar exemplos de treinamento, validação e teste;
5. escolher métricas antes de observar o resultado final;
6. verificar desempenho por grupos e condições relevantes;
7. planejar monitoramento após a implantação.

Esse processo é parte central do aprendizado supervisionado. Um conjunto grande de dados não corrige automaticamente rótulos ruins, amostragem enviesada ou uma pergunta mal definida.

3.12.9 Exercícios de fixação

1. Para um sistema que prevê evasão de estudantes, defina entrada, alvo e tipo de tarefa. Cite um possível vazamento de dados.
2. Explique por que “convocação passada” e “qualidade esportiva” não são necessariamente o mesmo alvo.
3. Em um filtro de spam, compare as consequências de falso positivo e falso negativo.
4. Transforme a previsão binária de inadimplência em um problema de regressão e explique o significado da saída.
5. Para previsão de demanda, proponha uma situação em que subestimar seja mais caro do que superestimar.
6. Escolha uma aplicação da tabela e proponha duas métricas complementares para avaliá-la.
7. Crie um novo exemplo supervisionado e descreva X, Y, a origem dos rótulos e um risco ético ou operacional.

3.13 Formalização

Até aqui descrevemos o aprendizado de maneira intuitiva. Agora reuniremos seus componentes em uma notação única. A formalização ajuda a responder perguntas precisas: o que o algoritmo recebe, o que ele escolhe, que quantidade procura minimizar e em que sentido esperamos que o resultado funcione para dados futuros.

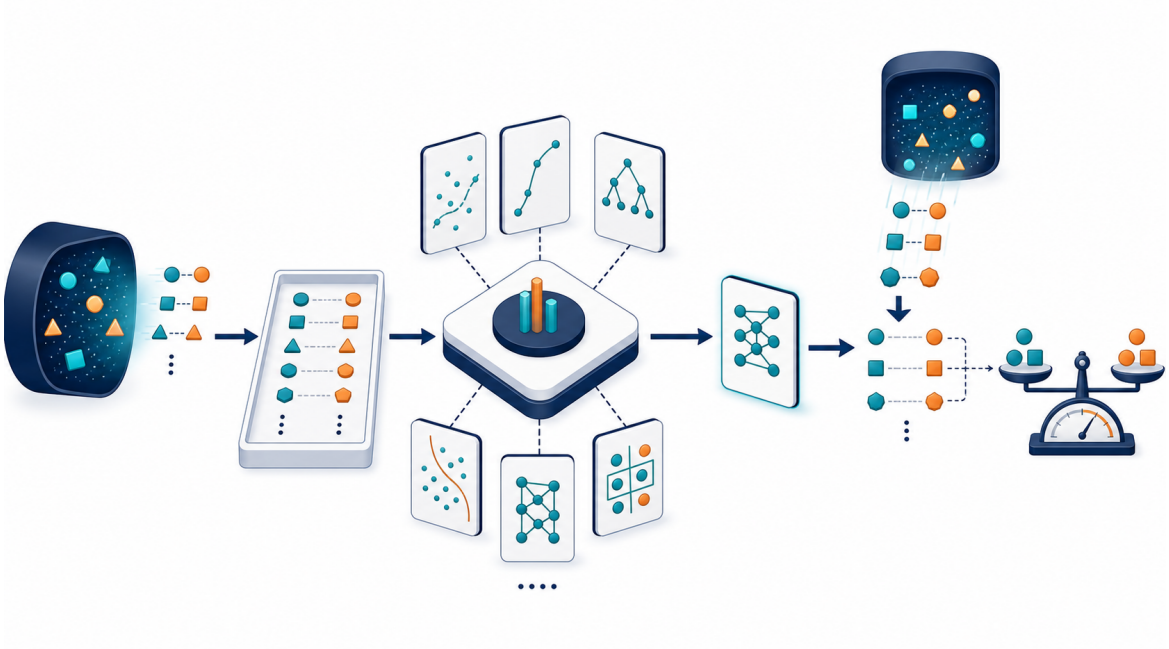


Figura 3.4: Uma distribuição gera uma amostra finita; o algoritmo seleciona uma hipótese em uma família de modelos, e a hipótese é avaliada em exemplos novos.

3.13.1 Espaço de entrada e espaço de saída

O **espaço de entrada** X contém todos os objetos que podem ser apresentados ao modelo. Dependendo da aplicação, um elemento $x \in X$ pode ser um número, um vetor, uma imagem, uma sequência de tokens ou uma estrutura mais complexa.

O **espaço de saída** Y contém as respostas possíveis. Alguns exemplos são:

$$Y = \{0, 1\}$$

para classificação binária,

$$Y = \{1, 2, \dots, K\}$$

para classificação com K classes, e

$$Y = \mathbb{R}$$

para regressão. A escolha de X e Y faz parte da formulação do problema. Dizer apenas "prever crédito" é insuficiente: precisamos decidir quais informações formam a entrada e se a saída será uma classe, uma probabilidade ou um valor monetário.

3.13.2 Processo gerador dos dados

Supomos que os pares entrada-saída são produzidos por algum processo desconhecido. Representamos esse processo por uma distribuição conjunta P sobre $X \times Y$:

$$(X, Y) \sim P.$$

Aqui, X e Y em maiúsculas representam variáveis aleatórias; x e y representam valores observados. A distribuição P descreve tanto quais entradas são frequentes quanto como as saídas se relacionam com elas.

Em um problema determinístico ideal, pode existir uma função-alvo

$$f \colon X \to Y$$

tal que $y = f(x)$ para todo exemplo. Entretanto, problemas reais frequentemente contêm ruído,

fatores não observados ou respostas genuinamente incertas. Nesse caso, uma única entrada pode estar associada a diferentes saídas com determinadas probabilidades. A descrição adequada é a distribuição condicional

$$P(Y = y \mid X = x).$$

Por exemplo, duas pessoas com os mesmos atributos registrados podem apresentar resultados clínicos diferentes porque o vetor de entrada não contém toda a informação biologicamente relevante.

> **i** A função-alvo é um modelo conceitual
>
> Falar em uma função-alvo que "sempre acerta" pode ser útil em problemas sem ruído, mas não deve ser tomado como regra geral. Em contextos probabilísticos, o alvo ideal pode ser uma probabilidade, uma média condicional ou a decisão que minimiza um custo esperado.

3.13.3 Amostra de treinamento

O algoritmo não conhece P. Ele observa uma amostra finita de N pares:

$$D = ((x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)).$$

Frequentemente assumimos que esses pares são independentes e identicamente distribuídos, abreviado por **i.i.d.**:

$$(x_i, y_i) \overset{\text{i.i.d.}}{\sim} P.$$

Essa hipótese afirma que cada exemplo foi obtido pelo mesmo processo e que a observação de um exemplo não altera as demais. Ela simplifica a teoria, mas nem sempre é realista. Dados temporais, pessoas da mesma família, quadros consecutivos de um vídeo ou múltiplas medições do mesmo paciente podem ser dependentes.

A amostra deve conter os pares (x_i, y_i), e não somente as entradas x_i, quando o problema é supervisionado. Em aprendizado não supervisionado, por outro lado, observamos apenas $(x_1, \dots, x_N)$.

3.13.4 Classe de hipóteses

Uma **hipótese** é uma função candidata que transforma entradas em previsões:

$$h\colon X \to \widehat{Y}.$$

Usamos $\widehat{Y}$ porque o formato da previsão pode diferir do alvo final. Em classificação binária, por exemplo, o alvo pode pertencer a $\{0, 1\}$, enquanto o modelo produz uma probabilidade em $[0, 1]$ que depois será convertida em classe.

A **classe de hipóteses** $\mathcal{H}$ é o conjunto de funções entre as quais o algoritmo pode escolher:

$$\mathcal{H} = \{h_\theta \mid \theta \in \Theta\}.$$

θ representa os parâmetros do modelo. Para funções afins,

$$h_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w}^{\mathsf{T}}\mathbf{x} + b,$$

temos $\theta = (\mathbf{w}, b)$. Em uma rede neural, θ reúne todas as matrizes de pesos e vetores de vieses.

A escolha de $\mathcal{H}$ impõe um **viés indutivo**: limita os padrões que o método consegue representar e indica quais soluções considera plausíveis. Sem alguma restrição, os dados finitos não determinam como o modelo deve agir em pontos ainda não observados.

3.13.5 Função de perda

Uma **função de perda** mede a qualidade de uma previsão para um único exemplo:

$$\ell\colon \widehat{Y} \times Y \to [0, \infty).$$

O primeiro argumento é a previsão $h(x)$; o segundo é o alvo y. Na regressão, uma escolha comum é a perda quadrática:

$$\ell(h(x), y) = (h(x) - y)^2.$$

Na classificação, a perda zero-um registra apenas se houve acerto:

$$\ell_{0/1}(h(x), y) = \begin{cases} 0, & h(x) = y, \\ 1, & h(x) \neq y. \end{cases}$$

Perda não é sinônimo de distância entre entradas. Ela compara uma saída prevista com uma saída

observada. Além disso, perdas diferentes expressam prioridades diferentes. Se um falso negativo custa mais do que um falso positivo, a função pode atribuir penalidades distintas a esses eventos.

3.13.6 Risco empírico

Como conhecemos apenas a amostra, calculamos a perda média nos dados observados:

$$\widehat{R}_D(h) = \frac{1}{N} \sum_{i=1}^{N} \ell(h(x_i), y_i).$$

Essa quantidade é chamada de **risco empírico**, **erro dentro da amostra** ou **erro de treinamento**. O acento em $\widehat{R}$ lembra que se trata de uma estimativa construída a partir de uma amostra finita.

Um algoritmo de minimização do risco empírico procura

$$\hat{h} \in \operatorname*{argmin}_{h \in \mathcal{H}} \widehat{R}_D(h).$$

O símbolo argmin devolve o argumento que minimiza a expressão — neste caso, uma função h — e não o menor valor da expressão.

3.13.7 Algoritmo de aprendizado

Formalmente, um **algoritmo de aprendizado** A transforma uma amostra em uma hipótese:

$$A \colon (X \times Y)^N \to \mathcal{H}, \qquad D \mapsto A(D) = \hat{h}.$$

O algoritmo pode usar descida do gradiente, uma solução algébrica, busca combinatória ou outro procedimento. A classe de hipóteses e o algoritmo não são a mesma coisa. Dois algoritmos podem procurar funções na mesma classe e produzir resultados diferentes por causa de inicialização, aproximações ou critérios de parada.

Em pseudocódigo, o fluxo geral é:

```
entrada: amostra D e classe de hipóteses H
inicialize os parâmetros do modelo
repita:
    calcule previsões nos exemplos
    avalie a perda média
    atualize os parâmetros para reduzir a perda
```

```
até o critério de parada
saída: hipótese treinada h
```

Nem todo algoritmo segue literalmente esse ciclo iterativo, mas a descrição evidencia o papel de cada componente.

3.13.8 Risco esperado e generalização

O que realmente desejamos minimizar não é apenas o erro nos exemplos conhecidos, mas o erro médio em novos pares gerados por P:

$$R(h) = \mathbb{E}_{(X,Y)\sim P}\big[\ell(h(X), Y)\big].$$

$R(h)$ é chamado de **risco esperado**, **risco verdadeiro** ou **erro fora da amostra**. Como P é desconhecida, não podemos calculá-lo exatamente. Usamos dados de validação e teste para estimá-lo.

A **lacuna de generalização** compara o risco esperado com o empírico:

$$R(h) - \widehat{R}_D(h).$$

Um modelo que memoriza a amostra pode ter $\widehat{R}_D(h)$ muito pequeno e, ainda assim, apresentar $R(h)$ grande. A teoria do aprendizado investiga condições sob as quais o risco empírico fornece informação confiável sobre o risco esperado.

3.13.9 Regularização

Para desencorajar soluções excessivamente complexas, podemos acrescentar um termo de regularização $\Omega(h)$:

$$\hat{h} \in \operatorname*{argmin}_{h\in\mathcal{H}} \left[\widehat{R}_D(h) + \lambda\Omega(h)\right].$$

$\lambda \geq 0$ controla o equilíbrio entre ajustar os dados e preferir uma solução mais simples. Em um modelo linear, $\Omega(h)$ pode penalizar pesos de grande magnitude. Regularização também pode ser implementada por parada antecipada, aumento de dados ou restrições na arquitetura.

3.13.10 Caso determinístico e caso probabilístico

No caso determinístico, supomos

$$Y = f(X),$$

e tentamos aproximar f. No caso probabilístico, modelamos

$$P(Y \mid X = x).$$

Para regressão com perda quadrática, a função que minimiza o risco esperado é a média condicional:

$$h^*(x) = \mathbb{E}[Y \mid X = x].$$

Para classificação binária, podemos estimar

$$\eta(x) = P(Y = 1 \mid X = x)$$

e escolher a classe 1 quando $\eta(x)$ ultrapassa um limiar. O limiar não precisa ser $0{,}5$ quando os custos dos erros são diferentes.

Essa perspectiva é mais precisa do que imaginar que toda frequência observada é uma propriedade fixa da entrada. A amostra fornece evidência limitada sobre uma distribuição; o modelo utiliza hipóteses e regularidades para generalizar além das ocorrências registradas.

3.13.11 Exemplo completo

Considere uma regressão que estima o tempo de execução de um programa a partir do tamanho da entrada:

$$X = \mathbb{R}_{>0}, \qquad Y = \mathbb{R}_{>0}.$$

A amostra contém medições

$$D = \{(n_i, t_i)\}_{i=1}^{N}.$$

Escolhemos a classe de funções afins

$$\mathcal{H} = \{h_{a,b}(n) = an + b \mid a, b \in \mathbb{R}\}$$

e a perda quadrática. O treinamento procura

$$(\hat{a}, \hat{b}) \in \operatorname*{argmin}_{a,b} \frac{1}{N} \sum_{i=1}^{N} (an_i + b - t_i)^2.$$

A hipótese treinada pode prever tempos para tamanhos não medidos. Entretanto, se o algoritmo real tiver comportamento quadrático ou se a máquina usada em produção for diferente, a classe linear ou a distribuição de treinamento pode ser inadequada.

3.13.12 Mapa da notação

Símbolo	Significado
X	espaço de entradas
Y	espaço de saídas ou alvos
P	distribuição geradora dos pares (X, Y)
D	amostra finita de treinamento
$\mathcal{H}$	classe de hipóteses
h	uma hipótese ou modelo candidato
θ	parâmetros que determinam uma hipótese
ℓ	perda em um exemplo
$\widehat{R}_D$	risco empírico calculado na amostra
R	risco esperado sob a distribuição P
A	algoritmo que transforma dados em hipótese

3.13.13 Exercícios de fixação

1. Para a classificação de spam, identifique X, Y, D, uma classe $\mathcal{H}$ e uma possível função de perda.
2. Explique por que ℓ costuma ter domínio $\widehat{Y} \times Y$, e não $X \times X$.
3. Qual é a diferença entre $\min_h \widehat{R}_D(h)$ e $\operatorname{argmin}_h \widehat{R}_D(h)$?
4. Dê um exemplo em que a hipótese i.i.d. não seja razoável.
5. Um modelo tem erro de treinamento igual a zero. Isso prova que seu risco esperado é zero? Justifique.

6. Explique o papel de λ em um objetivo regularizado.
7. No exemplo do tempo de execução, proponha uma classe de hipóteses adequada a um algoritmo cujo custo cresce como n^2.

3.14 Formalização dos Exemplos

A notação abstrata da seção anterior torna-se mais clara quando a aplicamos a problemas concretos. Em cada exemplo precisamos definir não apenas X, Y e a amostra D, mas também a representação dos objetos, a classe de hipóteses, a perda e a distribuição na qual o sistema será usado.

Uma formalização é uma escolha de modelagem. O mesmo problema verbal pode originar modelos matemáticos diferentes. “Avaliar um pedido de crédito”, por exemplo, pode significar prever inadimplência, estimar prejuízo ou ordenar propostas por risco. Cada interpretação altera o espaço de saída e a função de perda.

3.14.1 Diagnóstico assistido como classificação binária

Suponha que desejamos estimar a presença de uma condição específica. Escolhemos d atributos clínicos e representamos uma pessoa por

$$\mathbf{x} = (x_1, x_2, \dots, x_d) \in X \subseteq \mathbb{R}^d.$$

As coordenadas podem incluir idade, medições laboratoriais e indicadores de sintomas. O espaço de saídas é

$$Y = \{0, 1\},$$

com $y = 1$ para condição presente e $y = 0$ para ausente. A amostra é

$$D_{\text{clin}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N},$$

obtida de registros para os quais existe um diagnóstico de referência.

Em vez de produzir diretamente uma classe, uma hipótese pode estimar

$$h_\theta(\mathbf{x}) \in [0, 1],$$

interpretado como probabilidade da classe positiva. Uma regra de decisão converte essa pontuação em classe:

$$\widehat{y} = \begin{cases} 1, & h_\theta(\mathbf{x}) \geq \tau, \\ 0, & h_\theta(\mathbf{x}) < \tau. \end{cases}$$

O limiar τ deve refletir os custos de falsos negativos e falsos positivos. Uma perda ponderada pode atribuir custo c_{FN} ao primeiro e c_{FP} ao segundo.

Há ainda uma questão sobre P: a população de treinamento corresponde à população futura? Um modelo treinado em um único hospital pode não generalizar para outros equipamentos, protocolos ou perfis demográficos.

3.14.2 Spam como classificação de texto

Uma mensagem é um objeto textual, mas muitos algoritmos recebem vetores numéricos. Fixamos um vocabulário

$$V = \{w_1, w_2, \ldots, w_d\}$$

e usamos uma função de representação

$$\phi\colon \text{mensagens} \to \mathbb{R}^d.$$

A coordenada de $\phi(m)$ pode registrar a contagem ou a frequência de um termo. Assim,

$$X = \phi(\text{mensagens}) \subseteq \mathbb{R}^d, \qquad Y = \{0, 1\}.$$

A amostra rotulada é

$$D_{\text{spam}} = \{(\phi(m_i), y_i)\}_{i=1}^{N}.$$

Uma classe simples de hipóteses é formada por classificadores lineares:

$$\mathcal{H} = \left\{ h_{\mathbf{w},b}(\mathbf{x}) = \mathbb{1}[\mathbf{w}^{\mathsf{T}}\mathbf{x} + b \geq 0] \right\}.$$

$\mathbb{1}[C]$ vale 1 quando a condição C é verdadeira e 0 caso contrário. Cada peso w_j mede como o termo correspondente influencia a pontuação.

A dimensão d pode ser muito grande, pois um vocabulário contém milhares de termos. Entretanto, cada mensagem utiliza apenas uma pequena fração deles; seus vetores são **esparsos**. Estruturas de dados esparsas armazenam somente coordenadas não nulas e economizam memória.

3.14.3 Risco de crédito: classe ou probabilidade

Representamos uma proposta por um vetor de atributos:

$$\mathbf{x} = (\text{renda}, \text{dívida}, \text{prazo}, ...) \in X \subseteq \mathbb{R}^d.$$

Para prever inadimplência em determinado horizonte,

$$Y = \{0, 1\}$$

e

$$D_{\text{crédito}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}.$$

Se queremos uma probabilidade de inadimplência, usamos hipóteses $h \colon X \to [0, 1]$. A decisão de conceder crédito pode considerar o custo esperado:

$$\text{custo}(\mathbf{x}) = h(\mathbf{x})C_{\text{inadimplência}} + (1 - h(\mathbf{x}))C_{\text{regular}}.$$

Essa separação é importante: **previsão** estima um resultado; **decisão** combina a previsão com custos, regras e restrições.

A amostra histórica possui um viés de seleção: geralmente observamos inadimplência apenas para propostas aprovadas. Formalmente, a distribuição observada pode ser

$$P(X, Y \mid \text{crédito aprovado}),$$

enquanto o sistema será aplicado a $P(X, Y \mid \text{nova proposta})$. Ignorar essa diferença compromete a validade da avaliação.

3.14.4 Desempenho esportivo: escolhendo um alvo mensurável

Considere um vetor de características de um atleta,

$$\mathbf{x} = (\text{velocidade}, \text{precisão}, \text{resistência}, ...) \in \mathbb{R}^d.$$

Se o alvo for "convocado ou não convocado", então $Y = \{0, 1\}$ e podemos treinar com decisões passadas. Contudo, esse rótulo representa escolhas de comissões técnicas, não uma propriedade física objetiva.

Uma alternativa é prever uma estatística futura, como número de ações bem-sucedidas por partida:

$$Y = \mathbb{R}_{\geq 0}.$$

Nesse caso temos regressão e podemos usar perda absoluta,

$$\ell(h(\mathbf{x}), y) = |h(\mathbf{x}) - y|,$$

que penaliza proporcionalmente a magnitude do erro e é menos sensível a valores extremos do que a perda quadrática.

Também precisamos respeitar a ordem temporal. Atributos de uma temporada posterior não podem ser usados para prever um resultado anterior. Uma divisão aleatória ingênua pode colocar medições do mesmo atleta e período tanto no treino quanto no teste, superestimando a generalização.

3.14.5 Reconhecimento de algarismos

Uma imagem com 28×28 pixels em tons de cinza pode ser vetorizada:

$$X = [0, 1]^{784}.$$

O alvo é uma entre dez classes:

$$Y = \{0, 1, 2, ..., 9\}.$$

A amostra é

$$D_{\text{dígitos}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}.$$

Uma hipótese pode produzir um vetor de probabilidades

$$h(\mathbf{x}) = (p_0, p_1, \dots, p_9), \qquad p_k \geq 0, \quad \sum_{k=0}^{9} p_k = 1.$$

Nesse caso, $\widehat{Y}$ é o simplex de probabilidades, enquanto Y é o conjunto de classes. A classe prevista é $\operatorname{argmax}_k p_k$. Essa distinção explica por que previsão e alvo nem sempre pertencem ao mesmo conjunto.

3.14.6 Previsão de demanda como regressão

Para prever a demanda de um produto na semana seguinte, a entrada pode reunir vendas recentes, preço, promoção e calendário:

$$\mathbf{x}_t \in X \subseteq \mathbb{R}^d, \qquad y_t \in Y = \mathbb{R}_{\geq 0}.$$

A amostra temporal é

$$D_{\text{demanda}} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}.$$

Os índices t lembram que os exemplos não são necessariamente independentes. Uma classe linear pode servir como ponto de partida:

$$h_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w}^{\mathsf{T}}\mathbf{x} + b.$$

Como a saída real não pode ser negativa, podemos restringir a previsão usando $\max\{0, h(\mathbf{x})\}$ ou escolher uma família de modelos com saídas não negativas.

3.14.7 Visão comparativa

Problema	X	Y	Previsão $\widehat{Y}$	Observação crítica
Diagnóstico	atributos clínicos	$\{0, 1\}$	probabilidade em $[0, 1]$	custos dos erros diferem

Problema	X	Y	Previsão $\widehat{Y}$	Observação crítica
Spam	vetores textuais esparsos	$\{0,1\}$	classe ou probabilidade	vocabulário muda com o tempo
Crédito	atributos financeiros	$\{0,1\}$	probabilidade	dados observados são seletivos
Esporte	medições de desempenho	classe ou valor real	depende do alvo	rótulo pode não representar mérito
Algarismos	$[0,1]^{784}$	dez classes	vetor de dez probabilidades	variação visual
Demanda	histórico e contexto	$\mathbb{R}_{\geq 0}$	valor real	dependência temporal

O quadro evidencia que escrever $X = \mathbb{R}^d$ é apenas o começo. É preciso explicar o significado das coordenadas, a origem de Y, a população representada por P e o custo codificado por ℓ.

3.14.8 Dimensão e geometria

Quando $X \subseteq \mathbb{R}^2$, podemos desenhar cada exemplo como um ponto no plano. Em três dimensões ainda há uma visualização direta. Para $d > 3$, a interpretação geométrica continua válida, embora não possamos desenhar todo o espaço de forma literal.

Um e-mail representado por $50\,000$ termos é um ponto em $\mathbb{R}^{50\,000}$. A dimensão elevada não impede o uso de conceitos como distância, produto interno, hiperplano e região de decisão. Grande parte da aprendizagem de máquina consiste em estender a intuição geométrica para esses espaços.

Entretanto, atributos em escalas diferentes podem distorcer a geometria. Se renda é medida em milhares e uma proporção varia entre 0 e 1, distâncias e produtos internos podem ser dominados pela renda. Normalização e padronização são etapas importantes de representação.

3.14.9 Checklist de formalização

Antes de treinar um modelo, devemos conseguir preencher:

1. X: quais entradas são permitidas e como são representadas?

2. Y: qual é o alvo observado?
3. $\widehat{Y}$: qual objeto o modelo realmente produz?
4. D: como a amostra foi coletada e rotulada?
5. P: qual população ou processo futuro queremos representar?
6. $\mathcal{H}$: quais funções o algoritmo pode escolher?
7. ℓ: quais erros recebem maior penalidade?
8. A: como uma hipótese será escolhida?
9. avaliação: como estimaremos desempenho fora da amostra?

Se algum item não puder ser respondido, o problema ainda não está suficientemente definido para que a escolha de um algoritmo seja a principal preocupação.

3.14.10 Exercícios de fixação

1. Formalize um sistema de previsão de evasão escolar especificando X, Y, D, $\widehat{Y}$ e uma perda.
2. No reconhecimento de algarismos, por que Y e $\widehat{Y}$ podem ser conjuntos diferentes?
3. Explique o viés de seleção presente nos dados históricos de crédito.
4. Escreva uma hipótese linear para spam com três atributos e interprete cada peso.
5. Proponha dois alvos diferentes para análise esportiva e diga qual tipo de tarefa cada um produz.
6. Para demanda, explique por que uma divisão temporal é preferível a embaralhar todos os exemplos.
7. Escolha um dos casos e descreva uma mudança em P que poderia degradar o modelo após a implantação.

3.15 Pontuação

Muitos classificadores começam calculando uma **pontuação** numérica. A entrada é um vetor de atributos

$$\mathbf{x} = (x_1, x_2, \dots, x_d) \in \mathbb{R}^d,$$

e cada coordenada recebe um peso que expressa sua contribuição para a decisão. A pontuação linear é

$$s(\mathbf{x}) = \mathbf{w}^{\mathsf{T}}\mathbf{x} + b = w_1x_1 + w_2x_2 + \cdots + w_dx_d + b.$$

$\mathbf{w} = (w_1, \ldots, w_d)$ é o vetor de pesos e b é o **viés** (*bias*). O nome "viés" aqui designa um parâmetro do modelo e não deve ser confundido com viés estatístico ou discriminação nos dados.

Uma regra binária simples atribui a classe positiva quando a pontuação é não negativa:

$$h_{\mathbf{w},b}(\mathbf{x}) = \begin{cases} +1, & s(\mathbf{x}) \geq 0, \\ -1, & s(\mathbf{x}) < 0. \end{cases}$$

Essa função também pode ser escrita como

$$h_{\mathbf{w},b}(\mathbf{x}) = \operatorname{sign}(\mathbf{w}^{\mathsf{T}}\mathbf{x} + b).$$

3.15.1 Pesos e atributos

O sinal de um peso indica a direção de sua influência, mantendo os demais atributos fixos:

- $w_j > 0$: aumentar x_j aumenta a pontuação;
- $w_j < 0$: aumentar x_j diminui a pontuação;
- $w_j = 0$: o atributo não afeta diretamente a pontuação.

A magnitude $|w_j|$ indica a sensibilidade da pontuação à coordenada x_j, mas sua interpretação depende da escala. Um peso de $0{,}1$ aplicado a uma variável medida em milhares pode contribuir mais do que um peso 10 aplicado a uma proporção entre 0 e 1. Portanto, comparar pesos só é informativo quando conhecemos as unidades ou padronizamos os atributos.

O termo b desloca a fronteira de decisão. Podemos interpretar a regra usando um limiar τ:

$$\mathbf{w}^{\mathsf{T}}\mathbf{x} \geq \tau.$$

As duas formas são equivalentes quando $b = -\tau$.

3.15.2 Exemplo numérico

Considere três perfis fictícios descritos por dois atributos já normalizados:

- x_1: medida de desempenho técnico;
- x_2: medida de consistência.

Usaremos a pontuação

$$s(\mathbf{x}) = 0{,}6x_1 + 0{,}8x_2 - 5.$$

Os pesos mostram que, nessa regra específica, uma unidade adicional de consistência contribui $0{,}8$, enquanto uma unidade de desempenho técnico contribui $0{,}6$.

Perfil	x_1	x_2	Pontuação $s(\mathbf{x})$	Classe
A	4	5	$0{,}6(4) + 0{,}8(5) - 5 = 1{,}4$	$+1$
B	7	2	$0{,}6(7) + 0{,}8(2) - 5 = 0{,}8$	$+1$
C	4	2	$0{,}6(4) + 0{,}8(2) - 5 = -1{,}0$	-1

O cálculo é determinístico: fixados $\mathbf{w}$ e b, qualquer vetor recebe uma pontuação e uma classe. O aprendizado consiste em escolher esses parâmetros a partir dos exemplos, e não em calcular a soma depois que os parâmetros já são conhecidos.

3.15.3 Interpretação geométrica

Os pontos para os quais a pontuação é exatamente zero formam a **fronteira de decisão**:

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

Em duas dimensões, essa fronteira é uma reta. Em três dimensões, é um plano. Em d dimensões, é um **hiperplano**. De um lado estão os pontos com pontuação positiva; do outro, os pontos com pontuação negativa.

O vetor $\mathbf{w}$ é perpendicular ao hiperplano e aponta para o lado em que a pontuação aumenta. Se multiplicarmos $\mathbf{w}$ e b pelo mesmo número positivo, a fronteira e as classes não mudam, embora os valores das pontuações sejam escalados.

No exemplo,

$$0{,}6x_1 + 0{,}8x_2 - 5 = 0$$

pode ser isolada como

$$x_2 = 6{,}25 - 0{,}75x_1.$$

Essa equação permite desenhar a reta e verificar visualmente a região de cada perfil.

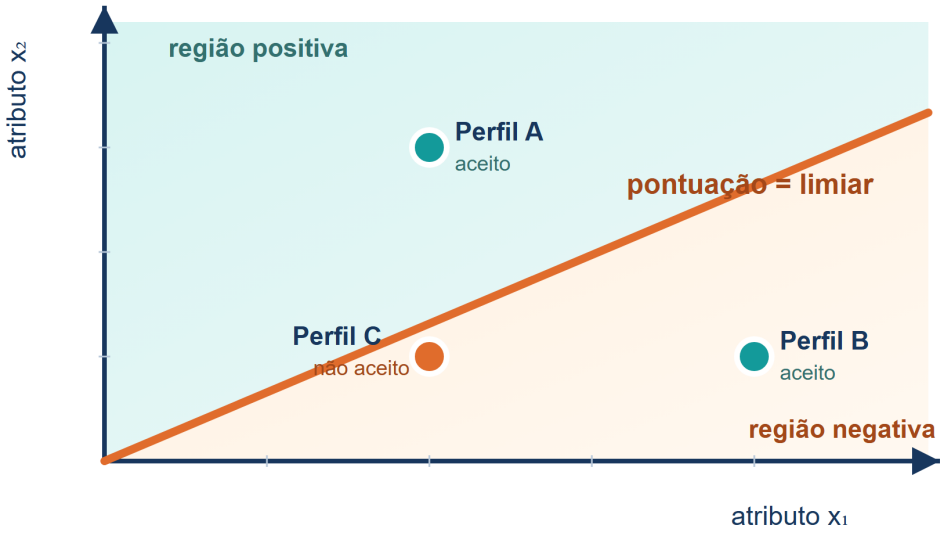


Figura 3.5: Uma pontuação linear divide o espaço de atributos em duas regiões por meio de uma reta de decisão.

3.15.4 Distância assinada à fronteira

A pontuação bruta depende da escala dos pesos. Dividindo-a pela norma de $\mathbf{w}$, obtemos a distância assinada do ponto ao hiperplano:

$$d(\mathbf{x}) = \frac{\mathbf{w}^{\mathsf{T}}\mathbf{x} + b}{\|\mathbf{w}\|_2}.$$

O sinal informa o lado da fronteira; o módulo fornece a distância perpendicular. No exemplo,

$$\|\mathbf{w}\|_2 = \sqrt{0{,}6^2 + 0{,}8^2} = 1,$$

portanto a pontuação coincide numericamente com a distância assinada.

Pontos próximos da fronteira são mais sensíveis a pequenas alterações dos atributos ou dos parâmetros. Entretanto, uma distância grande não é automaticamente uma probabilidade alta; ela apenas expressa separação geométrica no modelo linear.

3.15.5 Pontuação não é probabilidade

$s(\mathbf{x})$ pode assumir qualquer valor real e não precisa estar entre 0 e 1. Para obter uma quantidade interpretável como probabilidade, podemos aplicar uma função logística:

$$p(\mathbf{x}) = \sigma(s(\mathbf{x})) = \frac{1}{1 + e^{-s(\mathbf{x})}}.$$

Essa transformação preserva a ordem das pontuações: se $s(\mathbf{x}_a) > s(\mathbf{x}_b)$, então $p(\mathbf{x}_a) > p(\mathbf{x}_b)$. Contudo, interpretar p como probabilidade confiável exige treinamento e calibração apropriados.

Uma pontuação também pode ser usada apenas para **ranquear** elementos. Um mecanismo de busca, por exemplo, ordena documentos por relevância sem necessariamente converter cada valor em uma classe.

3.15.6 Aprendendo os pesos

Dada uma amostra

$$D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}, \qquad y_i \in \{-1, +1\},$$

queremos escolher $\mathbf{w}$ e b para produzir boas previsões. Uma condição de classificação correta é

$$y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) > 0.$$

Quando os dados são **linearmente separáveis**, existe ao menos um hiperplano que classifica corretamente todos os exemplos de treinamento. Isso não garante que a mesma fronteira terá bom desempenho em dados futuros, nem que todos os conjuntos reais sejam separáveis.

O perceptron, estudado adiante, ajusta os pesos quando encontra um exemplo classificado incorretamente. A regressão logística utiliza uma perda diferenciável e produz pontuações transformadas pela função logística. Máquinas de vetores de suporte procuram uma fronteira com margem ampla. Esses métodos compartilham a estrutura linear, mas possuem objetivos de treinamento diferentes.

3.15.7 Código de uma pontuação linear

Uma implementação direta em Python é:

```
def pontuacao(x, w, b):
    return sum(x_j * w_j for x_j, w_j in zip(x, w)) + b


def classificar(x, w, b):
```

```
    return 1 if pontuacao(x, w, b) >= 0 else -1


w = [0.6, 0.8]
b = -5.0

print(classificar([4, 5], w, b))  # 1
print(classificar([4, 2], w, b))  # -1
```

Em bibliotecas numéricas, o produto interno costuma ser calculado de forma vetorizada. Para uma matriz $X \in \mathbb{R}^{N \times d}$ contendo N exemplos nas linhas, todas as pontuações são

$$\mathbf{s} = X\mathbf{w} + b\mathbf{1}.$$

Essa forma evita um laço explícito por exemplo e aproveita implementações eficientes de álgebra linear.

3.15.8 Limitações e cuidados

Uma fronteira linear é simples e interpretável, mas não representa qualquer padrão. Se a classe positiva formar um círculo cercado pela classe negativa, nenhuma reta separará os grupos perfeitamente no espaço original. Podemos então criar novos atributos, aplicar uma transformação ou escolher uma classe de hipóteses não linear.

Além disso, um peso aprendido representa associação dentro dos dados e do modelo; não prova causalidade. Atributos sensíveis ou variáveis que funcionam como substitutas podem produzir decisões injustas. Em aplicações que afetam pessoas, é necessário avaliar desempenho entre grupos, documentar limitações e permitir revisão adequada.

> **!** Separar a amostra não basta
>
> Encontrar pesos que classificam perfeitamente os exemplos de treinamento demonstra ajuste à amostra, não generalização. A fronteira precisa ser avaliada em dados independentes e representativos da população de uso.

3.15.9 Exercícios de fixação

1. Calcule a pontuação dos três perfis usando $s(\mathbf{x}) = x_1 + x_2 - 6$ e determine suas classes.
2. Reescreva a regra $2x_1 - 3x_2 \geq 4$ na forma $\mathbf{w}^\mathsf{T}\mathbf{x} + b \geq 0$.

3. Para $\mathbf{w} = (3, 4)$ e $b = -10$, calcule $\|\mathbf{w}\|_2$ e a distância assinada do ponto $(2, 2)$ à fronteira.
4. Explique por que multiplicar $\mathbf{w}$ e b por 5 não altera as classes, mas altera as pontuações.
5. Dê um exemplo de conjunto de pontos em $\mathbb{R}^2$ que não seja linearmente separável.
6. Qual é a diferença entre uma pontuação, uma classe e uma probabilidade calibrada?
7. Implemente a versão vetorizada da pontuação para vários exemplos usando uma biblioteca numérica.

3.16 Exercícios de múltipla escolha

As questões a seguir retomam os principais conceitos do capítulo. Em cada uma, assinale apenas uma alternativa. Procure justificar sua escolha antes de consultar o gabarito.

1. Se $A = \{1, 2, 3\}$ e $B = \{3, 4\}$, qual é o conjunto $A \cap B$?

   a. $\{1, 2, 4\}$

   b. $\{3\}$

   c. $\{1, 2, 3, 4\}$

   d. $\emptyset$

2. Uma função $f : A \to B$ é injetiva quando:

   a. todo elemento de B possui exatamente uma pré-imagem.

   b. elementos distintos de A sempre produzem imagens distintas.

   c. sua imagem é necessariamente igual a A.

   d. seu domínio contém apenas números reais.

3. Para que uma função $f : A \to B$ possua uma inversa $f^{-1} : B \to A$, ela deve ser:

   a. constante.

   b. apenas injetiva.

   c. bijetiva.

   d. apenas sobrejetiva.

4. Qual é a representação binária do número decimal 13?

   a. 1011_2

   b. 1100_2

   c. 1101_2

   d. 1110_2

5. Um byte contém:

   a. 2 bits.

   b. 4 bits.

   c. 8 bits.

   d. 16 bits.

6. Em complemento de dois com 8 bits, o intervalo de inteiros representáveis é:

   a. de -127 a 127.

   b. de -128 a 127.

   c. de -128 a 128.

   d. de 0 a 255.

7. Em um problema supervisionado, o conjunto de treinamento contém:

   a. somente entradas sem qualquer informação de saída.

   b. pares de entrada e saída-alvo.

   c. apenas regras escritas por um programador.

   d. somente ações e recompensas futuras.

8. Qual alternativa descreve uma tarefa de regressão?

   a. Decidir se uma mensagem é spam.

b. Agrupar clientes sem rótulos prévios.

c. Prever o consumo mensal de energia em kWh.

d. Reconhecer qual algarismo aparece em uma imagem.

9. Na pontuação linear $s(\mathbf{x}) = \mathbf{w}^{\mathsf{T}}\mathbf{x} + b$, o vetor $\mathbf{w}$ determina principalmente:

a. a orientação da fronteira de decisão.

b. o número de exemplos da amostra.

c. a base numérica usada pelo computador.

d. a probabilidade exata de qualquer classe.

10. Obter erro zero nos dados de treinamento significa necessariamente que:

a. o modelo terá erro zero em produção.

b. o problema é não supervisionado.

c. a hipótese ajustou a amostra, mas ainda precisa ter sua generalização avaliada.

d. todos os pesos do modelo são iguais a zero.

Gabarito comentado

1. **b.** A interseção contém somente os elementos presentes nos dois conjuntos; neste caso, apenas 3.
2. **b.** Injetividade impede que duas entradas distintas sejam associadas à mesma saída.
3. **c.** A inversa definida em todo o contradomínio exige injetividade e sobrejetividade, isto é, bijetividade.
4. **c.** $13 = 8 + 4 + 1$, portanto seus bits são 1101_2.
5. **c.** Por definição, um byte possui 8 bits.
6. **b.** Com n bits em complemento de dois, o intervalo é $[-2^{n-1}, 2^{n-1} - 1]$.
7. **b.** O aprendizado supervisionado utiliza exemplos nos quais a saída-alvo acompanha a entrada.
8. **c.** O consumo é uma quantidade numérica contínua; as alternativas a e d são classifi-

cações, e b é agrupamento.

9. **a.** **w** é normal ao hiperplano e, por isso, controla sua orientação; b controla o deslocamento.
10. **c.** Erro de treinamento mede ajuste aos exemplos observados, não garante desempenho em dados novos.

Capítulo 4

O que é aprendizado?

No capítulo anterior, formalizamos um método de aprendizado como um algoritmo que recebe uma amostra e devolve uma hipótese. Essa descrição esclarece **como** o treinamento é organizado, mas ainda deixa aberta a pergunta central deste capítulo: quando podemos afirmar que houve aprendizado?

Acertar os exemplos conhecidos não é suficiente. Um programa poderia armazenar cada par da amostra e devolver a resposta memorizada quando encontrasse exatamente a mesma entrada. Seu erro de treinamento seria zero, mas ele não saberia como responder a uma entrada nova. Aprender exige alguma forma de **generalização**.

Neste capítulo, diremos que um método aprende quando utiliza uma amostra finita para produzir uma hipótese cujo desempenho permanece bom em exemplos ainda não observados, provenientes do processo de interesse. Essa ideia envolve três elementos:

1. uma medida de qualidade ou erro;
2. uma afirmação sobre dados fora da amostra;
3. uma garantia probabilística, pois diferentes amostras podem levar a hipóteses diferentes.

Retomando a notação.

No aprendizado supervisionado, consideramos:

- um espaço de entradas X;
- um espaço de saídas Y;
- uma distribuição desconhecida P sobre $X \times Y$;
- uma amostra de treinamento

$$D = \{(x_i, y_i)\}_{i=1}^{N};$$

- uma classe de hipóteses $\mathcal{H}$;
- uma função de perda ℓ;
- um algoritmo A que produz

$$\hat{h} = A(D) \in \mathcal{H}.$$

Em um modelo determinístico, podemos supor $y = f(x)$ para alguma função-alvo f. Em problemas com ruído, trabalhamos com a distribuição condicional $P(Y \mid X = x)$. As ideias de erro e generalização valem nos dois casos.

O risco empírico mede o desempenho nos exemplos observados:

$$\widehat{R}_D(h) = \frac{1}{N} \sum_{i=1}^{N} \ell(h(x_i), y_i).$$

O risco esperado mede o desempenho médio no processo gerador:

$$R(h) = \mathbb{E}_{(X,Y)\sim P}\big[\ell(h(X), Y)\big].$$

O primeiro pode ser calculado a partir do treinamento; o segundo é o que realmente nos interessa, mas depende de uma distribuição desconhecida.

Aprendizado como generalização.

Uma hipótese generaliza quando seu erro em dados novos é próximo do erro observado na amostra. Intuitivamente, desejamos que

$$R(\hat{h}) \approx \widehat{R}_D(\hat{h}).$$

Essa aproximação não ocorre automaticamente. O próprio algoritmo escolheu $\hat{h}$ depois de observar D, portanto a hipótese pode ter explorado detalhes acidentais da amostra. Quanto mais flexível for a classe $\mathcal{H}$, maior sua capacidade de ajustar padrões reais — e também ruído.

Dois tipos de falha ajudam a organizar essa discussão:

- **subajuste** (*underfitting*): a classe ou o treinamento é simples demais para capturar o padrão relevante; os erros de treino e de teste tendem a ser altos;
- **sobreajuste** (*overfitting*): a hipótese se adapta excessivamente à amostra; o erro de treino é baixo, mas o erro em dados novos é alto.

O objetivo não é obter o menor erro de treinamento a qualquer custo. Queremos equilibrar ajuste e capacidade de generalização.

> **!** O conjunto de teste não participa do aprendizado
>
> Se usamos repetidamente o desempenho de teste para escolher atributos, hiperparâmetros ou modelos, o teste deixa de representar dados não vistos. Essas escolhas devem ser feitas com dados de treinamento e validação; o teste é reservado para a avaliação final.

O papel da probabilidade.

A amostra é aleatória. Duas equipes podem coletar N exemplos da mesma população e obter conjuntos diferentes. Consequentemente, o algoritmo pode produzir hipóteses diferentes.

Uma garantia de aprendizado costuma ter a forma:

$$P_D\big(R(A(D)) \leq \varepsilon\big) \geq 1 - \delta,$$

em que:

- ε é a tolerância de erro;
- δ é a probabilidade máxima de a garantia falhar;
- a probabilidade é tomada sobre a escolha aleatória da amostra D.

Lemos a expressão assim: com probabilidade de pelo menos $1 - \delta$, o algoritmo devolve uma hipótese cujo risco não ultrapassa ε. Em formulações mais gerais, comparamos $R(A(D))$ ao melhor risco alcançável dentro de $\mathcal{H}$.

Essa é a origem da expressão **provavelmente aproximadamente correto**: "provavelmente" refere-se à confiança $1 - \delta$ e "aproximadamente" à tolerância ε.

O que determina a dificuldade.

A quantidade de dados necessária depende de vários fatores:

- a precisão desejada, controlada por ε;
- a confiança desejada, controlada por δ;
- a complexidade da classe de hipóteses $\mathcal{H}$;
- o nível de ruído;
- a qualidade e representatividade da amostra;
- a função de perda e o tipo de garantia procurada.

Exigir erro menor ou confiança maior normalmente requer mais exemplos. Classes muito

expressivas também precisam de maior quantidade de dados para que possamos distinguir padrões generalizáveis de ajustes acidentais.

Três perguntas do capítulo.

As próximas seções desenvolverão três perguntas complementares.

1. **Como medir a discrepância?** Precisamos definir erro, perda e noção de proximidade entre uma hipótese e o comportamento desejado.
2. **Quando o erro observado é confiável?** Estudaremos garantias probabilísticas que relacionam amostra e população.
3. **Como estimar o desempenho na prática?** Discutiremos divisões entre treino, validação e teste.

Essas perguntas conectam teoria e engenharia. A teoria fornece condições e limites; o protocolo experimental produz evidência sobre um modelo concreto.

Exemplo motivador.

Suponha que um classificador de spam acerte 990 de $1\,000$ mensagens de treinamento. A taxa de erro empírico é

$$\widehat{R}_D(h) = \frac{10}{1\,000} = 0{,}01.$$

Ainda não podemos concluir que seu erro futuro será 1%. Talvez as mensagens tenham sido usadas para escolher palavras-chave específicas, talvez a amostra contenha mensagens muito semelhantes ou talvez o padrão de spam tenha mudado.

Se avaliarmos o classificador uma única vez em $2\,000$ mensagens de teste independentes e encontrarmos 60 erros, a estimativa de teste será

$$\widehat{R}_{\text{teste}}(h) = \frac{60}{2\,000} = 0{,}03.$$

A diferença entre 1% no treino e 3% no teste sugere uma lacuna de generalização. Para interpretar a incerteza dessa estimativa, precisaremos de ferramentas probabilísticas.

Exercícios de preparação.

1. Explique por que memorizar uma amostra não é suficiente para caracterizar aprendizado.
2. Diferencie risco empírico e risco esperado.
3. Dê um exemplo de subajuste e um de sobreajuste.
4. Interprete ε e δ em uma garantia probabilística.

5. Por que avaliar repetidamente no conjunto de teste compromete sua função?
6. No exemplo de spam, cite duas causas possíveis para a diferença entre os erros de treino e teste.

4.1 2.1 Erro

Aprender exige comparar previsões com resultados observados. Essa comparação é feita por uma **função de perda**, que traduz a qualidade de uma previsão em um número. Quanto menor a perda, melhor a previsão segundo o critério escolhido.

Não existe uma única noção de erro apropriada para todos os problemas. Errar uma classe é diferente de errar uma quantidade contínua; em algumas aplicações, certos erros têm consequências muito mais graves do que outros. A função de perda faz parte da formulação e deve refletir o objetivo real do sistema.

4.1.1 Perda por exemplo, risco e métrica

A **perda por exemplo** compara uma previsão $\hat{y}$ com o alvo y:

$$\ell \colon \widehat{Y} \times Y \to [0, \infty).$$

O **risco empírico** agrega as perdas na amostra:

$$\widehat{R}_D(h) = \frac{1}{N} \sum_{i=1}^{N} \ell(h(x_i), y_i).$$

Uma **métrica de avaliação** resume algum aspecto do comportamento do modelo, como acurácia, precisão, revocação ou erro absoluto médio. Algumas métricas podem ser usadas como objetivo de treinamento; outras são difíceis de otimizar diretamente e servem principalmente para avaliação.

O algoritmo pode minimizar um objetivo com risco empírico e regularização:

$$J(\theta) = \widehat{R}_D(h_\theta) + \lambda \Omega(\theta).$$

Usaremos **perda** para um exemplo, **risco** para uma média de perdas e **objetivo** para a quantidade efetivamente otimizada.

4.1.2 Classificação: perda zero-um

Em classificação, a perda zero-um vale 0 para um acerto e 1 para um erro:

$$\ell_{0/1}(\hat{y}, y) = \mathbb{1}[\hat{y} \neq y].$$

Seu risco empírico é a taxa de classificações incorretas:

$$\widehat{R}_{0/1}(h) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}[h(x_i) \neq y_i].$$

Se um classificador erra 18 de 300 exemplos, então $\widehat{R}_{0/1}(h) = 18/300 = 0{,}06$ e sua acurácia é $0{,}94$.

A perda zero-um expressa diretamente a quantidade de erros, mas é descontínua. Por isso, muitos algoritmos treinam com perdas substitutas diferenciáveis.

4.1.3 Matriz de confusão

Em classificação binária há quatro resultados:

Alvo y	Previsão $\hat{y}$	Nome
positivo	positivo	verdadeiro positivo (VP)
negativo	negativo	verdadeiro negativo (VN)
negativo	positivo	falso positivo (FP)
positivo	negativo	falso negativo (FN)

A partir dessas contagens calculamos

$$\text{precisão} = \frac{VP}{VP + FP}, \qquad \text{revocação} = \frac{VP}{VP + FN}$$

e

$$\text{especificidade} = \frac{VN}{VN + FP}.$$

Precisão responde: entre as previsões positivas, quantas estavam corretas? Revocação responde: entre os casos realmente positivos, quantos foram detectados?

Quando classes são muito desbalanceadas, a acurácia pode enganar. Se apenas 1% das transações são fraudulentas, prever "não fraude" para todas produz 99% de acurácia e nenhuma capacidade de detectar fraude.

4.1.4 Custos assimétricos

Falsos positivos e falsos negativos podem ter custos diferentes:

$$\ell(\hat{y}, y) = \begin{cases} 0, & \hat{y} = y, \\ c_{FP}, & \hat{y} = 1 \text{ e } y = 0, \\ c_{FN}, & \hat{y} = 0 \text{ e } y = 1. \end{cases}$$

O risco empírico correspondente é

$$\widehat{R}_D(h) = \frac{c_{FP} FP + c_{FN} FN}{N}.$$

Em autenticação, aceitar uma pessoa não autorizada pode ser mais grave do que pedir que a pessoa autorizada tente novamente. Em triagem, deixar de sinalizar um caso urgente pode ser mais grave do que encaminhar um caso não urgente. Os custos devem representar consequências do contexto.

4.1.5 Regressão: resíduo

Em regressão, o **resíduo** do exemplo i é

$$r_i = h(x_i) - y_i.$$

$r_i > 0$ indica superestimação e $r_i < 0$ indica subestimação. Uma perda transforma esse resíduo em penalidade não negativa.

4.1.6 Erro quadrático médio

A perda quadrática é

$$\ell_{\text{quad}}(\hat{y}, y) = (\hat{y} - y)^2.$$

O **erro quadrático médio** (*mean squared error*, MSE) é

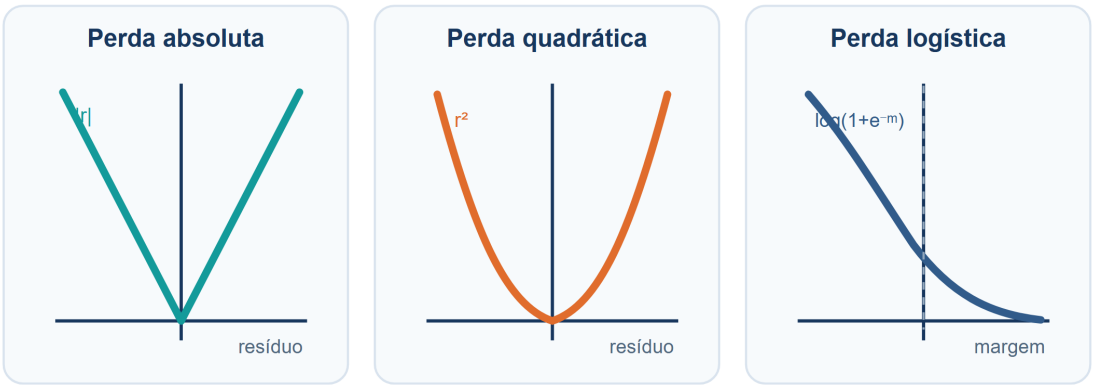


Figura 4.1: A perda absoluta cresce linearmente com o resíduo; a quadrática cresce mais rapidamente; a logística diminui conforme aumenta a margem correta.

$$\mathrm{MSE}(h) = \frac{1}{N}\sum_{i=1}^{N}\big(h(x_i) - y_i\big)^2.$$

Erros grandes recebem penalidade desproporcional. Um resíduo de magnitude 3 contribui 9, enquanto três resíduos de magnitude 1 contribuem 3 no total. Isso torna o MSE sensível a valores extremos.

A raiz do MSE,

$$\mathrm{RMSE}(h) = \sqrt{\mathrm{MSE}(h)},$$

possui a mesma unidade do alvo.

4.1.7 Erro absoluto médio

A perda absoluta é

$$\ell_{\text{abs}}(\hat{y}, y) = |\hat{y} - y|.$$

O **erro absoluto médio** (*mean absolute error*, MAE) é

$$\mathrm{MAE}(h) = \frac{1}{N}\sum_{i=1}^{N}|h(x_i) - y_i|.$$

O MAE cresce linearmente e é menos sensível a observações extremas do que o MSE. Para resíduos $(1, 1, 4)$,

$$\text{MAE} = 2, \qquad \text{MSE} = 6.$$

O erro de magnitude 4 exerce influência muito maior no MSE.

4.1.8 Perda de Huber

A perda de Huber combina comportamento quadrático perto de zero e linear para resíduos grandes:

$$\ell_\delta(r) = \begin{cases} \frac{1}{2}r^2, & |r| \le \delta, \\ \delta\left(|r| - \frac{1}{2}\delta\right), & |r| > \delta. \end{cases}$$

O parâmetro δ determina a transição. Essa perda mantém suavidade perto do mínimo e reduz a influência de valores muito discrepantes.

4.1.9 Classes codificadas por -1 e $+1$

Se $y, \hat{y} \in \{-1, +1\}$, então

$$(\hat{y} - y)^2 = \begin{cases} 0, & \hat{y} = y, \\ 4, & \hat{y} \neq y. \end{cases}$$

Logo, a perda quadrática é quatro vezes a perda zero-um nesse caso específico. Ela não vale 1 quando há erro. Se o modelo produz uma pontuação real, a perda quadrática também considera a distância da pontuação ao rótulo.

4.1.10 Probabilidades e entropia cruzada binária

Considere $y \in \{0, 1\}$ e uma previsão $p = h(x) \in (0, 1)$. A **entropia cruzada binária** é

$$\ell_{\text{BCE}}(p, y) = -\big[y \log p + (1 - y) \log(1 - p)\big].$$

Se $y = 1$, a perda é $-\log p$; se $y = 0$, é $-\log(1 - p)$. Uma previsão confiante e errada recebe perda grande:

$$-\log(0{,}9) \approx 0{,}105, \qquad -\log(0{,}01) \approx 4{,}605.$$

Essa perda avalia a probabilidade atribuída ao resultado observado, não apenas a classe final.

4.1.11 Máxima verossimilhança

Para exemplos i.i.d. e um modelo $P_\theta(Y = y \mid X = x)$, a verossimilhança é

$$L(\theta) = \prod_{i=1}^{N} P_\theta(Y = y_i \mid X = x_i).$$

Máxima verossimilhança escolhe parâmetros que **maximizam** $L(\theta)$. Como o logaritmo é estritamente crescente,

$$\operatorname*{argmax}_\theta \; L(\theta) = \operatorname*{argmax}_\theta \; \log L(\theta).$$

Para escrever o treinamento como minimização, usamos a log-verossimilhança negativa média:

$$\mathrm{NLL}(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \log P_\theta(Y = y_i \mid X = x_i).$$

Para um modelo de Bernoulli, a NLL coincide com a entropia cruzada binária.

> ⚠ Transformações e pontos ótimos
>
> Uma transformação estritamente crescente preserva mínimos e máximos. Uma transformação estritamente decrescente troca máximos por mínimos. Assim, log preserva o máximo da verossimilhança; $-\log$ converte esse máximo em mínimo.

4.1.12 Entropia cruzada multiclasse

Para K classes, o modelo produz $\mathbf{p} = (p_1, \ldots, p_K)$, com $p_k \geq 0$ e $\sum_k p_k = 1$. Se a classe correta é y, a perda é

$$\ell(\mathbf{p}, y) = -\log p_y.$$

Com alvo *one-hot* $\mathbf{y}$,

$$\ell(\mathbf{p}, \mathbf{y}) = -\sum_{k=1}^{K} y_k \log p_k.$$

Essa é a perda habitual para classificadores multiclasse treinados com *softmax*.

4.1.13 Perdas de margem

Em classificação binária com $y \in \{-1, +1\}$ e pontuação $s(x) \in \mathbb{R}$, definimos a margem

$$m = y\, s(x).$$

Margem positiva indica classificação correta; margem negativa indica erro. A perda logística é

$$\ell_{\log}(m) = \log(1 + e^{-m}).$$

Ela diminui suavemente conforme a margem correta aumenta. Podemos otimizar entropia cruzada e avaliar acurácia, precisão, revocação e calibração.

4.1.14 Escolhendo uma perda

Situação	Opção comum	Característica
Regressão com ruído aproximadamente simétrico	MSE	penaliza fortemente erros grandes
Regressão com valores extremos	MAE ou Huber	maior robustez
Classificação probabilística binária	entropia cruzada binária	avalia probabilidades
Classificação multiclasse	entropia cruzada	penaliza baixa probabilidade da classe correta
Custos diferentes entre erros	perda ponderada	incorpora assimetria

A escolha deve considerar o tipo de saída, as consequências dos erros, valores extremos, propriedades necessárias à otimização e a métrica de interesse. Nenhuma perda corrige dados inadequados ou substitui avaliação fora da amostra.

4.1.15 Exemplo computacional

```
def mse(previsoes, alvos):
    erros = [(p - y) ** 2 for p, y in zip(previsoes, alvos)]
    return sum(erros) / len(erros)


def mae(previsoes, alvos):
    erros = [abs(p - y) for p, y in zip(previsoes, alvos)]
    return sum(erros) / len(erros)


y = [10, 12, 14]
pred = [11, 13, 18]

print(mse(pred, y))  # 6.0
print(mae(pred, y))  # 2.0
```

Em produção, também devemos verificar comprimentos, valores ausentes, tipos numéricos e o comportamento quando a amostra está vazia.

4.1.16 Exercícios de fixação

1. Um classificador obteve $VP = 80$, $VN = 900$, $FP = 20$ e $FN = 100$. Calcule acurácia, precisão e revocação.
2. Para resíduos $(2, -1, -3)$, calcule MAE, MSE e RMSE.
3. Explique por que a perda quadrática vale 4, e não 1, quando $y = -1$ e $\hat{y} = +1$.
4. Calcule a entropia cruzada binária para $y = 1$ com $p = 0{,}8$ e com $p = 0{,}2$.
5. Mostre por que maximizar $L(\theta)$ equivale a minimizar $-\log L(\theta)$.
6. Dê um exemplo no qual $c_{FN} > c_{FP}$ e outro no qual $c_{FP} > c_{FN}$.
7. Quando você preferiria MAE à MSE?
8. Explique a diferença entre perda de treinamento e métrica de avaliação.

4.2 Provavelmente Aproximadamente Correto

A teoria **provavelmente aproximadamente correta**, abreviada por **PAC** (*probably approximately correct*), transforma a ideia de generalização em uma garantia matemática. Ela reconhece duas limitações inevitáveis:

- uma amostra finita não determina perfeitamente o comportamento em toda a população;
- como a amostra é aleatória, algumas amostras podem ser pouco representativas.

Em vez de exigir certeza e perfeição, fixamos uma tolerância de erro ε e uma probabilidade de falha δ. Perguntamos quantos exemplos são necessários para que o algoritmo produza uma hipótese suficientemente boa com alta probabilidade.

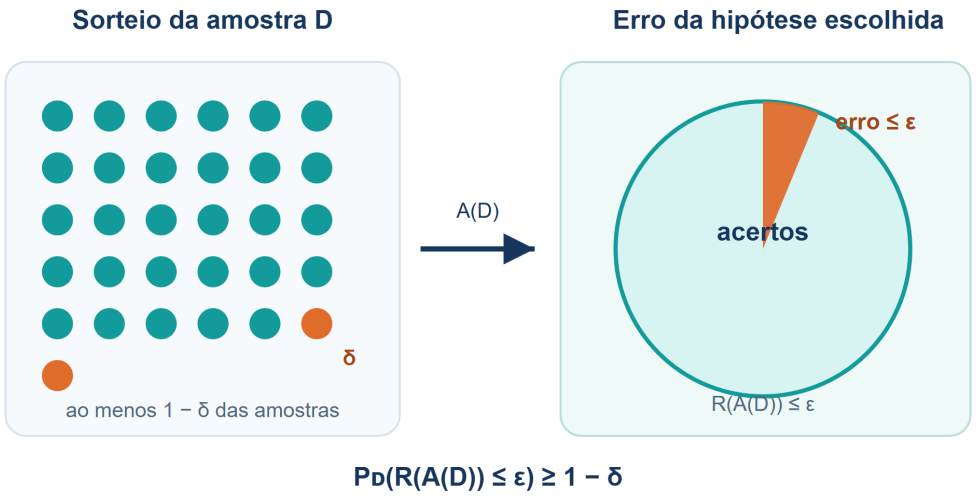


Figura 4.2: Na garantia PAC, pelo menos uma fração $1 - \delta$ das amostras leva o algoritmo a uma hipótese cujo erro é no máximo ε.

4.2.1 Duas fontes de probabilidade

Considere uma distribuição P sobre $X \times Y$ e uma hipótese h. Seu risco é

$$R_P(h) = \mathbb{E}_{(X,Y)\sim P}[\ell(h(X), Y)].$$

Na classificação com perda zero-um,

$$R_P(h) = P_{(X,Y)\sim P}(h(X) \neq Y).$$

Essa probabilidade é sobre um **novo exemplo** sorteado de P. Ela mede a região da população em que h erra.

O algoritmo recebe uma amostra aleatória

$$D = ((x_1, y_1), \ldots, (x_N, y_N)) \sim P^N$$

e devolve $A(D)$. Como D varia, a hipótese também varia. A expressão

$$P_{D\sim P^N}(R_P(A(D)) \leq \varepsilon) \geq 1 - \delta$$

contém uma segunda probabilidade, agora sobre o **sorteio da amostra inteira**. Ela afirma que no máximo uma fração δ das amostras conduz a uma hipótese cujo risco ultrapassa ε.

> **!** Não confunda ε e δ
>
> ε limita o erro da hipótese na população. δ limita a probabilidade de o procedimento de treinamento não alcançar essa garantia. Um atua dentro de cada hipótese; o outro atua sobre as possíveis amostras.

4.2.2 Lendo "provavelmente aproximadamente correto"

Na desigualdade

$$P_D(R_P(A(D)) \leq \varepsilon) \geq 1 - \delta,$$

cada palavra possui um papel:

- **provavelmente:** o evento desejado ocorre com probabilidade de pelo menos $1 - \delta$;
- **aproximadamente:** admitimos erro de até ε;
- **correto:** o erro é medido em relação à distribuição de interesse, não somente na amostra de treinamento.

Se $\varepsilon = 0{,}05$ e $\delta = 0{,}01$, a garantia diz: com confiança de pelo menos 99% sobre a amostra coletada, a hipótese produzida terá risco de no máximo 5%.

Isso não significa que exatamente 5% das previsões estarão erradas ou que o algoritmo falhará exatamente uma vez em cem. Trata-se de uma cota probabilística válida sob as hipóteses do teorema.

4.2.3 Cenário realizável

A forma mais simples da teoria supõe que existe uma hipótese perfeita dentro da classe $\mathcal{H}$. Esse é o **caso realizável**:

$$\exists h^* \in \mathcal{H} \quad \text{tal que} \quad R_P(h^*) = 0.$$

Um algoritmo A é um aprendiz PAC realizável para $\mathcal{H}$ se existe uma função de complexidade amostral

$$m_{\mathcal{H}} \colon (0,1)^2 \to \mathbb{N}$$

tal que, para quaisquer $\varepsilon, \delta \in (0,1)$, qualquer distribuição de entradas e qualquer alvo representável por $\mathcal{H}$, toda amostra i.i.d. com

$$N \geq m_{\mathcal{H}}(\varepsilon, \delta)$$

satisfaz

$$P_D(R_P(A(D)) \leq \varepsilon) \geq 1 - \delta.$$

No caso realizável, um algoritmo **consistente** procura uma hipótese com erro empírico zero. A teoria precisa mostrar quando consistência na amostra implica risco pequeno fora dela.

A hipótese de realizabilidade é forte. Dados podem conter ruído, classes podem se sobrepor e a família escolhida pode não representar o mecanismo real.

4.2.4 Cenário agnóstico

No **aprendizado PAC agnóstico**, não supomos que alguma hipótese em $\mathcal{H}$ seja perfeita. Comparamos o resultado do algoritmo à melhor hipótese disponível na classe:

$$R_P(A(D)) \leq \inf_{h \in \mathcal{H}} R_P(h) + \varepsilon.$$

Formalmente, exigimos

$$P_D\left(R_P(A(D)) \leq \inf_{h \in \mathcal{H}} R_P(h) + \varepsilon\right) \geq 1 - \delta.$$

ε agora é um **excesso de risco**. O algoritmo pode não atingir erro absoluto pequeno se toda hipótese da classe for inadequada; ele deve aproximar o melhor desempenho possível dentro de $\mathcal{H}$.

Essa formulação separa dois componentes:

$$R_P(A(D)) = \underbrace{\inf_{h \in \mathcal{H}} R_P(h)}_{\text{limitação da classe}} + \underbrace{\text{excesso de risco}}_{\text{limitação da amostra e do algoritmo}}.$$

Ampliar $\mathcal{H}$ pode reduzir a limitação de representação, mas torna a seleção estatística mais difícil. Esse é um aspecto do equilíbrio entre ajuste e complexidade.

4.2.5 Complexidade amostral

A **complexidade amostral** $m_{\mathcal{H}}(\varepsilon, \delta)$ é o número mínimo de exemplos suficiente para a garantia desejada. Ela responde a uma pergunta estatística: quanta informação é necessária?

Em geral:

- diminuir ε exige mais exemplos;
- diminuir δ exige mais exemplos;
- aumentar a complexidade de $\mathcal{H}$ exige mais exemplos.

Para uma classe finita no caso realizável, uma cota típica é

$$N \geq \frac{1}{\varepsilon}\left(\log|\mathcal{H}| + \log\frac{1}{\delta}\right).$$

Constantes e pequenas variações dependem da derivação, mas a estrutura é importante. A dependência é logarítmica no número de hipóteses e em $1/\delta$, e proporcional a $1/\varepsilon$.

No caso agnóstico, uma cota de ordem típica é

$$N = O\left(\frac{\log|\mathcal{H}| + \log(1/\delta)}{\varepsilon^2}\right).$$

A dependência $1/\varepsilon^2$ reflete a dificuldade adicional de comparar hipóteses quando nenhuma é necessariamente perfeita.

4.2.6 De onde vem a cota para uma classe finita?

Considere o caso realizável e uma hipótese ruim h com

$$R_P(h) > \varepsilon.$$

A probabilidade de h acertar todos os N exemplos independentes é no máximo

$$(1-\varepsilon)^N \leq e^{-\varepsilon N}.$$

Há no máximo $|\mathcal{H}|$ hipóteses ruins. Pela cota da união, a probabilidade de **alguma** hipótese ruim permanecer consistente é no máximo

$$|\mathcal{H}|e^{-\varepsilon N}.$$

Queremos que essa quantidade seja no máximo δ:

$$|\mathcal{H}|e^{-\varepsilon N} \leq \delta.$$

Aplicando logaritmos e isolando N,

$$N \geq \frac{1}{\varepsilon}\left(\log|\mathcal{H}| + \log\frac{1}{\delta}\right).$$

A derivação mostra por que precisamos controlar simultaneamente a chance de cada hipótese ruim sobreviver e a quantidade de hipóteses candidatas.

4.2.7 Um exemplo numérico

Suponha uma classe finita com

$$|\mathcal{H}| = 1\,000, \qquad \varepsilon = 0{,}05, \qquad \delta = 0{,}01.$$

A cota realizável fornece

$$N \geq \frac{1}{0{,}05}\left(\log 1\,000 + \log 100\right).$$

Usando logaritmo natural,

$$N \geq 20(6{,}908 + 4{,}605) \approx 230{,}26.$$

Logo, a cota pede pelo menos 231 exemplos. Esse valor é uma garantia de pior caso, não uma previsão exata de quantos exemplos serão necessários em toda aplicação.

4.2.8 Classes infinitas

Muitas classes importantes são infinitas. Há infinitas retas no plano e infinitas combinações de pesos de uma rede neural. Nesse caso, $\log |\mathcal{H}|$ não pode ser usado diretamente.

Precisamos de uma medida mais refinada da capacidade da classe. A **dimensão de Vapnik–Chervonenkis**, estudada adiante, mede quantos padrões de classificação uma família consegue realizar sobre conjuntos finitos. Outras ferramentas incluem números de cobertura, complexidade de Rademacher e normas dos parâmetros.

Uma classe infinita pode ser PAC-aprendível se sua capacidade efetiva for controlada. Infinito em cardinalidade não significa automaticamente impossível de aprender.

4.2.9 Aprendibilidade e eficiência computacional

Complexidade amostral e complexidade computacional são diferentes.

- **Aprendibilidade estatística:** existe um algoritmo que alcança a garantia usando um número finito e controlado de exemplos?
- **Eficiência computacional:** esse algoritmo executa em tempo e memória viáveis?

Uma classe pode admitir boa complexidade amostral, mas exigir uma busca computacionalmente intratável. Um aprendiz PAC eficiente costuma exigir tempo polinomial nos parâmetros relevantes da representação, em $1/\varepsilon$ e em $\log(1/\delta)$ ou $1/\delta$, conforme a convenção formal adotada. O ponto central é que obter informação suficiente não garante que conseguiremos processá-la eficientemente.

4.2.10 O que a garantia PAC não afirma

Uma garantia PAC depende de condições. Ela não afirma que:

- os dados i.i.d. representam qualquer cenário futuro;
- a distribuição permanecerá estável após a implantação;
- os rótulos são corretos;
- a perda escolhida representa todos os impactos relevantes;
- o modelo é causal, justo ou seguro;
- a cota é apertada para uma aplicação concreta.

Se treinamento e uso têm distribuições diferentes, a garantia derivada para P pode não descrever a operação real. A teoria fornece uma afirmação condicional: **se** as hipóteses forem satisfeitas, **então** a cota vale.

4.2.11 Exemplo finito e o viés indutivo

Considere

$$X = \{1, 2, \dots, 8\}, \qquad Y = \{-1, +1\},$$

e uma amostra que revela os rótulos de cinco entradas. Restam três rótulos desconhecidos, portanto existem

$$2^3 = 8$$

extensões compatíveis com os cinco valores observados.

Somente a amostra não permite escolher logicamente uma extensão. O algoritmo precisa de um viés indutivo: preferir funções simples, padrões suaves, fronteiras lineares ou outra estrutura. PAC não elimina essa necessidade; ele analisa quando a combinação de classe, algoritmo e quantidade de dados generaliza.

4.2.12 PAC e validação experimental

Uma cota teórica e um conjunto de teste respondem a perguntas diferentes.

- A cota PAC fornece garantia sob hipóteses matemáticas e frequentemente vale no pior caso.
- O teste estima o desempenho de uma hipótese concreta em uma amostra independente.

Na prática, usamos ambos quando disponíveis: teoria para compreender dependências e limites; avaliação experimental para medir o sistema treinado.

4.2.13 Resumo

Um método PAC controla dois níveis de incerteza:

$$\boxed{P_D(R_P(A(D)) \leq \varepsilon) \geq 1 - \delta}$$

no caso realizável, ou

$$\boxed{P_D\left(R_P(A(D)) \leq \inf_{h \in \mathcal{H}} R_P(h) + \varepsilon\right) \geq 1 - \delta}$$

no caso agnóstico.

ε controla a qualidade aproximada, δ controla a confiança e $m_{\mathcal{H}}(\varepsilon, \delta)$ informa quantos exemplos são suficientes.

4.2.14 Exercícios de fixação

1. Explique, com palavras, as duas probabilidades envolvidas em uma garantia PAC.
2. Interprete uma garantia com $\varepsilon = 0{,}02$ e $\delta = 0{,}05$.
3. Diferencie o caso realizável do caso agnóstico.
4. Para $|\mathcal{H}| = 100$, $\varepsilon = 0{,}1$ e $\delta = 0{,}05$, calcule a cota realizável apresentada.
5. Mostre a passagem de $|\mathcal{H}|e^{-\varepsilon N} \leq \delta$ para a cota sobre N.
6. Por que uma classe infinita não é automaticamente impossível de aprender?
7. Dê um exemplo em que a hipótese i.i.d. falha.
8. Diferencie complexidade amostral de eficiência computacional.
9. Explique por que uma garantia PAC não substitui um conjunto de teste.

4.3 2.3 Treinar e Testar

Uma hipótese é escolhida porque se ajusta aos dados disponíveis. Avaliá-la nos mesmos exemplos usados para escolhê-la produz uma estimativa otimista: o modelo já teve oportunidade de adaptar-se a particularidades dessa amostra. Para estimar generalização, precisamos de dados que não tenham participado das escolhas.

A prática usual divide as observações em conjuntos com papéis distintos:

- **treinamento:** ajusta os parâmetros do modelo;
- **validação:** escolhe hiperparâmetros, atributos, arquitetura e limiares;
- **teste:** estima uma única vez o desempenho da configuração final.

4.3.1 Parâmetros e hiperparâmetros

Parâmetros são aprendidos diretamente pelo algoritmo de treinamento. Em uma função linear, são os pesos $\mathbf{w}$ e o viés b.

Hiperparâmetros controlam o método, mas não são ajustados pela mesma minimização. Exemplos incluem taxa de aprendizado, intensidade de regularização, número de camadas, profundidade de uma árvore e quantidade de vizinhos.

Treinar em D_{treino} produz

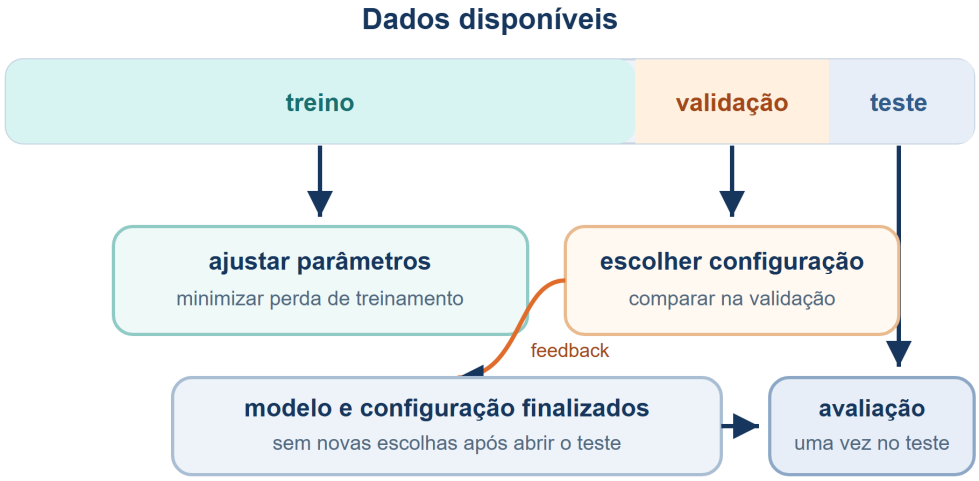


Figura 4.3: Os dados de treino ajustam parâmetros, a validação orienta escolhas e o teste permanece isolado até a avaliação final.

$$\hat{\theta}_\lambda \in \operatorname*{argmin}_\theta \widehat{R}_{\text{treino}}(h_{\theta,\lambda}),$$

em que λ representa uma configuração de hiperparâmetros. A validação escolhe

$$\hat{\lambda} \in \operatorname*{argmin}_{\lambda \in \Lambda} \widehat{R}_{\text{val}}(h_{\hat{\theta}_\lambda,\lambda}).$$

Somente após fixar $\hat{\lambda}$ avaliamos a configuração final no teste.

4.3.2 Holdout

No protocolo **holdout**, fazemos uma única divisão dos dados. Uma proporção ilustrativa é

$$70\% \text{ treino}, \qquad 15\% \text{ validação}, \qquad 15\% \text{ teste}.$$

Não existe uma proporção universal. Com milhões de exemplos, uma pequena fração pode formar um teste preciso. Com poucos dados, reservar uma parte grande reduz demais o treinamento e torna a validação instável.

A divisão deve ser feita antes da exploração orientada pelos rótulos. Uma semente pseudoaleatória registrada permite reproduzir a partição, mas testar muitas sementes e escolher a mais favorável também constitui uma forma de sobreajuste.

> **!** Teste não é validação
>
> Se uma decisão muda depois que observamos o resultado no teste, então o teste participou do desenvolvimento. Ele deve ser considerado validação, e um novo conjunto independente será necessário para avaliação final.

4.3.3 Independência e representatividade

A validade do holdout depende de treino, validação e teste representarem o cenário de uso sem compartilhar informação indevida. Em uma divisão aleatória i.i.d., esperamos que os subconjuntos tenham distribuições semelhantes.

Entretanto, a divisão aleatória simples não é adequada em todos os casos:

- em séries temporais, o futuro não pode aparecer no treino de uma previsão do passado;
- em dados de pacientes, registros da mesma pessoa não devem ficar em subconjuntos diferentes;
- em imagens extraídas de um vídeo, quadros vizinhos são quase duplicados;
- em recomendação, podemos querer testar usuários ou itens ainda não observados;
- em implantação geográfica, pode ser necessário reservar regiões inteiras.

A unidade de independência deve corresponder ao uso. Se a previsão será feita para novas pessoas, a divisão precisa ocorrer por pessoa, não por registro.

4.3.4 Divisão estratificada

Em classificação desbalanceada, uma divisão puramente aleatória pode produzir poucos exemplos da classe rara. A **estratificação** preserva aproximadamente as proporções das classes em cada subconjunto.

Se há $1\,000$ exemplos, dos quais 100 são positivos, uma divisão estratificada de $80\%/20\%$ coloca aproximadamente 80 positivos no treino e 20 no teste.

Estratificar não cria informação nova e não resolve uma classe extremamente rara. Também não deve violar grupos ou tempo. Quando há várias restrições, usamos uma estratégia que respeite primeiro a estrutura causal do conjunto de dados.

4.3.5 Vazamento de dados

Vazamento ocorre quando informação indisponível no momento real da previsão influencia o treinamento ou a avaliação. O resultado parece melhor do que o sistema será em produção.

Exemplos:

- normalizar usando média e desvio calculados em todos os dados;
- escolher atributos observando correlação no conjunto de teste;
- imputar valores ausentes antes da divisão;
- incluir uma variável registrada depois do evento previsto;
- manter cópias do mesmo exemplo em treino e teste.

Todo pré-processamento que aprende alguma quantidade deve ser ajustado somente no treino. Para uma padronização,

$$z_j = \frac{x_j - \mu_j}{\sigma_j},$$

μ_j e σ_j são calculados em D_{treino} e depois aplicados, sem reajuste, à validação e ao teste.

Uma **pipeline** reúne pré-processamento e modelo para que cada etapa seja ajustada dentro do protocolo correto.

4.3.6 Validação cruzada k-fold

Quando os dados são limitados, uma única validação pode depender muito da divisão. Na **validação cruzada k-fold**, separamos os dados de desenvolvimento em k blocos aproximadamente iguais:

$$D_{\text{desenv}} = F_1 \cup F_2 \cup \cdots \cup F_k.$$

Para cada j, treinamos em todos os blocos exceto F_j e avaliamos em F_j. A estimativa é

$$\widehat{R}_{CV} = \frac{1}{k} \sum_{j=1}^{k} \widehat{R}_{F_j}(h^{(-j)}),$$

em que $h^{(-j)}$ foi treinada sem o bloco F_j.

Cada exemplo participa da validação uma vez e do treinamento $k-1$ vezes. Valores comuns são $k = 5$ ou $k = 10$, mas a escolha depende do custo computacional e da quantidade de dados.

É importante que o teste final permaneça fora da validação cruzada. Usamos os folds para escolher a configuração; depois treinamos a configuração escolhida nos dados de desenvolvimento e avaliamos no teste isolado.

4.3.7 Validação cruzada estratificada e por grupos

A validação cruzada também precisa respeitar a estrutura dos dados.

- **Estratificada:** preserva proporções de classes em cada fold.
- **Por grupos:** mantém todos os exemplos de um grupo no mesmo fold.
- **Temporal:** usa blocos que respeitam a ordem, treinando no passado e validando no futuro.

Em previsão temporal, uma forma progressiva é

$$\{1, \dots, t_1\} \to \{t_1 + 1, \dots, t_2\},$$

seguida por

$$\{1, \dots, t_2\} \to \{t_2 + 1, \dots, t_3\}.$$

Isso imita o modo como o sistema acumulará histórico e fará previsões futuras.

4.3.8 Amostragem aleatória repetida

O **holdout repetido**, também chamado de *random subsampling*, cria várias divisões aleatórias treino-validação. Para cada repetição r, treinamos uma nova hipótese h_r e calculamos uma métrica M_r.

Relatamos, por exemplo,

$$\overline{M} = \frac{1}{R} \sum_{r=1}^{R} M_r$$

e a variabilidade entre repetições. Diferentemente do k-fold, um exemplo pode aparecer na validação várias vezes e outro nenhuma vez.

Esse protocolo mede a sensibilidade do **procedimento** à divisão dos dados. Os valores não são totalmente independentes porque as partições se sobrepõem; a dispersão deve ser interpretada como diagnóstico, não automaticamente como intervalo de confiança clássico.

4.3.9 Validação cruzada aninhada

Se usamos validação cruzada tanto para escolher hiperparâmetros quanto para estimar desempenho, podemos obter otimismo. A **validação cruzada aninhada** usa dois níveis:

1. folds internos escolhem hiperparâmetros;
2. um fold externo, que não participou da escolha, avalia o procedimento.

O ciclo externo estima o desempenho do processo completo de seleção. Esse método é útil quando o conjunto é pequeno e muitas configurações são comparadas, embora seja computacionalmente mais caro.

4.3.10 Matriz de confusão no teste

Para K classes, definimos a matriz $C \in \mathbb{N}^{K\times K}$ por

$$C_{ij} = \#\{(x,y) \in D_{\text{teste}} \mid y = i,\ h(x) = j\}.$$

Nesta convenção, linhas são classes verdadeiras e colunas são classes previstas. A diagonal contém acertos; elementos fora da diagonal mostram quais classes foram confundidas.

Um exemplo é

alvo \ previsão	classe 1	classe 2	classe 3
classe 1	7	2	1
classe 2	1	8	1
classe 3	1	0	9

A acurácia é

$$\frac{7+8+9}{30} = 0{,}8.$$

A matriz é mais informativa do que a acurácia isolada porque revela a direção dos erros.

4.3.11 Incerteza da estimativa de teste

O desempenho observado no teste ainda é uma estimativa aleatória. Para classificação, se M exemplos de teste são aproximadamente independentes e a taxa observada de erro é $\hat{p}$, um erro-padrão aproximado é

$$\mathrm{EP}(\hat{p}) \approx \sqrt{\frac{\hat{p}(1-\hat{p})}{M}}.$$

Com $\hat{p} = 0{,}10$ e $M = 1\,000$,

$$\text{EP} \approx \sqrt{\frac{0{,}1 \cdot 0{,}9}{1\,000}} \approx 0{,}0095.$$

Um teste pequeno produz uma estimativa instável. Para proporções próximas de 0 ou 1, amostras pequenas ou dependência entre exemplos, devemos usar métodos de intervalo adequados em vez de depender apenas dessa aproximação.

4.3.12 Escolha de limiar

Modelos probabilísticos frequentemente produzem uma pontuação $p(x)$. Escolher um limiar com base no teste vaza informação. O procedimento correto é:

1. treinar o modelo no treino;
2. escolher o limiar na validação, considerando a métrica ou o custo;
3. congelar modelo e limiar;
4. avaliar ambos no teste.

O mesmo vale para seleção de atributos, regularização, quantidade de épocas e qualquer decisão guiada por desempenho.

4.3.13 Um protocolo recomendado

Para um projeto supervisionado típico:

1. defina a unidade de divisão e reserve o teste;
2. construa uma pipeline de pré-processamento e modelo;
3. use treino e validação, ou validação cruzada, para escolhas;
4. registre a configuração final;
5. se apropriado, reajuste essa configuração em treino mais validação;
6. avalie uma única vez no teste;
7. relate métricas, incerteza, tamanho dos conjuntos e estratégia de divisão;
8. monitore mudança de distribuição após a implantação.

Reajustar em treino mais validação é permitido depois que todas as escolhas estão congeladas. O teste continua isolado. Se o resultado de teste motivar nova alteração, inicia-se outra rodada de desenvolvimento e será preciso obter uma avaliação final independente.

4.3.14 Reprodutibilidade

Uma avaliação reproduzível registra:

- versão e origem dos dados;
- critérios de inclusão e exclusão;
- identificadores ou índices de cada partição;
- semente aleatória;
- etapas de pré-processamento;
- hiperparâmetros e critério de seleção;
- métricas e respectivas definições;
- versão do código e das dependências.

Registrar somente a semente não basta se os dados ou a biblioteca mudarem.

4.3.15 Relação com a teoria

A teoria PAC pergunta quando o risco empírico se aproxima do risco esperado sob certas hipóteses. O protocolo treino-validação-teste fornece uma avaliação operacional para um conjunto de dados concreto.

Este livro usa análises teóricas para compreender algoritmos, mas exemplos computacionais devem manter avaliação separada sempre que afirmarem desempenho preditivo. Não devemos confiar apenas no erro de treinamento.

> ⚠ Regra de ouro
>
> Qualquer dado usado para escolher o modelo pertence ao processo de desenvolvimento. Somente dados que permaneceram intocados podem sustentar uma avaliação final honesta.

4.3.16 Exercícios de fixação

1. Diferencie parâmetros, hiperparâmetros e métricas.
2. Explique por que normalizar antes da divisão causa vazamento.
3. Proponha uma divisão adequada para várias consultas do mesmo usuário.
4. Em que situação uma divisão temporal é necessária?
5. Descreva o procedimento de validação cruzada 5-fold.
6. Diferencie holdout repetido e validação cruzada.
7. Por que a validação cruzada aninhada reduz o viés de seleção?
8. Calcule a acurácia da matriz de confusão apresentada e a revocação da classe 2.
9. Para $\hat{p} = 0{,}2$ e $M = 400$, calcule o erro-padrão aproximado.
10. Explique o que fazer se o resultado de teste motivar uma nova escolha de modelo.

4.4 Exercícios de múltipla escolha

Assinale a única alternativa correta em cada questão. O gabarito comentado está recolhido ao final da seção.

1. A perda zero–um de um classificador vale 1 quando:

   a. a previsão está correta.

   b. a previsão difere do rótulo verdadeiro.

   c. a probabilidade prevista é exatamente 1.

   d. o conjunto de teste está vazio.

2. Em uma aplicação médica na qual deixar de detectar uma doença é muito grave, qual medida merece atenção especial?

   a. Revocação da classe positiva.

   b. Número de atributos.

   c. Erro de arredondamento.

   d. Cardinalidade do espaço de saída.

3. Em comparação com o erro quadrático médio, o erro absoluto médio é, em geral:

   a. mais sensível a valores atípicos.

   b. menos sensível a valores atípicos.

   c. sempre igual a zero.

   d. aplicável somente à classificação.

4. Na entropia cruzada binária, atribuir probabilidade próxima de zero ao evento que realmente ocorreu produz:

   a. perda próxima de zero.

   b. perda negativa.

c. perda muito elevada.

d. nenhuma alteração na perda.

5. Na linguagem PAC, o parâmetro ϵ controla:

   a. a tolerância de erro ou precisão desejada.

   b. a quantidade de classes do problema.

   c. a semente do gerador aleatório.

   d. o custo de uma multiplicação matricial.

6. Ainda no contexto PAC, $1 - \delta$ representa:

   a. a taxa de aprendizado.

   b. o nível de confiança da garantia.

   c. o erro empírico exato.

   d. o número de hipóteses.

7. Se aumentamos a complexidade da classe de hipóteses e mantemos os demais fatores, a quantidade de dados necessária para uma mesma garantia tende a:

   a. diminuir até zero.

   b. permanecer sempre idêntica.

   c. aumentar.

   d. tornar-se independente de ϵ.

8. Qual prática caracteriza vazamento de dados?

   a. Ajustar o normalizador somente com o treino.

   b. Escolher hiperparâmetros usando validação.

c. Calcular média e desvio com todo o conjunto antes da divisão.

d. Avaliar uma única vez no teste ao final.

9. Quando várias observações pertencem ao mesmo paciente, a divisão mais segura é:

a. sortear cada observação independentemente.

b. manter todas as observações de cada paciente no mesmo subconjunto.

c. duplicar pacientes raros no teste.

d. usar treino e teste com os mesmos pacientes obrigatoriamente.

10. Depois que o conjunto de teste influenciou uma escolha de modelo, ele deve ser tratado como:

a. parte do processo de desenvolvimento, e uma nova avaliação final exige dados intocados.

b. um teste final ainda perfeitamente independente.

c. um conjunto sem qualquer utilidade.

d. uma amostra de treinamento sem rótulos.

Gabarito comentado

1. **b.** A perda zero–um registra erro com 1 e acerto com 0.
2. **a.** A revocação mede a fração dos positivos reais que foram detectados e, portanto, evidencia falsos negativos.
3. **b.** O quadrado amplifica resíduos grandes; o valor absoluto cresce apenas linearmente.
4. **c.** O logaritmo penaliza fortemente previsões confiantes e incorretas.
5. **a.** ϵ expressa quão próximo do desempenho desejado o resultado deve estar.
6. **b.** δ limita a probabilidade de falha; assim, $1 - \delta$ é a confiança.
7. **c.** Uma classe mais rica exige mais evidência para controlar a seleção entre suas hipóteses.
8. **c.** As estatísticas incorporariam informação dos conjuntos que deveriam permanecer independentes.
9. **b.** A separação por grupos impede que informações específicas do paciente apareçam

simultaneamente no treino e na avaliação.

10. **a.** Toda informação usada para decidir passa a integrar o desenvolvimento; a estimativa final requer novos dados independentes.

Capítulo 5

Quão bem aprendemos?

Nos capítulos anteriores, definimos aprendizado como a construção de uma hipótese a partir de uma amostra finita. Também distinguimos o erro observado nos dados de treinamento do erro esperado em exemplos futuros. Agora enfrentaremos a pergunta quantitativa: **quando o erro observado fornece informação confiável sobre o erro verdadeiro?**

Considere uma hipótese h e uma amostra i.i.d.

$$D = \{(x_i, y_i)\}_{i=1}^{N} \sim P^N.$$

Para uma perda limitada, calculamos o risco empírico

$$\widehat{R}_D(h) = \frac{1}{N} \sum_{i=1}^{N} \ell(h(x_i), y_i),$$

mas desejamos conhecer o risco esperado

$$R_P(h) = \mathbb{E}_{(X,Y)\sim P}\big[\ell(h(X), Y)\big].$$

Como P é desconhecida, $R_P(h)$ não pode ser calculado diretamente. A teoria de generalização estuda a diferença

$$\big|\widehat{R}_D(h) - R_P(h)\big|.$$

Uma diferença pequena indica que a amostra estima bem o desempenho da hipótese. Uma diferença grande significa que o resultado observado pode ser uma descrição enganosa da

população.

A primeira dificuldade: a amostra é aleatória. Se repetirmos a coleta, obteremos outros exemplos e outro risco empírico. Precisamos de uma afirmação probabilística que limite a chance de um desvio grande. A desigualdade de Hoeffding cumprirá esse papel para médias de variáveis independentes e limitadas.

A segunda dificuldade: a hipótese também depende da amostra. Hoeffding controla diretamente uma hipótese fixada antes de observar os dados. Um algoritmo, porém, examina os dados e escolhe

$$\hat{h} = A(D).$$

Se procurarmos entre muitas hipóteses, alguma pode parecer boa apenas por acaso. Esse efeito é semelhante a lançar muitas moedas e selecionar a que apresentou a maior proporção de caras: a seleção torna o resultado extremo menos surpreendente.

A terceira dificuldade: muitas classes são infinitas. Existem infinitas retas, planos e combinações de pesos reais. Não podemos simplesmente contar todos os elementos de $\mathcal{H}$. Precisamos medir quantos comportamentos distintos a classe consegue produzir **sobre uma amostra finita**.

A dimensão de Vapnik–Chervonenkis, ou **dimensão VC**, mede essa capacidade em classificação binária. Ela é o maior tamanho de um conjunto que pode ser **estilhaçado** pela classe: todas as suas rotulações binárias podem ser realizadas por alguma hipótese.

Para classificadores lineares no plano, a dimensão VC é 3:

- existem três pontos que podem receber quaisquer rótulos e ainda ser separados por uma reta;
- nenhum conjunto de quatro pontos pode realizar, por retas, todas as 2^4 rotulações possíveis.

Os quatro vértices de um quadrado com rótulos alternados formam o conhecido padrão XOR e mostram uma rotulação que uma reta não separa. Para concluir que a dimensão é 3, porém, precisamos ainda argumentar que **todo** conjunto de quatro pontos falha em alguma rotulação, não apenas exibir um conjunto particular.

O roteiro do capítulo. Desenvolveremos a análise em etapas:

1. entenderemos o defeito de selecionar hipóteses usando a mesma amostra em que medimos o erro;
2. aplicaremos a desigualdade de Hoeffding a uma hipótese fixa e depois a classes finitas;

3. substituiremos a cardinalidade de uma classe infinita pelo número de rotulações que ela induz em amostras finitas;
4. definiremos dimensão VC e obteremos cotas de generalização dependentes da capacidade.

Uma garantia típica terá a forma

$$P_D\left(\sup_{h\in\mathcal{H}}\left|\widehat{R}_D(h)-R_P(h)\right|>\varepsilon\right)\leq\delta.$$

O supremo exige que a aproximação seja válida **simultaneamente para todas as hipóteses**. Isso permite escolher uma hipótese depois de observar a amostra sem perder a garantia.

5.0.1 Capacidade, dados e confiança

As cotas revelarão três dependências gerais:

- maior precisão, isto é, menor ε, requer mais exemplos;
- maior confiança, isto é, menor δ, requer mais exemplos;
- uma classe mais expressiva requer mais exemplos para controlar seleção e sobreajuste.

Essas relações não justificam regras universais como "dez exemplos por parâmetro". A quantidade necessária depende da perda, da capacidade efetiva, da distribuição, do ruído e do tipo de garantia. Cotas teóricas de pior caso podem ser conservadoras, mas sua estrutura explica quais fatores tornam o aprendizado mais difícil.

5.0.2 O que as cotas podem e não podem fazer

Uma cota de generalização não escolhe automaticamente a melhor representação, não corrige rótulos incorretos e não garante estabilidade quando a distribuição muda. Ela responde a uma pergunta condicional: sob determinadas hipóteses, qual é a probabilidade de o desempenho empírico diferir muito do esperado?

Também devemos distinguir:

- **cota teórica:** válida para uma família de situações, muitas vezes conservadora;
- **estimativa experimental:** calculada em validação ou teste para um modelo e conjunto concretos.

As duas são complementares. A teoria explica por que o procedimento pode generalizar; a avaliação mede como uma implementação específica se comportou.

> **Uma mudança de perspectiva**
>
> O problema não é apenas encontrar uma hipótese com baixo erro de treinamento. É controlar quantas oportunidades o algoritmo teve de encontrar um ajuste acidental e demonstrar que o baixo erro contém informação sobre novos dados.

5.0.3 Exemplo motivador: muitas moedas

Suponha M moedas honestas, cada uma lançada N vezes. Para uma moeda fixada antes do experimento, a frequência de caras tende a ficar próxima de $1/2$. Entretanto, se escolhermos depois a moeda com menor frequência de caras, o valor selecionado será mais extremo à medida que M cresce.

A moeda selecionada desempenha o papel da hipótese escolhida pelo algoritmo. Cada moeda individual possui boa concentração, mas controlar a seleção exige considerar todas as candidatas. Para um conjunto finito, faremos isso com a cota da união; para classes infinitas, usaremos capacidade combinatória.

5.0.4 Perguntas de preparação

1. Diferencie risco empírico e risco esperado.
2. Por que uma garantia para uma hipótese fixa não pode ser aplicada diretamente a $A(D)$?
3. Como aumentar a quantidade de hipóteses afeta a chance de encontrar um ajuste acidental?
4. O que significa estilhaçar um conjunto?
5. Por que o padrão XOR de quatro pontos, isoladamente, não completa a prova de que retas no plano têm dimensão VC igual a 3?
6. Explique por que não existe uma regra universal de exemplos por parâmetro.

5.1 Defeito

Dada uma hipótese h, podemos calcular seu risco empírico na amostra,

$$\widehat{R}_D(h),$$

mas não seu risco esperado exato,

$$R_P(h),$$

porque a distribuição P é desconhecida. Chamaremos de **lacuna de generalização** — ou, seguindo a terminologia histórica deste texto, **defeito** — a diferença

$$\Delta_D(h) = R_P(h) - \widehat{R}_D(h).$$

A identidade

$$R_P(h) = \widehat{R}_D(h) + \Delta_D(h)$$

mostra os dois ingredientes de um bom resultado: erro de treinamento pequeno e lacuna controlada.

5.1.1 Lacuna assinada e lacuna absoluta

$\Delta_D(h)$ é uma quantidade assinada:

- $\Delta_D(h) > 0$: o treino subestimou o erro verdadeiro;
- $\Delta_D(h) < 0$: o treino superestimou o erro verdadeiro;
- $\Delta_D(h) \approx 0$: a amostra estimou bem o risco.

Para uma garantia que não dependa da direção, usamos

$$\left|R_P(h) - \widehat{R}_D(h)\right|.$$

Então,

$$R_P(h) \leq \widehat{R}_D(h) + \left|R_P(h) - \widehat{R}_D(h)\right|.$$

Não tentamos "minimizar a lacuna" diretamente, pois ela envolve $R_P(h)$, que não conhecemos. Procuramos demonstrar que ela é pequena com alta probabilidade.

5.1.2 Classificação binária

Para $Y = \{-1, +1\}$ e perda zero-um,

$$\widehat{R}_D(h) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}[h(x_i) \neq y_i]$$

e

$$R_P(h) = P_{(X,Y)\sim P}(h(X) \neq Y).$$

O primeiro é a frequência de erros nos N exemplos; o segundo é a probabilidade de erro em um novo exemplo.

Se $\widehat{R}_D(h) = 0{,}02$ e uma cota garante

$$\left|R_P(h) - \widehat{R}_D(h)\right| \leq 0{,}03,$$

então

$$R_P(h) \leq 0{,}05.$$

A cota não afirma que o risco seja exatamente 5%; fornece um limite superior.

5.1.3 Uma hipótese fixada antes dos dados

Suponha que h seja escolhida sem observar D. Cada variável

$$Z_i = \mathbb{1}[h(x_i) \neq y_i]$$

é Bernoulli, com média

$$\mathbb{E}[Z_i] = R_P(h).$$

O risco empírico é a média

$$\widehat{R}_D(h) = \frac{1}{N}\sum_{i=1}^{N} Z_i.$$

Uma desigualdade de concentração pode então limitar a probabilidade de essa média afastar-se de sua esperança. Para uma hipótese fixa, a amostra funciona como uma pesquisa de opinião: mais observações tornam desvios grandes menos prováveis.

5.1.4 O defeito da seleção

Um algoritmo não avalia apenas uma hipótese fixada antecipadamente. Ele usa os dados para escolher

$$\hat{h} \in \operatorname*{argmin}_{h \in \mathcal{H}} \widehat{R}_D(h).$$

A hipótese escolhida é uma variável aleatória dependente de D. Aplicar diretamente uma garantia para h fixa a $\hat{h} = A(D)$ ignora essa dependência.

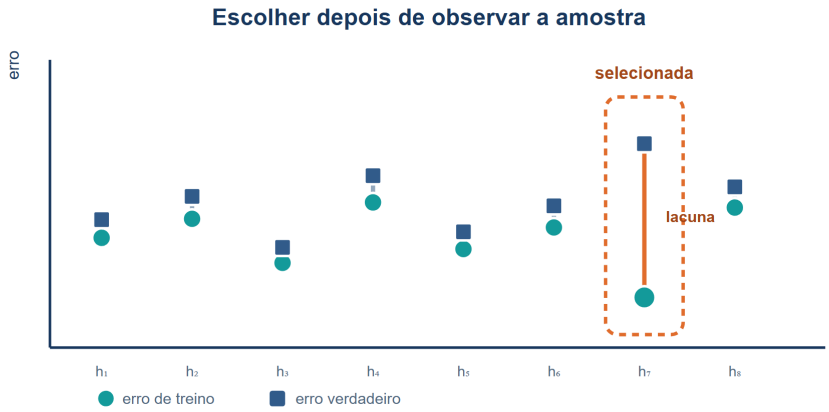


Figura 5.1: Ao comparar muitas hipóteses, a selecionada pelo menor erro de treino pode ser justamente aquela que mais se beneficiou de uma flutuação da amostra.

Mesmo que cada risco empírico seja individualmente um estimador razoável, o menor entre muitos estimadores tende a ser otimista. Esse fenômeno é chamado de **viés de seleção** ou efeito de comparações múltiplas.

5.1.5 Analogia das moedas

Imagine M moedas honestas lançadas N vezes. Para cada moeda j, seja ν_j a frequência de caras. Cada ν_j concentra-se em torno de $1/2$.

Agora selecionamos

$$j^* \in \operatorname*{argmin}_{1 \le j \le M} \nu_j.$$

ν_{j^*} tende a ficar abaixo de $1/2$, não porque a moeda escolhida seja desonesta, mas porque procuramos deliberadamente a flutuação mais baixa entre M tentativas.

Quanto maior M, maior a oportunidade de observar um valor extremo. Em aprendizado, hipóteses desempenham o papel das moedas e o erro empírico desempenha o papel da frequência observada.

5.1.6 Controle uniforme

Para permitir seleção dependente dos dados, buscamos controlar todas as hipóteses simultaneamente:

$$\sup_{h \in \mathcal{H}} \left| R_P(h) - \widehat{R}_D(h) \right|.$$

Se, para uma amostra, vale

$$\sup_{h \in \mathcal{H}} \left| R_P(h) - \widehat{R}_D(h) \right| \leq \varepsilon,$$

então a desigualdade vale para qualquer hipótese da classe, inclusive aquela escolhida depois de observar D.

Esse é o motivo para uma **cota uniforme**. A complexidade de $\mathcal{H}$ aparece porque controlar mais comportamentos simultaneamente é mais difícil.

5.1.7 Decomposição para minimização do risco empírico

Seja

$$\hat{h} \in \operatorname*{argmin}_{h \in \mathcal{H}} \widehat{R}_D(h)$$

e seja

$$h^* \in \operatorname*{argmin}_{h \in \mathcal{H}} R_P(h).$$

Suponha que o desvio uniforme seja no máximo ε:

$$\sup_{h \in \mathcal{H}} \left| R_P(h) - \widehat{R}_D(h) \right| \leq \varepsilon.$$

Então,

$$\begin{aligned} R_P(\hat{h}) &\leq \widehat{R}_D(\hat{h}) + \varepsilon \\ &\leq \widehat{R}_D(h^*) + \varepsilon \\ &\leq R_P(h^*) + 2\varepsilon. \end{aligned}$$

A segunda desigualdade usa o fato de que $\hat{h}$ minimiza o risco empírico. Portanto,

$$R_P(\hat{h}) - R_P(h^*) \leq 2\varepsilon.$$

Essa derivação mostra como convergência uniforme transforma minimização empírica em uma garantia de excesso de risco.

5.1.8 Erro de aproximação e erro de estimação

O desempenho pode ser separado conceitualmente em:

- **erro de aproximação:** a melhor hipótese de $\mathcal{H}$ ainda pode ser inadequada;
- **erro de estimação:** a amostra finita pode levar o algoritmo a escolher uma hipótese pior do que a melhor da classe;
- **erro de otimização:** o procedimento numérico pode não encontrar o mínimo empírico.

Uma classe maior tende a reduzir o erro de aproximação, mas pode aumentar o erro de estimação. Mais dados reduzem a incerteza estatística, mas não corrigem uma classe que não representa o padrão.

5.1.9 Sobreajuste como lacuna

O sobreajuste ocorre quando o modelo aproveita detalhes específicos da amostra que não se repetem na população. Um sinal típico é

$$\widehat{R}_D(\hat{h}) \ll R_P(\hat{h}).$$

A lacuna pode crescer quando:

- a classe possui alta capacidade;
- comparamos muitas configurações;
- a amostra é pequena;
- os dados contêm ruído;
- repetimos decisões usando o mesmo conjunto de validação.

Regularização, validação independente e controle de capacidade reduzem oportunidades de ajuste acidental, mas não substituem dados representativos.

5.1.10 Uma estimativa não observável

Durante treinamento conhecemos $\widehat{R}_D(h)$, mas não $\Delta_D(h)$, pois este depende de $R_P(h)$. Uma cota teórica limita a lacuna sob hipóteses. Um conjunto de teste independente fornece outra estimativa empírica:

$$\widehat{R}_{\text{teste}}(h) \approx R_P(h).$$

Teste e teoria abordam o mesmo risco por caminhos diferentes, ambos sujeitos a condições.

> **Escolher e avaliar no mesmo conjunto**
>
> Quanto mais decisões forem tomadas olhando uma métrica, menos esse conjunto poderá ser tratado como avaliação independente. O sobreajuste pode ocorrer na validação, não apenas no treinamento.

5.1.11 Exemplo numérico

Considere três hipóteses:

Hipótese	Erro de treino	Erro de validação
h_1	0,12	0,13
h_2	0,07	0,11
h_3	0,00	0,24

Se escolhermos apenas pelo treino, h_3 vence. A validação sugere que ela se ajustou demais. h_2 possui menor erro de validação, embora não tenha o menor erro de treino.

Não conhecemos o risco verdadeiro dessas hipóteses, mas dados independentes ajudam a detectar a discrepância.

5.1.12 Exercícios de fixação

1. Interprete o sinal de $\Delta_D(h)$.
2. Se $\widehat{R}_D(h) = 0{,}08$ e a lacuna absoluta é no máximo $0{,}04$, qual cota obtemos para $R_P(h)$?
3. Por que a hipótese escolhida por minimização empírica não é fixa em relação à amostra?

4. Explique a analogia entre muitas moedas e muitas hipóteses.
5. Reproduza a decomposição que leva a $R_P(\hat{h}) \leq R_P(h^*) + 2\varepsilon$.
6. Diferencie erro de aproximação, estimação e otimização.
7. Dê um exemplo de sobreajuste ao conjunto de validação.
8. Por que não podemos calcular diretamente a lacuna de generalização durante o treinamento?

5.2 3.2 Desigualdade de Hoeffding

A desigualdade de Hoeffding controla o desvio entre a média de variáveis aleatórias limitadas e sua esperança. Ela é especialmente útil em aprendizado porque muitas perdas são limitadas e o risco empírico é uma média amostral.

5.2.1 Enunciado geral

Sejam $Z_1, \dots, Z_N$ variáveis aleatórias independentes, com

$$a_i \leq Z_i \leq b_i.$$

Defina

$$\overline{Z} = \frac{1}{N} \sum_{i=1}^{N} Z_i$$

e

$$\mu = \mathbb{E}[\overline{Z}].$$

Então, para $\varepsilon > 0$,

$$P\left(|\overline{Z} - \mu| \geq \varepsilon\right) \leq 2 \exp\left(-\frac{2N^2\varepsilon^2}{\sum_{i=1}^{N}(b_i - a_i)^2}\right).$$

Quando todas as variáveis pertencem a $[0, 1]$, obtemos a forma conhecida:

$$P\left(|\overline{Z} - \mu| \geq \varepsilon\right) \leq 2e^{-2N\varepsilon^2}.$$

A desigualdade é **livre de distribuição**: não exige que conheçamos a forma da distribuição de Z_i. Exige independência e limites conhecidos.

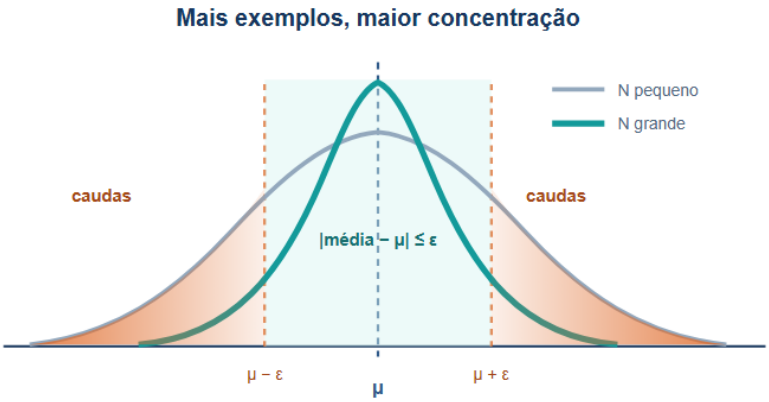


Figura 5.2: À medida que N aumenta, a média amostral se concentra ao redor da esperança e a probabilidade nas caudas fora de $\mu \pm \varepsilon$ diminui.

5.2.2 Interpretação

A cota decai exponencialmente em N e em ε^2:

$$2e^{-2N\varepsilon^2}.$$

Isso significa:

- mais exemplos tornam desvios fixos menos prováveis;
- tolerar um desvio maior torna a garantia mais fácil;
- reduzir o desvio desejado pela metade requer aproximadamente quatro vezes mais exemplos.

A cota pode ser maior que 1 para N pequeno ou ε muito pequeno. Como uma probabilidade nunca excede 1, podemos escrever

$$P(\cdots) \leq \min\{1, 2e^{-2N\varepsilon^2}\}.$$

Uma cota maior que 1 é válida, mas não informativa.

5.2.3 Aplicação a uma hipótese fixa

Fixe uma hipótese h **antes** de observar a amostra. Na classificação binária, defina

$$Z_i = \mathbb{1}[h(x_i) \neq y_i].$$

Cada $Z_i \in \{0, 1\}$ e, para exemplos i.i.d.,

$$\mathbb{E}[Z_i] = R_P(h).$$

A média é exatamente o risco empírico:

$$\overline{Z} = \widehat{R}_D(h).$$

Portanto,

$$P_D\left(\left|\widehat{R}_D(h) - R_P(h)\right| \geq \varepsilon\right) \leq 2e^{-2N\varepsilon^2}.$$

Essa é uma garantia para uma hipótese fixa e independente de D. O subscrito D lembra que a aleatoriedade está na amostra.

5.2.4 Forma de intervalo

Queremos uma afirmação com probabilidade de falha no máximo δ:

$$2e^{-2N\varepsilon^2} \leq \delta.$$

Isolando ε,

$$\varepsilon \geq \sqrt{\frac{\log(2/\delta)}{2N}}.$$

Assim, com probabilidade de pelo menos $1 - \delta$,

$$\left|\widehat{R}_D(h) - R_P(h)\right| \leq \sqrt{\frac{\log(2/\delta)}{2N}}.$$

Equivalentemente,

$$R_P(h) \leq \widehat{R}_D(h) + \sqrt{\frac{\log(2/\delta)}{2N}}.$$

Essa é uma cota superior para o risco de uma hipótese previamente fixada.

5.2.5 Forma de complexidade amostral

Também podemos isolar N. Para garantir desvio no máximo ε com confiança $1-\delta$, basta

$$N \geq \frac{\log(2/\delta)}{2\varepsilon^2}.$$

Por exemplo, com

$$\varepsilon = 0{,}05, \qquad \delta = 0{,}05,$$

temos

$$N \geq \frac{\log 40}{2(0{,}05)^2} \approx 737{,}78.$$

Logo, 738 exemplos independentes são suficientes segundo essa cota para uma hipótese fixa. É uma garantia de pior caso; não afirma que toda aplicação necessitará exatamente dessa quantidade.

5.2.6 Exemplo numérico de intervalo

Suponha uma hipótese fixada avaliada em $N = 1\,000$ exemplos independentes, com erro observado

$$\widehat{R}_D(h) = 0{,}12.$$

Para $\delta = 0{,}05$,

$$\varepsilon = \sqrt{\frac{\log 40}{2\,000}} \approx 0{,}0429.$$

Com confiança de pelo menos 95%, Hoeffding garante

$$R_P(h) \leq 0{,}12 + 0{,}0429 = 0{,}1629.$$

A cota bilateral também fornece limite inferior, truncado em zero:

$$R_P(h) \in [0{,}0771,\ 0{,}1629].$$

Esse intervalo não é necessariamente o mais estreito para uma proporção binomial, mas é simples e não depende do valor desconhecido de $R_P(h)$.

5.2.7 Forma unilateral

Se precisamos apenas limitar o risco por cima, uma versão unilateral fornece

$$P_D\left(R_P(h) - \widehat{R}_D(h) \geq \varepsilon\right) \leq e^{-2N\varepsilon^2}.$$

Com probabilidade de pelo menos $1 - \delta$,

$$R_P(h) \leq \widehat{R}_D(h) + \sqrt{\frac{\log(1/\delta)}{2N}}.$$

A ausência do fator 2 melhora ligeiramente a cota porque controlamos somente uma direção do desvio.

5.2.8 Perdas limitadas

A aplicação não se restringe à perda zero-um. Se

$$0 \leq \ell(h(x), y) \leq B,$$

podemos normalizar $Z_i = \ell(h(x_i), y_i)/B \in [0, 1]$. Resulta

$$P_D\left(\left|\widehat{R}_D(h) - R_P(h)\right| \geq \varepsilon\right) \leq 2\exp\left(-\frac{2N\varepsilon^2}{B^2}\right).$$

Quanto maior o intervalo possível da perda, mais fraca é a concentração para o mesmo ε.

Perdas não limitadas, como a quadrática sem restrição nas previsões e alvos, exigem hipóteses adicionais ou outras desigualdades. Não podemos aplicar Hoeffding diretamente sem um limite quase certo.

5.2.9 O problema da hipótese escolhida

Em aprendizado, usamos

$$\hat{h} = A(D),$$

portanto a hipótese depende da mesma amostra. A desigualdade para uma h fixa não pode ser simplesmente reescrita como

$$P_D\left(\left|\widehat{R}_D(A(D)) - R_P(A(D))\right| \geq \varepsilon\right) \leq 2e^{-2N\varepsilon^2}.$$

Essa passagem seria injustificada: $A(D)$ foi selecionada justamente por seu comportamento em D.

Precisamos controlar todas as hipóteses que o algoritmo poderia escolher.

5.2.10 Classe finita e cota da união

Se

$$\mathcal{H} = \{h_1, \ldots, h_M\},$$

então o evento de falha uniforme é

$$\left\{\exists h \in \mathcal{H} : \left|\widehat{R}_D(h) - R_P(h)\right| \geq \varepsilon\right\}.$$

Pela cota da união,

$$\begin{aligned} &P_D\left(\exists h \in \mathcal{H} : \left|\widehat{R}_D(h) - R_P(h)\right| \geq \varepsilon\right) \\ &\quad \leq \sum_{h \in \mathcal{H}} P_D\left(\left|\widehat{R}_D(h) - R_P(h)\right| \geq \varepsilon\right) \\ &\quad \leq 2Me^{-2N\varepsilon^2}. \end{aligned}$$

No evento complementar, todas as hipóteses satisfazem a cota. Portanto, qualquer $A(D) \in \mathcal{H}$ também satisfaz.

Observe que, dentro da união, cada h é fixa. O evento não deve ser escrito com $h(D)$ para cada parcela.

5.2.11 Cota uniforme em forma de intervalo

Impondo

$$2Me^{-2N\varepsilon^2} \le \delta,$$

obtemos

$$\varepsilon \ge \sqrt{\frac{\log(2M/\delta)}{2N}}.$$

Com probabilidade de pelo menos $1-\delta$,

$$\sup_{h\in\mathcal{H}} \left|\widehat{R}_D(h) - R_P(h)\right| \le \sqrt{\frac{\log(2M/\delta)}{2N}}.$$

Para a hipótese selecionada,

$$R_P(A(D)) \le \widehat{R}_D(A(D)) + \sqrt{\frac{\log(2M/\delta)}{2N}}.$$

O preço de examinar M hipóteses aparece como $\log M$ depois que isolamos ε.

5.2.12 Complexidade amostral para classe finita

Isolando N,

$$N \ge \frac{\log(2M/\delta)}{2\varepsilon^2}$$

é suficiente para desvio uniforme no máximo ε.

Se $M = 1$, recuperamos a cota para uma hipótese fixa. Se multiplicarmos o número de hipóteses por mil, adicionamos $\log 1\,000$ ao numerador, não um fator mil ao tamanho necessário.

5.2.13 Exemplo com seleção

Considere $M = 100$ hipóteses, $N = 2\,000$ e $\delta = 0{,}05$. A largura uniforme é

$$\varepsilon = \sqrt{\frac{\log(2 \cdot 100/0{,}05)}{2 \cdot 2\,000}} = \sqrt{\frac{\log 4\,000}{4\,000}} \approx 0{,}0455.$$

Se a hipótese selecionada tem erro empírico $0{,}08$, então

$$R_P(A(D)) \leq 0{,}1255$$

com confiança de pelo menos 95%, sob as condições da desigualdade.

5.2.14 O equilíbrio entre ajuste e capacidade

Uma classe maior pode conter uma hipótese com erro empírico menor, mas aumenta a penalidade de complexidade:

$$\widehat{R}_D(h) + \sqrt{\frac{\log(2M/\delta)}{2N}}.$$

Isso sugere uma ideia de **minimização estrutural do risco**: comparar não apenas o ajuste, mas ajuste mais uma penalidade de capacidade.

Não há uma oposição absoluta entre "erro empírico" e "erro verdadeiro". O objetivo é escolher uma classe expressiva o suficiente para representar padrões, mas controlada o suficiente para generalizar com os dados disponíveis.

5.2.15 Limitações

Hoeffding não resolve todos os problemas:

- exige independência;
- exige variáveis limitadas;
- a cota pode ser conservadora;
- a união direta não ajuda quando $\mathcal{H}$ é infinita;
- não trata mudança de distribuição;
- não corrige seleção feita fora da classe contabilizada.

Para classes infinitas, substituiremos M por uma medida do número de comportamentos relevantes sobre amostras finitas. Esse caminho leva à função de crescimento e à dimensão VC.

5.2.16 Exemplo computacional

```
from math import log, sqrt



def raio_hoeffding(n, delta, numero_hipoteses=1):
```

```
    return sqrt(log(2 * numero_hipoteses / delta) / (2 * n))


print(raio_hoeffding(1000, 0.05))
print(raio_hoeffding(2000, 0.05, numero_hipoteses=100))
```

O cálculo supõe perda em $[0, 1]$. Para perda em $[0, B]$, multiplique o raio por B.

5.2.17 Exercícios de fixação

1. Enuncie as condições da desigualdade de Hoeffding.
2. Calcule a cota bilateral para $N = 500$ e $\varepsilon = 0{,}1$.
3. Isole ε em $2e^{-2N\varepsilon^2} \leq \delta$.
4. Quantos exemplos são suficientes para $\varepsilon = 0{,}02$ e $\delta = 0{,}01$ em uma hipótese fixa?
5. Por que Hoeffding não se aplica diretamente a $A(D)$?
6. Derive a cota $2Me^{-2N\varepsilon^2}$ usando a união.
7. Para $M = 50$, $N = 1\,000$ e $\delta = 0{,}05$, calcule o raio uniforme.
8. Explique por que a dependência em M se torna logarítmica ao isolar ε.
9. Dê um exemplo de perda não limitada à qual Hoeffding não pode ser aplicada diretamente.
10. Compare as formas bilateral e unilateral.

5.3 A dimensão de Vapnik–Chervonenkis

Para uma classe finita, a cota de Hoeffding uniforme contém $\log |\mathcal{H}|$. Classes importantes, porém, possuem infinitas hipóteses: há infinitas retas no plano e infinitas escolhas de pesos reais.

A cardinalidade infinita superestima o número de comportamentos relevantes em uma amostra finita. Duas retas diferentes podem classificar os mesmos N pontos exatamente da mesma maneira. Para generalização, interessa contar as **rotulações distintas** que a classe consegue produzir sobre conjuntos finitos.

5.3.1 Restrições e dicotomias

Seja uma classe de classificadores binários

$$\mathcal{H} \subseteq \{-1, +1\}^X$$

e um conjunto de pontos distintos

$$S = \{x_1, \dots, x_N\} \subseteq X.$$

A restrição de h a S é o vetor

$$h|_S = (h(x_1), \dots, h(x_N)) \in \{-1, +1\}^N.$$

O conjunto de dicotomias realizadas por $\mathcal{H}$ em S é

$$\mathcal{H}|_S = \{(h(x_1), \dots, h(x_N)) \mid h \in \mathcal{H}\}.$$

Embora $\mathcal{H}$ possa ser infinita,

$$|\mathcal{H}|_S| \leq 2^N,$$

pois existem apenas 2^N rotulações binárias dos N pontos.

Duas hipóteses que diferem fora de S, mas coincidem em todos os pontos de S, contam como uma única dicotomia sobre esse conjunto.

5.3.2 Função de crescimento

A **função de crescimento** é o maior número de dicotomias que a classe consegue realizar em qualquer conjunto de N pontos:

$$m_{\mathcal{H}}(N) = \max_{\substack{S \subseteq X \\ |S|=N}} |\mathcal{H}|_S|.$$

Também é comum usar a notação $\Pi_{\mathcal{H}}(N)$. Sempre vale

$$1 \leq m_{\mathcal{H}}(N) \leq 2^N.$$

O máximo é tomado sobre a posição dos pontos. Um conjunto particular pode admitir poucas rotulações mesmo quando outro conjunto do mesmo tamanho admite todas.

5.3.3 Estilhaçamento

Dizemos que $\mathcal{H}$ **estilhaça** S quando realiza todas as rotulações possíveis:

$$|\mathcal{H}|_S| = 2^{|S|}.$$

Equivalentemente,

$$\forall \mathbf{y} \in \{-1, +1\}^{|S|}, \quad \exists h \in \mathcal{H} \quad \text{tal que} \quad h|_S = \mathbf{y}.$$

A ordem dos quantificadores é essencial: fixamos um único conjunto S e exigimos que, para **cada** rotulação, possa existir uma hipótese diferente.

Se a classe estilhaça algum conjunto de N pontos, então

$$m_{\mathcal{H}}(N) = 2^N.$$

Se nenhum conjunto de N pontos é estilhaçado, então

$$m_{\mathcal{H}}(N) < 2^N.$$

5.3.4 Dimensão VC

A **dimensão de Vapnik–Chervonenkis** é o maior tamanho de um conjunto estilhaçado:

$$d_{\text{VC}}(\mathcal{H}) = \sup \left\{ N \in \mathbb{N} \mid m_{\mathcal{H}}(N) = 2^N \right\}.$$

Se conjuntos arbitrariamente grandes podem ser estilhaçados, definimos

$$d_{\text{VC}}(\mathcal{H}) = \infty.$$

Se k é o menor inteiro para o qual

$$m_{\mathcal{H}}(k) < 2^k,$$

ele é chamado de **ponto de interrupção**. Quando existe,

$$d_{\text{VC}}(\mathcal{H}) = k - 1.$$

Para provar $d_{\text{VC}}(\mathcal{H}) = d$, precisamos de duas partes:

1. **cota inferior:** exibir um conjunto de d pontos que é estilhaçado;
2. **cota superior:** provar que nenhum conjunto de $d+1$ pontos pode ser estilhaçado.

Encontrar uma rotulação impossível em um conjunto particular fornece informação sobre aquele conjunto, mas não prova sozinho a cota superior para todos os conjuntos.

5.3.5 Exemplo 1: limiares na reta

Considere

$$\mathcal{H}_{\text{limiar}} = \left\{ h_a(x) = \begin{cases} -1, & x < a, \\ +1, & x \geq a \end{cases} \middle| \; a \in \mathbb{R} \right\}.$$

Para N pontos ordenados,

$$x_1 < x_2 < \cdots < x_N,$$

o limiar pode ser colocado antes de todos, entre dois pontos consecutivos ou depois de todos. Existem $N+1$ dicotomias:

$$m_{\mathcal{H}_{\text{limiar}}}(N) = N + 1.$$

Um ponto é estilhaçado, pois podemos classificá-lo como -1 ou $+1$. Dois pontos não são estilhaçados: a rotulação $(+1, -1)$ contradiz a monotonicidade do limiar. Logo,

$$d_{\text{VC}}(\mathcal{H}_{\text{limiar}}) = 1.$$

Se permitíssemos ambas as orientações do limiar, a classe seria diferente e a contagem precisaria ser refeita.

5.3.6 Exemplo 2: intervalos na reta

Agora considere classificadores positivos dentro de um intervalo:

$$h_{a,b}(x) = \begin{cases} +1, & a \leq x \leq b, \\ -1, & \text{caso contrário.} \end{cases}$$

Sobre N pontos ordenados, os positivos formam um bloco consecutivo. Há

$$\frac{N(N+1)}{2}$$

blocos não vazios e uma rotulação sem ponto positivo. Portanto,

$$m_{\mathcal{H}_{\text{int}}}(N) = \frac{N(N+1)}{2} + 1.$$

Dois pontos podem receber qualquer uma das quatro rotulações. Três pontos não podem realizar $(+1, -1, +1)$, pois um intervalo que contém os extremos também contém o ponto central. Assim,

$$d_{\text{VC}}(\mathcal{H}_{\text{int}}) = 2.$$

5.3.7 Exemplo 3: retas no plano

Considere semiplanos afins em $\mathbb{R}^2$:

$$\mathcal{H}_{\text{reta}} = \left\{ \operatorname{sign}(\mathbf{w}^{\mathsf{T}}\mathbf{x} + b) \mid \mathbf{w} \in \mathbb{R}^2,\ b \in \mathbb{R} \right\}.$$

Três pontos não colineares podem ser estilhaçados. Para qualquer rotulação, uma reta separa os positivos dos negativos.

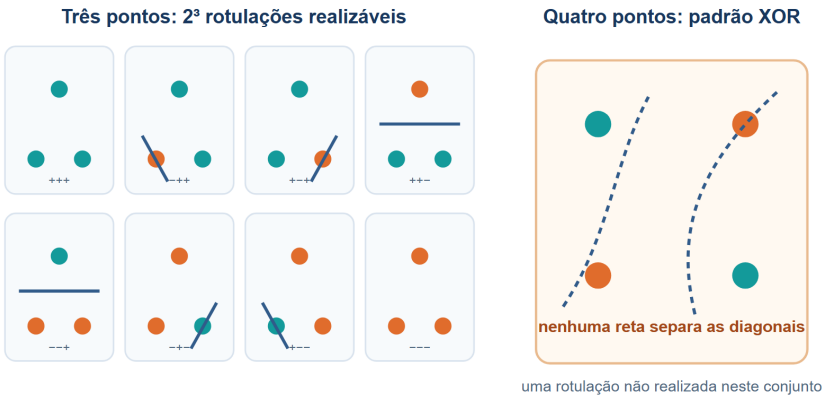


Figura 5.3: Retas realizam todas as oito rotulações de três pontos não colineares; a rotulação XOR em quatro vértices é um exemplo não separável.

Para a cota superior, considere qualquer conjunto de quatro pontos.

- Se um ponto está no interior do triângulo formado pelos outros três, rotule o ponto interno como positivo e os demais como negativos. Nenhum semiplano o separa.
- Se os quatro pontos estão em posição convexa, rotule vértices alternados como positivos. Surge o padrão XOR, que não é linearmente separável.
- Degenerações colineares não aumentam a capacidade.

Logo, nenhum conjunto de quatro pontos é estilhaçado e

$$d_{\mathrm{VC}}(\mathcal{H}_{\text{reta}}) = 3.$$

Para quatro pontos em posição geral, retas realizam 14 das 16 rotulações: faltam as duas rotulações XOR com diagonais opostas.

5.3.8 Exemplo 4: semiplanos em $\mathbb{R}^d$

Para classificadores afins

$$h_{\mathbf{w},b}(\mathbf{x}) = \operatorname{sign}(\mathbf{w}^{\mathsf{T}}\mathbf{x} + b)$$

em $\mathbb{R}^d$,

$$d_{\mathrm{VC}} = d + 1.$$

A cota inferior pode ser demonstrada usando $d + 1$ pontos afim-independentes. A cota superior decorre da dependência afim de qualquer conjunto com $d + 2$ pontos e de um argumento de separação.

Ao incorporar uma coordenada constante,

$$\tilde{\mathbf{x}} = (1, \mathbf{x}),$$

o viés b passa a fazer parte de um vetor de $d + 1$ pesos. Isso ajuda a compreender o valor $d + 1$, mas contar parâmetros é apenas uma intuição, não uma prova geral de dimensão VC.

5.3.9 Exemplo 5: conjuntos convexos no plano

Considere a classe de todos os subconjuntos convexos de $\mathbb{R}^2$, com rótulo positivo dentro do conjunto.

Para qualquer N, escolha N pontos sobre uma circunferência. Dada uma rotulação, tome o fecho convexo dos pontos positivos. Nenhum ponto negativo da circunferência pertence a esse fecho, porque cada ponto é um vértice extremo da configuração. Os casos de todos positivos ou nenhum positivo também são realizáveis.

Portanto, a classe estilhaça conjuntos de qualquer tamanho:

$$d_{\text{VC}}(\mathcal{H}_{\text{conv}}) = \infty.$$

Esse exemplo mostra que uma descrição geométrica simples em palavras pode ter capacidade ilimitada.

5.3.10 Comparação dos exemplos

Classe	Função de crescimento	Dimensão VC
limiar orientado em $\mathbb{R}$	$N+1$	1
intervalo em $\mathbb{R}$	$\frac{N(N+1)}{2}+1$	2
semiplano afim em $\mathbb{R}^2$	2^N até $N=3$; $m(4)=14$	3
semiplano afim em $\mathbb{R}^d$	crescimento polinomial após $d+1$	$d+1$
conjuntos convexos em $\mathbb{R}^2$	2^N para todo N	∞

A função de crescimento contém mais informação que um único número. A dimensão VC identifica o ponto a partir do qual o crescimento deixa de ser máximo; o próximo resultado mostrará que, depois desse ponto, o crescimento é limitado por um polinômio em N.

5.3.11 Dimensão VC não é apenas número de parâmetros

Em alguns modelos lineares, dimensão VC e quantidade de parâmetros têm a mesma ordem. Isso não é uma lei universal. Restrições algébricas, parâmetros redundantes, precisão numérica e estrutura computacional alteram a capacidade.

Além disso, dimensão VC mede rotulações possíveis, não dificuldade de otimização, velocidade de treinamento ou qualidade da representação.

5.3.12 Como calcular uma dimensão VC

Um roteiro prático é:

1. escolha candidatos a conjuntos pequenos em posição favorável;
2. prove que todas as rotulações de um conjunto de tamanho d são realizáveis;
3. considere uma configuração arbitrária de $d + 1$ pontos;
4. construa, para cada possível configuração geométrica, ao menos uma rotulação impossível;
5. conclua as cotas inferior e superior.

Diagramas ajudam a descobrir a resposta, mas cada quantificador precisa aparecer na prova.

5.3.13 Exercícios de fixação

1. Liste as dicotomias de três pontos ordenados produzidas por limiares orientados.
2. Verifique a fórmula da função de crescimento de intervalos para $N = 3$.
3. Mostre explicitamente como três pontos não colineares realizam todas as oito rotulações por retas.
4. Explique por que o padrão XOR não é linearmente separável.
5. Complete o argumento da cota superior para quatro pontos quando um deles está dentro do triângulo dos outros.
6. Por que pontos sobre uma circunferência podem ser estilhaçados por conjuntos convexos?
7. Diferencie função de crescimento, ponto de interrupção e dimensão VC.
8. Dê um exemplo de afirmação em que contar parâmetros seria apenas heurístico.

5.3.14 Cota do número efetivo de hipóteses sobre uma amostra

A função de crescimento sempre satisfaz

$$m_{\mathcal{H}}(N) \leq 2^N.$$

Essa cota é exata quando a classe estilhaça algum conjunto de N pontos, mas é inútil para demonstrar controle de capacidade: substituir M por 2^N em uma cota de união introduz um termo linear em N no expoente e pode impedir a convergência desejada.

A dimensão VC finita produz uma mudança decisiva. Depois do primeiro ponto de interrupção, a função de crescimento não pode continuar exponencial; ela passa a ser limitada por um polinômio.

5.3.15 Lema de Sauer–Shelah

Se

$$d = d_{\text{VC}}(\mathcal{H}) < \infty,$$

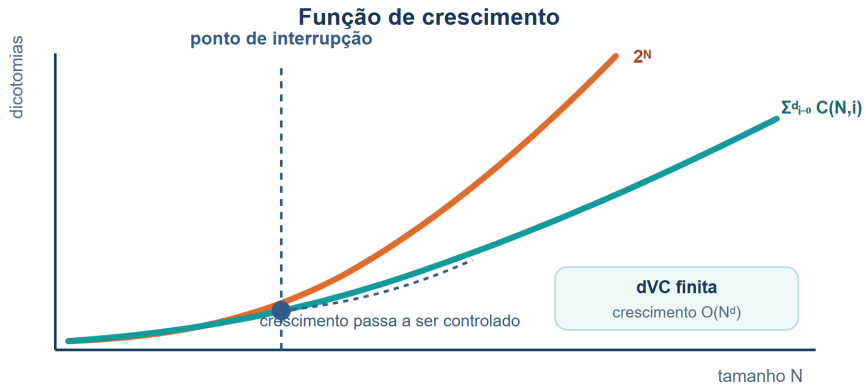


Figura 5.4: Com dimensão VC finita, a função de crescimento deixa de acompanhar 2^N após o ponto de interrupção e fica limitada por uma soma polinomial.

então, para todo N,

$$m_{\mathcal{H}}(N) \leq \sum_{i=0}^{d} \binom{N}{i}.$$

Adotamos $\binom{N}{i} = 0$ quando $i > N$. Para $N \leq d$, a soma é

$$\sum_{i=0}^{N} \binom{N}{i} = 2^N,$$

compatível com a possibilidade de estilhaçamento. Para $N > d$, a soma contém apenas os termos até d e cresce como um polinômio de grau d.

Se $k = d + 1$ é o menor ponto de interrupção, a mesma cota pode ser escrita

$$m_{\mathcal{H}}(N) \leq \sum_{i=0}^{k-1} \binom{N}{i}.$$

5.3.16 Exemplos da cota

Para $d = 1$,

$$m_{\mathcal{H}}(N) \leq \binom{N}{0} + \binom{N}{1} = 1 + N.$$

A classe de limiares orientados atinge exatamente esse valor.

Para $d = 2$,

$$m_{\mathcal{H}}(N) \leq 1 + N + \frac{N(N-1)}{2}.$$

A classe de intervalos possui

$$m_{\text{int}}(N) = 1 + \frac{N(N+1)}{2},$$

que é maior do que a expressão de Sauer com $d = 2$? Vamos comparar cuidadosamente:

$$1 + N + \frac{N(N-1)}{2} = 1 + \frac{N(N+1)}{2}.$$

As expressões são iguais. Portanto, intervalos também atingem a cota de Sauer.

Para semiplanos no plano, $d = 3$:

$$m_{\mathcal{H}}(N) \leq 1 + N + \binom{N}{2} + \binom{N}{3}.$$

Em $N = 4$,

$$m_{\mathcal{H}}(4) \leq 1 + 4 + 6 + 4 = 15.$$

A função de crescimento real de semiplanos afins em posição geral é 14, portanto a cota é válida, mas não exata nesse ponto.

5.3.17 Ideia combinatória da prova

Defina $M(N, d)$ como o maior número de dicotomias possível em N pontos para uma classe de dimensão VC no máximo d. A prova usa a recorrência

$$M(N, d) \leq M(N-1, d) + M(N-1, d-1).$$

Para compreender a recorrência, separe o último ponto x_N. Ao restringir as dicotomias aos primeiros $N - 1$ pontos, há dois tipos:

1. padrões que admitem apenas uma extensão para x_N;
2. padrões que admitem as duas extensões, -1 e $+1$.

O primeiro grupo contribui no máximo $M(N-1,d)$. No segundo grupo, se fosse possível estilhaçar d pontos entre os primeiros $N-1$, acrescentar x_N produziria um conjunto de $d+1$ pontos estilhaçado. Portanto, a classe dos padrões com duas extensões possui dimensão no máximo $d-1$ e contribui no máximo $M(N-1,d-1)$.

Os valores de fronteira são

$$M(N,0) = 1$$

e

$$M(0,d) = 1.$$

Aplicando a recorrência repetidamente e usando a identidade de Pascal,

$$\binom{N}{i} = \binom{N-1}{i} + \binom{N-1}{i-1},$$

obtemos

$$M(N,d) \leq \sum_{i=0}^{d} \binom{N}{i}.$$

Essa é a estrutura central da prova de Sauer–Shelah.

5.3.18 Cota simplificada

Para $1 \leq d \leq N$, vale a cota padrão

$$\sum_{i=0}^{d} \binom{N}{i} \leq \left(\frac{eN}{d}\right)^{d}.$$

Logo,

$$m_{\mathcal{H}}(N) \leq \left(\frac{eN}{d}\right)^{d}.$$

Tomando logaritmos,

$$\log m_{\mathcal{H}}(N) \leq d \log \left(\frac{eN}{d} \right).$$

Essa forma será útil na cota de generalização, pois a penalidade depende do logaritmo do número efetivo de dicotomias.

Para d fixo,

$$m_{\mathcal{H}}(N) = O(N^d),$$

enquanto

$$2^N$$

cresce exponencialmente. A diferença entre crescimento polinomial e exponencial é o motivo pelo qual dimensão VC finita permite aprendizado uniforme.

5.3.19 Uma cota ainda mais simples

Também podemos usar, para $N \geq 1$ e $d \geq 1$, uma estimativa grosseira como

$$m_{\mathcal{H}}(N) \leq (d+1)N^d.$$

Ela perde constantes e dependência refinada em d, mas evidencia o grau polinomial. A expressão $(eN/d)^d$ costuma ser preferível em derivações quantitativas.

5.3.20 Ponto de interrupção e efeito dominó

O lema contém uma mensagem combinatória forte: basta que a classe falhe em estilhaçar todos os conjuntos de algum tamanho finito para que o crescimento futuro inteiro fique controlado.

Não é possível ter

$$m_{\mathcal{H}}(k) < 2^k$$

e depois retornar ao crescimento máximo

$$m_{\mathcal{H}}(N) = 2^N$$

para algum $N > k$. Se um conjunto maior fosse estilhaçado, qualquer subconjunto de tamanho k também seria estilhaçado, contradizendo o ponto de interrupção.

5.3.21 Cálculo numérico

Considere $d = 3$ e $N = 20$. Sauer–Shelah fornece

$$m_{\mathcal{H}}(20) \leq \binom{20}{0} + \binom{20}{1} + \binom{20}{2} + \binom{20}{3}.$$

Calculando,

$$m_{\mathcal{H}}(20) \leq 1 + 20 + 190 + 1\,140 = 1\,351.$$

Sem usar capacidade, a cota trivial seria

$$2^{20} = 1\,048\,576.$$

A redução do número efetivo de comportamentos é enorme.

A cota simplificada dá

$$\left(\frac{20e}{3}\right)^3 \approx 5\,950,$$

menos precisa que $1\,351$, mas ainda muito menor do que 2^{20}.

5.3.22 Relação com compressão de padrões

A soma

$$\sum_{i=0}^{d} \binom{N}{i}$$

conta subconjuntos de até d elementos. Isso sugere que uma classe de dimensão VC d não consegue tomar decisões independentes em todas as N posições; seus padrões são determinados por uma estrutura combinatória de ordem d.

Essa interpretação não significa que toda classe VC admita automaticamente um esquema simples de compressão com exatamente d exemplos. A relação formal entre dimensão VC e compressão de amostras é mais delicada. A soma binomial é, antes de tudo, uma cota combinatória.

5.3.23 O que o lema não diz

Sauer–Shelah limita o número de dicotomias, mas não:

- encontra a hipótese ótima;
- garante que o algoritmo seja eficiente;
- mede erro de aproximação;
- trata regressão contínua diretamente;
- fornece sozinho uma probabilidade de generalização.

Para obter uma cota de erro, combinaremos a função de crescimento com uma desigualdade de concentração e um argumento de uniformização.

5.3.24 Exemplo computacional

```
from math import comb


def cota_sauer(n, d):
    return sum(comb(n, i) for i in range(min(d, n) + 1))


print(cota_sauer(20, 3))  # 1351
print(2 ** 20)             # 1048576
```

5.3.25 Exercícios de fixação

1. Calcule a cota de Sauer para $N = 6$ e $d = 2$.
2. Verifique que a fórmula para $d = 1$ coincide com a função de crescimento dos limiares.
3. Mostre algebricamente que $1 + N + \binom{N}{2} = 1 + N(N+1)/2$.
4. Explique a recorrência $M(N, d) \leq M(N-1, d) + M(N-1, d-1)$.
5. Use a identidade de Pascal para verificar que a soma binomial satisfaz a mesma recorrência.
6. Compare numericamente 2^N e a cota de Sauer para $N = 10$, $d = 3$.
7. Por que uma classe não pode voltar a estilhaçar conjuntos maiores depois de um ponto de interrupção?

8. Diferencie a cota exata pela soma binomial da aproximação $(eN/d)^d$.
9. Explique por que o lema de Sauer–Shelah ainda não é, sozinho, uma cota de generalização.

5.3.26 Cota de erro

Até aqui, a dimensão VC e a função de crescimento mediram **quantas formas diferentes uma classe de hipóteses pode produzir sobre uma amostra**. Falta transformar essa medida combinatória em uma afirmação estatística: quão próximo o erro observado nos dados está do erro que a hipótese terá em exemplos futuros?

Seja

$$\Delta_D(h) = E_{\text{out}}(h) - E_{\text{in}}(h),$$

em que E_{in} é o erro na amostra de treinamento D e E_{out} é o erro esperado na distribuição que gera os dados. O módulo $|\Delta_D(h)|$ é chamado **lacuna de generalização**.

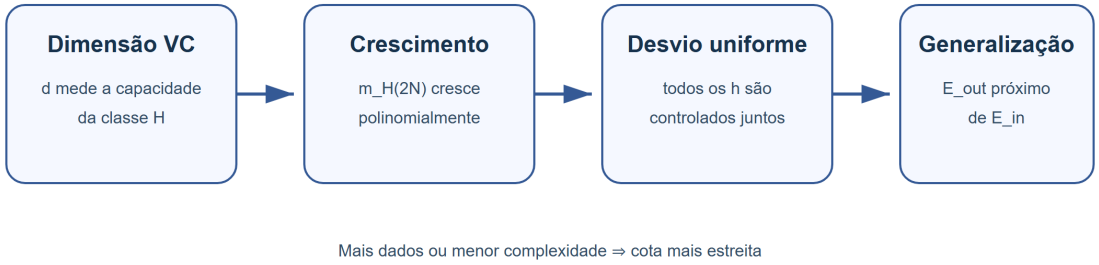


Figura 5.5: A dimensão VC limita a função de crescimento, que permite controlar simultaneamente a lacuna de todas as hipóteses.

5.3.26.1 Uma cota uniforme

Uma forma clássica da desigualdade de Vapnik–Chervonenkis afirma que, para qualquer $\epsilon > 0$,

$$\Pr\left[\sup_{h\in\mathcal{H}} |E_{\text{out}}(h) - E_{\text{in}}(h)| > \epsilon\right] \leq 4\, m_{\mathcal{H}}(2N) \exp\left(-\frac{N\epsilon^2}{8}\right). \tag{5.1}$$

As constantes podem variar entre versões do teorema, mas a estrutura é sempre a mesma: uma parcela mede a **complexidade** de $\mathcal{H}$ e a exponencial mede o efeito benéfico do número N de exemplos.

O supremo é essencial. A desigualdade não controla apenas uma hipótese fixada antes de observar os dados; ela controla **simultaneamente todas as hipóteses** da classe. Portanto, continua válida para a hipótese $g = A(D)$ escolhida pelo algoritmo depois de examinar a amostra. Esse é justamente o passo que uma aplicação direta da desigualdade de Hoeffding não consegue justificar.

Fixando uma probabilidade de falha $\delta \in (0, 1)$ e isolando ϵ em Equação 5.1, obtemos, com probabilidade pelo menos $1 - \delta$,

$$\sup_{h \in \mathcal{H}} |E_{\text{out}}(h) - E_{\text{in}}(h)| \leq \sqrt{\frac{8}{N} \ln\left(\frac{4m_{\mathcal{H}}(2N)}{\delta}\right)}. \tag{5.2}$$

Em particular, a hipótese devolvida pelo algoritmo satisfaz

$$E_{\text{out}}(g) \leq E_{\text{in}}(g) + \sqrt{\frac{8}{N} \ln\left(\frac{4m_{\mathcal{H}}(2N)}{\delta}\right)}.$$

Assim, a cota separa dois objetivos: o algoritmo procura reduzir $E_{\text{in}}(g)$, enquanto a teoria limita a distância entre esse valor e $E_{\text{out}}(g)$.

5.3.26.2 Inserindo a dimensão VC

Se $d = d_{\text{VC}}(\mathcal{H}) < \infty$, o lema de Sauer–Shelah fornece

$$m_{\mathcal{H}}(2N) \leq \sum_{i=0}^{d} \binom{2N}{i} \leq \left(\frac{2eN}{d}\right)^{d}, \qquad 2N \geq d.$$

Substituindo essa estimativa em Equação 5.2, chegamos à cota

$$|E_{\text{out}}(g) - E_{\text{in}}(g)| \leq \sqrt{\frac{8}{N}\left[d \ln\left(\frac{2eN}{d}\right) + \ln\left(\frac{4}{\delta}\right)\right]}. \tag{5.3}$$

Essa expressão explicita três relações importantes:

- aumentar N tende a estreitar a cota;
- aumentar d torna a classe mais flexível e alarga a cota;
- reduzir δ aumenta a confiança, mas também torna a garantia mais conservadora.

Para d fixo, o lado direito tem ordem aproximada

$$O\left(\sqrt{\frac{d \ln N + \ln(1/\delta)}{N}}\right),$$

que converge para zero quando $N \to \infty$. A convergência ocorre porque a função de crescimento é polinomial. Se ela permanecesse igual a 2^{2N}, seu logaritmo seria proporcional a N e a cota não desapareceria.

5.3.26.3 Quantos exemplos são necessários?

Para exigir uma lacuna de no máximo ϵ, basta procurar N tal que

$$N \geq \frac{8}{\epsilon^2}\left[d \ln\left(\frac{2eN}{d}\right) + \ln\left(\frac{4}{\delta}\right)\right]. \tag{5.4}$$

A desigualdade é **implícita**, pois N também aparece dentro do logaritmo. Em um programa, podemos encontrar o menor inteiro que a satisfaz por busca incremental ou binária. Uma iteração de ponto fixo também fornece uma estimativa rápida:

```
import math

def amostras_vc(d, epsilon, delta, n_inicial=1):
    n = max(n_inicial, math.ceil(d / 2))
    while True:
        rhs = (8 / epsilon**2) * (
            d * math.log(2 * math.e * n / d)
            + math.log(4 / delta)
        )
        novo_n = math.ceil(rhs)
        if novo_n <= n:
            return n
        n = novo_n

print(amostras_vc(d=3, epsilon=0.1, delta=0.1))
```

O resultado deve ser interpretado como uma **condição suficiente**, não como o número exato de exemplos de que todo problema precisará. Cotas VC são independentes da distribuição e do algoritmo; por cobrirem até os casos mais desfavoráveis, frequentemente são conservadoras.

> **!** Dimensão VC não é simplesmente número de parâmetros
>
> Em alguns modelos lineares, a dimensão VC é próxima do número de parâmetros livres. Isso não constitui uma regra universal. Restrições, arquitetura, margem, regularização e até a forma de parametrização podem alterar a capacidade efetiva. O objeto matemático relevante é a classe de funções realizáveis, não apenas o tamanho do vetor de parâmetros.

5.3.26.4 Da generalização ao aprendizado PAC

Suponha que o algoritmo encontre g com erro de treinamento pequeno. Se a classe tem dimensão VC finita e N satisfaz Equação 5.4, então, com alta probabilidade, o erro fora da amostra também será pequeno. No caso realizável, em que existe uma hipótese perfeita na classe e o algoritmo encontra $E_{\text{in}}(g) = 0$, Equação 5.3 fornece diretamente uma garantia para $E_{\text{out}}(g)$.

No caso não realizável, é útil decompor o desempenho em duas partes:

$$E_{\text{out}}(g) = \underbrace{E_{\text{in}}(g)}_{\text{qualidade do ajuste}} + \underbrace{\left(E_{\text{out}}(g) - E_{\text{in}}(g)\right)}_{\text{lacuna de generalização}}.$$

A teoria VC controla a segunda parte; otimização e escolha do modelo tratam da primeira. Uma cota estreita não compensa um modelo que ajusta mal os dados, assim como erro de treinamento zero não garante sozinho boa generalização.

5.3.26.5 Exemplo de leitura da cota

Considere duas classes treinadas com a mesma amostra. A classe $\mathcal{H}_1$ tem $d = 5$ e a classe $\mathcal{H}_2$ tem $d = 500$. Se ambas alcançam o mesmo erro de treinamento, a cota favorece $\mathcal{H}_1$, pois ela oferece menos maneiras de se adaptar acidentalmente ao ruído. Entretanto, se $\mathcal{H}_1$ apresenta erro de treinamento muito alto, $\mathcal{H}_2$ pode produzir a melhor soma entre ajuste e complexidade. Essa tensão é uma formulação matemática do compromisso entre **subajuste** e **sobreajuste**.

5.3.26.6 Recapitulação

1. A desigualdade VC controla uniformemente todas as hipóteses de $\mathcal{H}$.
2. O lema de Sauer–Shelah converte dimensão VC finita em crescimento polinomial.
3. A lacuna garantida decresce aproximadamente como $\sqrt{(d \ln N + \ln(1/\delta))/N}$.
4. A complexidade amostral cresce com a capacidade d, com a precisão desejada $1/\epsilon$ e com a confiança desejada $1 - \delta$.
5. A cota é uma garantia de pior caso; ela orienta o raciocínio, mas não substitui validação empírica.

5.3.26.7 Exercícios

1. Explique por que o supremo sobre $h \in \mathcal{H}$ permite aplicar a cota à hipótese escolhida depois de observar os dados.
2. Mantendo d e δ fixos, o que acontece aproximadamente com a cota quando N é quadruplicado?
3. Implemente a função `amostras_vc` e compare os resultados para $d \in \{5, 50, 500\}$.
4. Mostre que, para d fixo, $(d \ln N)/N \to 0$.
5. Dê um exemplo em que escolher a classe de menor dimensão VC provoque subajuste.

5.4 Exercícios de múltipla escolha

Cada questão possui uma única resposta correta. Tente relacionar a alternativa escolhida às definições e cotas estudadas no capítulo.

1. A lacuna de generalização absoluta de uma hipótese h é:

   a. $|E_{\text{out}}(h) - E_{\text{in}}(h)|$.

   b. $E_{\text{out}}(h) + E_{\text{in}}(h)$.

   c. $E_{\text{in}}(h)^2$.

   d. o número de parâmetros de h.

2. Por que uma cota para uma hipótese fixada antes dos dados não basta para a hipótese selecionada por treinamento?

   a. Porque o treinamento escolhe a hipótese com base na mesma amostra.

   b. Porque hipóteses treinadas não possuem erro.

   c. Porque a amostra deixa de conter variáveis aleatórias.

   d. Porque Hoeffding se aplica somente à regressão.

3. A desigualdade de Hoeffding, na forma estudada, requer variáveis:

   a. independentes e limitadas.

   b. sempre gaussianas e negativas.

c. determinísticas e ilimitadas.

d. necessariamente vetoriais.

4. Ao quadruplicar N, mantendo os outros termos fixos, uma cota proporcional a $1/\sqrt{N}$ é aproximadamente:

a. duplicada.

b. reduzida à metade.

c. quadruplicada.

d. inalterada.

5. Para uma classe finita com M hipóteses, a cota da união introduz tipicamente qual dependência na complexidade?

a. $\ln M$.

b. $1/M^2$.

c. 2^M dentro do erro.

d. Nenhuma dependência em M.

6. Um conjunto é estilhaçado por $\mathcal{H}$ quando:

a. apenas uma rotulação pode ser produzida.

b. todas as rotulações binárias de seus pontos podem ser realizadas por hipóteses de $\mathcal{H}$.

c. todos os pontos recebem obrigatoriamente o mesmo rótulo.

d. sua cardinalidade é infinita.

7. A dimensão VC da classe de limiares em uma reta é:

a. 0.
b. 1.
c. 2.

d. infinita.

8. A dimensão VC da classe de intervalos em uma reta é:

   a. 1.
   b. 2.
   c. 3.
   d. infinita.

9. O lema de Sauer–Shelah mostra que, quando d_{VC} é finita, a função de crescimento para amostras grandes é limitada por um crescimento:

   a. polinomial em N.

   b. necessariamente constante.

   c. maior que todas as exponenciais.

   d. independente de N.

10. Qual afirmação interpreta corretamente uma cota VC?

    a. Ela prevê exatamente o erro de teste de qualquer experimento.

    b. Ela é uma garantia de pior caso e não substitui validação empírica.

    c. Ela prova que modelos mais simples são sempre melhores.

    d. Ela elimina o erro de aproximação.

i Gabarito comentado

1. **a.** O valor absoluto mede o tamanho da diferença sem depender de seu sinal.
2. **a.** A seleção adapta h às flutuações da amostra, exigindo controle simultâneo sobre toda a classe.
3. **a.** Independência e limitação são condições centrais da forma apresentada.
4. **b.** $1/\sqrt{4N} = 1/(2\sqrt{N})$.
5. **a.** A aplicação da união às M hipóteses aparece como $\ln M$ ao isolar a tolerância.
6. **b.** Estilhaçar significa realizar as 2^N dicotomias possíveis naquele conjunto.
7. **b.** Um ponto pode receber ambos os rótulos, mas dois pontos não admitem todas as dicotomias com um único limiar orientado.

8. **b.** Intervalos estilhaçam dois pontos; três pontos alternados não podem ser rotulados positivo–negativo–positivo por um só intervalo.
9. **a.** Para $N > d_{VC}$, o número efetivo de dicotomias deixa de crescer como 2^N e recebe uma cota polinomial.
10. **b.** A cota explica dependências e oferece garantia probabilística, mas pode ser conservadora em uma aplicação concreta.

Capítulo 6

Hipóteses Lineares

Modelos lineares estão entre as ferramentas mais importantes do aprendizado de máquina. Eles são matematicamente simples, eficientes e interpretáveis, mas também servem de bloco básico para modelos mais complexos. Uma unidade de uma rede neural, por exemplo, começa calculando a mesma soma ponderada que estudaremos neste capítulo.

O termo *linear* não significa que a saída final precise ser uma reta. Significa que o modelo começa combinando os atributos de entrada por uma **pontuação linear**. Para um exemplo $x = (x_1, \dots, x_d) \in \mathbb{R}^d$, definimos

$$s(x) = b + w_1 x_1 + \cdots + w_d x_d, \tag{6.1}$$

em que $w_1, \dots, w_d$ são os **pesos** e b é o **viés** (*bias*) ou intercepto. Cada peso expressa como uma variação no atributo correspondente altera a pontuação, mantendo os outros atributos fixos.

> **i** Peso não implica causalidade
>
> Um peso positivo indica associação positiva dentro do modelo; não prova que o atributo causa o resultado. Dados observacionais podem conter correlações espúrias, variáveis omitidas e vieses de seleção.

6.1 Visão geral dos modelos lineares

6.1.1 Uma estrutura comum para três tarefas

Depois de calcular $s(x)$, aplicamos uma função de saída h_0:

$$h_w(x) = h_0(s(x)) \,.$$

A escolha de h_0 adapta a mesma estrutura linear a diferentes tipos de problema:

Tarefa	Espaço de saída	Transformação	Interpretação
classificação binária	$\{-1, +1\}$	$h_0(s) = \text{sinal}(s)$	escolhe uma de duas classes
regressão linear	$\mathbb{R}$	$h_0(s) = s$	prevê uma quantidade
regressão logística	$(0, 1)$	$h_0(s) = \sigma(s) = 1/(1 + e^{-s})$	estima uma probabilidade

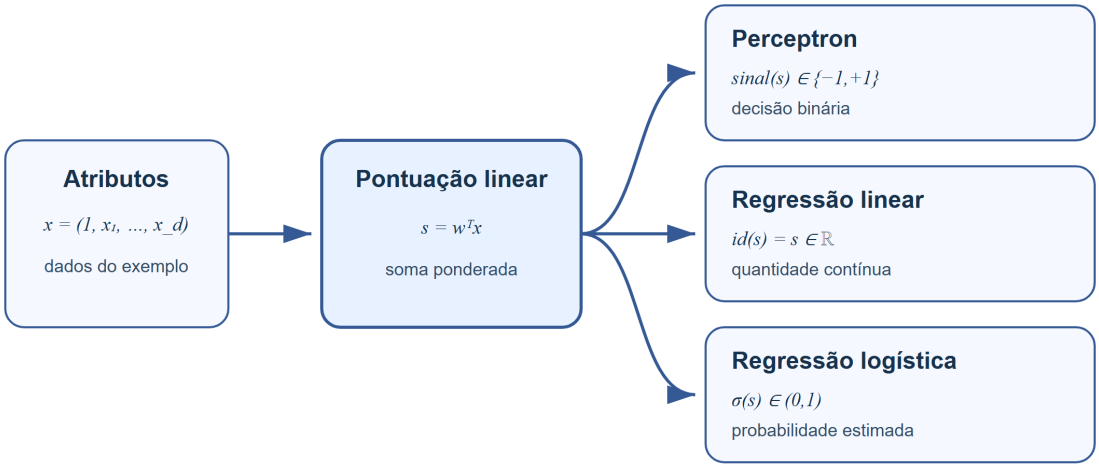


Figura 6.1: A mesma pontuação linear pode alimentar três funções de saída diferentes.

Considere um banco que representa cada cliente por renda, dívida, tempo de emprego e histórico de pagamentos. A mesma entrada pode originar perguntas distintas:

- **classificação:** o pedido deve ser aprovado?
- **regressão:** qual limite de crédito é adequado?
- **probabilidade:** qual é a probabilidade estimada de inadimplência?

O tipo da variável-alvo determina o modelo e a função de erro apropriada. Não faz sentido, por exemplo, interpretar diretamente uma regressão linear como probabilidade, pois sua saída pode ser menor que zero ou maior que um.

6.1.2 Notação matricial e a coordenada de viés

Podemos incorporar b ao vetor de pesos adicionando uma coordenada constante à entrada:

$$\tilde{x} = (1, x_1, \dots, x_d), \qquad \widetilde{w} = (b, w_1, \dots, w_d).$$

Então Equação 6.1 assume a forma compacta

$$s(x) = \widetilde{w}^{\mathsf{T}} \tilde{x}.$$

O sobrescrito T indica transposição. Se os dois vetores forem tratados como colunas, $\widetilde{w}^{\mathsf{T}}$ é uma linha e o produto matricial produz o escalar

$$\widetilde{w}^{\mathsf{T}} \tilde{x} = b + w_1 x_1 + \cdots + w_d x_d.$$

Esse "truque do 1" simplifica fórmulas e implementações: o algoritmo pode atualizar viés e pesos com uma única operação vetorial. É importante, porém, lembrar que um exemplo com d atributos passa a ser representado por $d + 1$ coordenadas.

Para uma amostra com N exemplos, organizamos as entradas nas linhas da matriz de projeto

$$X = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1d} \\ 1 & x_{21} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & \cdots & x_{Nd} \end{bmatrix}.$$

O vetor $X\widetilde{w}$ contém de uma só vez as pontuações dos N exemplos. Essa formulação é central em bibliotecas numéricas, pois substitui laços explícitos por operações matriciais otimizadas.

6.1.3 Aprender significa escolher os pesos

Seja

$$D = \{(x_1, y_1), \dots, (x_N, y_N)\}$$

uma amostra rotulada. Para cada vetor $\widetilde{w}$, o modelo produz predições $h_{\widetilde{w}}(x_n)$. Uma função de perda $\ell(h_{\widetilde{w}}(x_n), y_n)$ mede o custo de cada predição, e o erro empírico é normalmente a média

$$E_{\text{in}}(\widetilde{w}) = \frac{1}{N}\sum_{n=1}^{N} \ell(h_{\widetilde{w}}(x_n), y_n)\,.$$

O algoritmo de aprendizado procura pesos que reduzam esse erro:

$$\widetilde{w}^* \in \arg\min_{\widetilde{w}} E_{\text{in}}(\widetilde{w}).$$

Os três métodos deste capítulo diferem em dois pontos principais: a função de saída e a perda usada na otimização. O Perceptron atualiza os pesos quando encontra um exemplo classificado incorretamente; a regressão linear minimiza o erro quadrático; e a regressão logística minimiza uma perda probabilística.

6.1.4 Geometria da pontuação

O conjunto de pontos com pontuação zero,

$$b + w_1 x_1 + \cdots + w_d x_d = 0,$$

é um **hiperplano**. Em duas dimensões ele é uma reta; em três, um plano. O vetor $w = (w_1, \ldots, w_d)$ é perpendicular a esse hiperplano, enquanto o viés controla seu deslocamento em relação à origem. Na classificação, o sinal da pontuação indica de que lado da fronteira o ponto está. Na regressão, a própria pontuação representa a altura prevista.

Essa interpretação geométrica também revela uma limitação: sem transformar os atributos, uma única fronteira linear não consegue separar padrões como o XOR. Mais adiante, transformações de atributos e redes neurais permitirão construir fronteiras não lineares a partir de componentes lineares.

6.1.5 Escala dos atributos

Os pesos dependem das unidades adotadas. Se renda estiver em centavos e idade em anos, a primeira coordenada pode ter magnitude milhões de vezes maior. Isso pode tornar a otimização instável e dificulta comparar os pesos. Uma prática comum é padronizar cada atributo numérico:

$$x'_j = \frac{x_j - \mu_j}{\sigma_j},$$

em que μ_j e σ_j são calculados **somente no conjunto de treinamento**. Usar estatísticas de

validação ou teste introduziria vazamento de dados.

6.1.6 O que estudaremos

Nas próximas seções, a mesma sequência será repetida para cada modelo:

1. definir a classe de hipóteses;
2. escolher uma função de erro;
3. derivar ou apresentar o algoritmo de otimização;
4. implementar o método computacionalmente;
5. discutir convergência, limitações e uso adequado.

Ao final, será possível reconhecer que Perceptron, regressão linear e regressão logística não são técnicas isoladas. São três instâncias de uma mesma ideia: **combinar atributos linearmente e adaptar a saída ao tipo de pergunta que desejamos responder**.

6.2 4.1 `Perceptron` (ou Discriminador Linear)

O **Perceptron** é um algoritmo para classificação binária. Ele recebe exemplos descritos por vetores numéricos e aprende uma fronteira linear que procura separar duas classes. Apesar de simples, o método introduz ideias que reaparecem em redes neurais: soma ponderada, função de ativação e atualização iterativa dos pesos a partir dos erros.

Considere novamente a análise de crédito. Cada cliente é representado por atributos como renda, dívida e tempo de emprego, e o rótulo histórico informa se houve pagamento $(+1)$ ou inadimplência (-1). O modelo calcula uma pontuação

$$s(x) = w^{\mathsf{T}} x + b$$

e compara essa pontuação com zero:

$$h_{w,b}(x) = \begin{cases} +1, & w^{\mathsf{T}} x + b \geq 0, \\ -1, & w^{\mathsf{T}} x + b < 0. \end{cases} \tag{6.2}$$

Adotar uma regra explícita para o caso de igualdade evita que $\mathrm{sinal}(0)$ fique indefinido na implementação. Outra convenção seria possível, desde que fosse usada de maneira consistente.

6.2.1 Interpretação da pontuação

O valor $s(x)$ contém mais informação do que o rótulo final. Seu sinal determina a classe e seu módulo indica a distância algébrica até o limiar. Cada componente contribui com $w_j x_j$:

- se $w_j > 0$, valores maiores de x_j elevam a pontuação;
- se $w_j < 0$, valores maiores de x_j reduzem a pontuação;
- se $w_j = 0$, o atributo não participa da decisão atual.

Essa leitura exige atenção à escala. Um peso pequeno associado a um atributo medido em milhares pode ter impacto maior do que um peso grande associado a um atributo entre zero e um. Por isso, a contribuição $w_j x_j$ costuma ser mais informativa que o peso isolado.

6.2.2 Interpretação geométrica

A fronteira de decisão contém os pontos de pontuação zero:

$$\{x \in \mathbb{R}^d : w^\mathsf{T} x + b = 0\}.$$

Esse conjunto é um hiperplano perpendicular ao vetor w. Os pontos com pontuação positiva ficam em um semiespaço, e os de pontuação negativa, no outro.

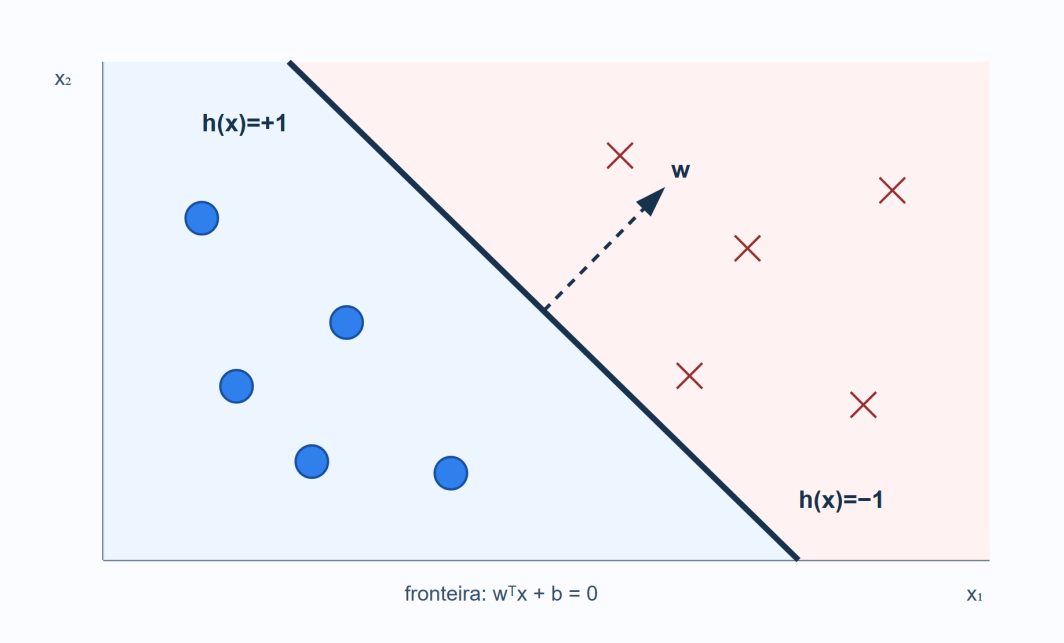


Figura 6.2: A fronteira do Perceptron separa os dois semiespaços; o vetor de pesos é perpendicular a ela.

Em duas dimensões, com $x = (x_1, x_2)$ e $w_2 \neq 0$, podemos escrever a fronteira como uma reta:

$$w_1 x_1 + w_2 x_2 + b = 0 \quad \Longleftrightarrow \quad x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}.$$

Logo, $-w_1/w_2$ é a inclinação e $-b/w_2$ é o intercepto vertical. Alterar a direção de w gira a fronteira; alterar b a desloca sem mudar sua orientação.

6.2.3 O que o algoritmo aprende

Dada uma amostra

$$D = \{(x_1, y_1), \dots, (x_N, y_N)\}, \qquad y_n \in \{-1, +1\},$$

o objetivo, quando os dados são linearmente separáveis, é encontrar w e b tais que

$$y_n(w^{\mathsf{T}} x_n + b) > 0 \qquad \text{para todo } n. \tag{6.3}$$

Por que essa única desigualdade representa as duas classes? Se $y_n = +1$, ela exige pontuação positiva. Se $y_n = -1$, multiplicar por -1 inverte o sinal e exige pontuação negativa. A quantidade

$$y_n(w^{\mathsf{T}} x_n + b)$$

é chamada **margem funcional** do exemplo: é positiva quando a classificação está correta e não positiva quando há erro ou empate.

O Perceptron não calcula a fronteira em uma única fórmula. Ele percorre os exemplos e, sempre que encontra um erro, move os parâmetros na direção que favorece o rótulo correto. As subseções seguintes irão formalizar a classe de hipóteses, o erro e essa regra de atualização.

> ⚠ Hipótese fundamental
>
> O Perceptron clássico converge em um número finito de atualizações somente quando os dados são linearmente separáveis. Se houver ruído, rótulos contraditórios ou uma fronteira essencialmente não linear, ele pode continuar oscilando. A variante *Pocket* tratará esse caso.

6.2.4 Exemplos de aplicação

- **Filtro de spam:** frequências de palavras formam o vetor de entrada, e o sinal da pontuação decide entre spam e mensagem legítima.
- **Controle de qualidade:** medidas de uma peça alimentam uma decisão entre aprovada e defeituosa.
- **Diagnóstico assistido:** atributos clínicos produzem uma triagem binária, que deve ser avaliada com atenção a custos de falsos positivos e falsos negativos.

Nesses exemplos, a decisão automatizada não deve ser confundida com uma verdade absoluta. O modelo reproduz padrões presentes na amostra e pode herdar seus erros e vieses. Em aplicações de alto impacto, métricas por subgrupo, revisão humana e documentação dos dados são partes do sistema, não detalhes opcionais.

6.2.5 Hipóteses

Uma **hipótese** é uma regra de classificação obtida ao fixar os parâmetros do modelo. Para evitar ambiguidade entre os atributos originais e a coordenada constante, usaremos um til na representação homogênea:

$$x = (x_1, \dots, x_d) \quad \longmapsto \quad \tilde{x} = (1, x_1, \dots, x_d),$$

$$(b, w_1, \dots, w_d) \quad \longmapsto \quad \widetilde{w} = (b, w_1, \dots, w_d).$$

O produto escalar entre dois vetores coluna $u, v \in \mathbb{R}^{d+1}$ é

$$u^{\mathsf{T}} v = \sum_{j=0}^{d} u_j v_j.$$

Portanto,

$$\widetilde{w}^{\mathsf{T}} \tilde{x} = b + w_1 x_1 + \cdots + w_d x_d. \tag{6.4}$$

Essa escrita incorpora o viés na mesma operação usada para os demais pesos. Ela também evita um erro comum de orientação: se $\widetilde{w}$ e $\tilde{x}$ são vetores coluna de tamanho $(d+1) \times 1$, então $\widetilde{w}^{\mathsf{T}} \tilde{x}$ tem tamanho 1×1 e representa um escalar. Já $\widetilde{w}\tilde{x}^{\mathsf{T}}$ seria uma matriz $(d+1) \times (d+1)$, não o produto escalar desejado.

Com a convenção de que empates recebem o rótulo $+1$, definimos

$$\text{sinal}_+(z) = \begin{cases} +1, & z \geq 0, \\ -1, & z < 0. \end{cases}$$

A classe de hipóteses do Perceptron em $\mathbb{R}^d$ é, então,

$$\mathcal{H}_{mathrmperc} = \left\{ h_{\widetilde{w}}(x) = \text{sinal}_+(\widetilde{w}^{\mathsf{T}} \tilde{x}) : \widetilde{w} \in \mathbb{R}^{d+1} \right\}. \tag{6.5}$$

Observe a diferença entre três objetos:

- $\mathcal{H}_{\text{perc}}$ é a classe de todas as fronteiras lineares permitidas;
- $h_{\widetilde{w}}$ é uma hipótese particular, determinada por um vetor de parâmetros;
- $g = h_{\widetilde{w}^*}$ é a hipótese final escolhida pelo algoritmo a partir dos dados.

6.2.6 Equivalência de parametrizações

Alguns textos escrevem a pontuação como $w^{\mathsf{T}}x - b$ e chamam b de limiar. Outros escrevem $w^{\mathsf{T}}x + b$ e chamam b de viés. As duas formas são equivalentes: basta trocar b por $-b$. Neste capítulo, adotaremos a segunda convenção para manter a compatibilidade com a notação usual de regressão e redes neurais.

Por exemplo, em duas dimensões,

$$h(x) = \text{sinal}_+(2 - 3x_1 + x_2)$$

possui vetor homogêneo $\widetilde{w} = (2, -3, 1)$. Para $x = (1, 4)$, temos $\tilde{x} = (1, 1, 4)$ e

$$\widetilde{w}^{\mathsf{T}}\tilde{x} = 2 - 3(1) + 1(4) = 3,$$

portanto $h(x) = +1$. Para $x = (2, 1)$, a pontuação é $2 - 3(2) + 1 = -3$ e a previsão é -1.

6.2.7 Invariância por escala positiva

Se $c > 0$, os vetores $\widetilde{w}$ e $c\widetilde{w}$ definem exatamente a mesma classificação, pois

$$\text{sinal}_+((c\widetilde{w})^{\mathsf{T}}\tilde{x}) = \text{sinal}_+(c\,\widetilde{w}^{\mathsf{T}}\tilde{x}).$$

Assim, há infinitos vetores de parâmetros que representam a mesma fronteira. Multiplicar por um número negativo, porém, troca os lados e inverte todos os rótulos. Essa redundância explica por que o tamanho bruto de $\widetilde{w}$ não determina sozinho a fronteira de decisão.

6.2.8 Representação de uma amostra

Se os exemplos homogêneos forem armazenados nas linhas de $\widetilde{X} \in \mathbb{R}^{N\times(d+1)}$, todas as pontuações podem ser calculadas por

$$s = \widetilde{X}\widetilde{w} \in \mathbb{R}^N.$$

A aplicação componente a componente de sinal_+ gera o vetor de predições. Em NumPy, essa ideia corresponde essencialmente a `scores = X_tilde @ w_tilde`, uma operação matricial que também orientará a implementação do algoritmo.

> **Teste de dimensões**
>
> Antes de implementar uma fórmula, anote as dimensões: $\widetilde{X}$ é $N\times(d+1)$ e $\widetilde{w}$ é $(d+1)\times 1$; logo, o produto é $N\times 1$, uma pontuação por exemplo.

6.2.9 Erro

Para avaliar uma hipótese de classificação, a medida mais direta é contar quantas previsões discordam dos rótulos. A **perda zero–um** de um exemplo é

$$\ell_{\text{0-1}}(h(x_n), y_n) = \mathbb{1}[h(x_n) \neq y_n]\,,$$

em que $\mathbb{1}[Q]$ vale 1 quando a proposição Q é verdadeira e 0 quando é falsa. O número total de erros na amostra é

$$N_{\text{erro}}(h) = \sum_{n=1}^{N} \mathbb{1}[h(x_n) \neq y_n]\,,$$

e a taxa de erro dentro da amostra é

$$E_{\text{in}}(h) = \frac{1}{N} N_{\text{erro}}(h). \tag{6.6}$$

A contagem e a taxa contêm a mesma informação quando N é fixo. A taxa é geralmente preferível porque permanece entre zero e um e permite comparar amostras de tamanhos diferentes.

6.2.10 Erro escrito com rótulos $\{-1, +1\}$

Como $h(x_n)$ e y_n pertencem a $\{-1, +1\}$, sua diferença é zero em um acerto e ± 2 em um erro. Logo,

$$\mathbb{1}[h(x_n) \neq y_n] = \frac{(h(x_n) - y_n)^2}{4}.$$

O fator $1/4$ é indispensável: sem ele, cada erro seria contado como quatro. Portanto, uma forma equivalente de Equação 6.6 é

$$E_{\text{in}}(h) = \frac{1}{4N} \sum_{n=1}^{N} (h(x_n) - y_n)^2 .$$

Essa equivalência depende especificamente da codificação $\{-1, +1\}$. Com rótulos $\{0, 1\}$, o quadrado da diferença já vale um nos erros.

6.2.11 Margem e condição de erro

Para o Perceptron, é mais conveniente expressar o erro antes de aplicar a função sinal. Defina a margem funcional do exemplo n por

$$m_n = y_n \left(w^{\mathsf{T}} x_n + b \right).$$

Com a convenção adotada para empates, um exemplo é certamente classificado de modo correto quando $m_n > 0$. Quando $m_n < 0$, ele está do lado errado da fronteira. Em $m_n = 0$, a perda zero–um depende do rótulo e da regra de desempate. O algoritmo clássico adota um critério mais conservador e solicita uma atualização para qualquer margem não positiva:

$$\ell_{\text{perc}}(m_n) = \mathbb{1}[m_n \leq 0]. \tag{6.7}$$

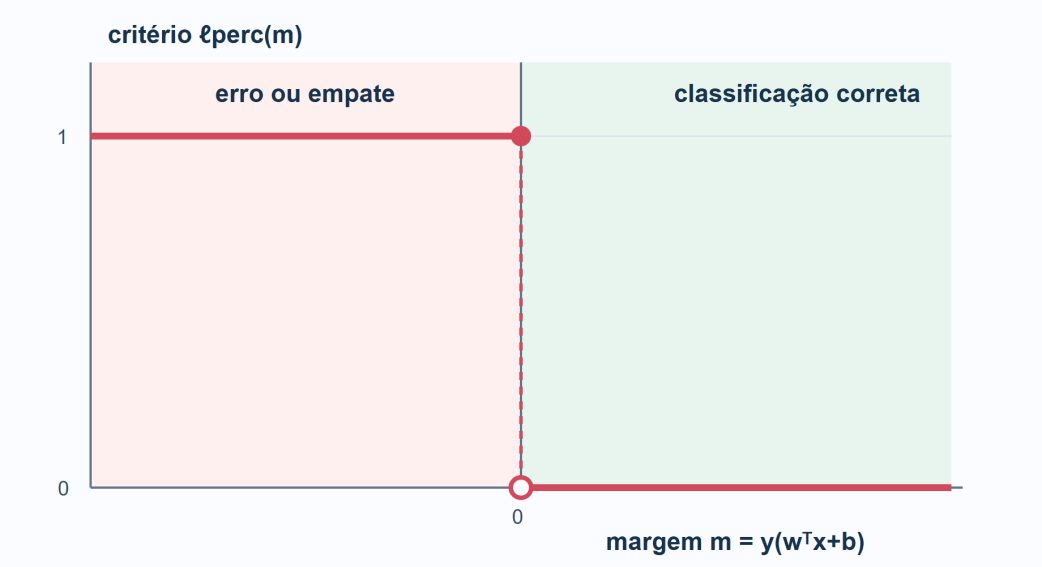


Figura 6.3: O critério conservador de atualização muda abruptamente quando a margem cruza zero.

O módulo de m_n não altera esse critério: uma classificação correta por margem $0{,}001$ e outra por margem 100 recebem perda zero. Esse comportamento é adequado para medir acurácia, mas fornece pouca informação para otimização.

6.2.12 Por que não minimizar diretamente a perda zero–um?

A função em degrau de Equação 6.7 é descontínua em zero e constante em quase todos os demais pontos. Seu gradiente é zero onde existe e não está definido no salto. Consequentemente, métodos de gradiente não recebem uma direção útil para mover os pesos.

O Perceptron contorna esse problema com uma regra operacional simples: seleciona um exemplo com $m_n \leq 0$ e ajusta os pesos para aumentar sua margem. Essa regra será derivada na próxima subseção. Outros modelos substituem a perda zero–um por funções contínuas, como a perda logística ou a *hinge loss*.

6.2.13 Separabilidade e valor mínimo

Se existe (w, b) para o qual $m_n > 0$ para todo n, a amostra é **linearmente separável** e o mínimo da taxa de erro é zero. Nesse caso, o teorema de convergência do Perceptron garante que o algoritmo encontra uma solução em número finito de atualizações.

Se os dados não são separáveis, então nenhuma hipótese da classe alcança erro zero. O Perceptron clássico pode visitar diferentes vetores de peso sem estabilizar. Isso não significa que todos

sejam igualmente bons: alguns cometem menos erros que outros. A variante *Pocket* guardará no "bolso" o melhor vetor observado durante a execução.

> **!** Erro de treinamento não é erro de generalização
>
> Mesmo quando $E_{\text{in}}(h) = 0$, ainda é necessário avaliar a hipótese em dados não usados no ajuste. Uma fronteira pode separar a amostra por acaso, especialmente quando há poucos exemplos ou muitos atributos.

6.2.14 Acurácia e custos assimétricos

A acurácia é $1 - E_{\text{in}}(h)$, mas nem sempre resume o objetivo da aplicação. Em diagnóstico médico, deixar de detectar uma doença pode ser mais grave do que gerar um alarme falso; em filtros de spam, bloquear uma mensagem legítima pode ter custo elevado. Nesses casos, além da taxa de erro, devem ser examinadas a matriz de confusão, precisão, revocação e uma função de custo coerente com o problema.

6.2.14.1 Verificação rápida

Considere rótulos verdadeiros $(+1, +1, -1, -1)$ e previsões $(+1, -1, -1, +1)$. Há dois erros em quatro exemplos, portanto $N_{\text{erro}} = 2$, $E_{\text{in}} = 0{,}5$ e a acurácia é $0{,}5$. Pela fórmula quadrática, a soma dos quadrados é $0 + 4 + 0 + 4 = 8$; dividindo por $4N = 16$, obtemos novamente $0{,}5$.

6.2.15 Algoritmo

O algoritmo do Perceptron constrói os parâmetros por correções locais. Começamos com um vetor, frequentemente $\widetilde{w}^{(0)} = 0$. Na iteração t, escolhemos um exemplo $(\tilde{x}_n, y_n)$ cuja margem não é positiva:

$$y_n \left(\widetilde{w}^{(t)}\right)^{\mathsf{T}} \tilde{x}_n \leq 0. \tag{6.8}$$

Em seguida, atualizamos

$$\widetilde{w}^{(t+1)} = \widetilde{w}^{(t)} + \eta y_n \tilde{x}_n, \tag{6.9}$$

em que $\eta > 0$ é a **taxa de aprendizado**. No Perceptron clássico, $\eta = 1$ é suficiente; deixar o parâmetro explícito ajuda a interpretar o tamanho do passo e aproxima a notação da usada em outros métodos.

6.2.16 Por que a atualização aponta para o lado correto?

Se $y_n = +1$, somamos $\eta\tilde{x}_n$ aos pesos, aumentando a pontuação desse exemplo. Se $y_n = -1$, subtraímos $\eta\tilde{x}_n$, reduzindo sua pontuação. Em ambos os casos, a nova margem é

$$\begin{aligned} m_n^{(t+1)} &= y_n \left(\widetilde{w}^{(t+1)}\right)^{\mathsf{T}} \tilde{x}_n \\ &= y_n \left(\widetilde{w}^{(t)} + \eta y_n \tilde{x}_n\right)^{\mathsf{T}} \tilde{x}_n \\ &= m_n^{(t)} + \eta y_n^2 \tilde{x}_n^{\mathsf{T}} \tilde{x}_n \\ &= m_n^{(t)} + \eta \|\tilde{x}_n\|_2^2. \end{aligned} \tag{6.10}$$

Usamos $y_n^2 = 1$. Como $\eta > 0$ e $\|\tilde{x}_n\|_2^2 > 0$, a margem do exemplo selecionado sempre aumenta.

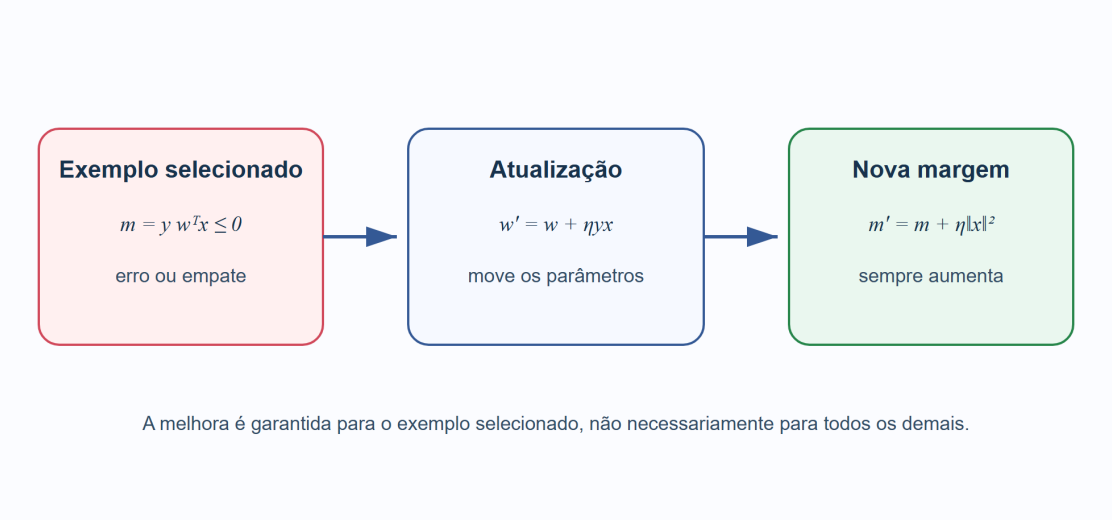


Figura 6.4: Uma atualização aumenta a margem do exemplo que provocou a correção.

Essa garantia é **local**: a atualização pode reduzir a margem de outros exemplos e até fazer uma previsão antes correta tornar-se incorreta. O teorema de convergência não afirma que o número de erros diminui a cada passo; afirma que, sob separabilidade, esse processo de correções não pode continuar para sempre.

6.2.17 Atualizando viés e pesos separadamente

Como $\tilde{x}_n = (1, x_n)$ e $\widetilde{w} = (b, w)$, Equação 6.9 equivale a

$$\begin{aligned} b^{(t+1)} &= b^{(t)} + \eta y_n, \\ w^{(t+1)} &= w^{(t)} + \eta y_n x_n. \end{aligned}$$

Essa forma é útil quando a implementação armazena o viés separado. Se os atributos forem transformados ou normalizados, a coordenada constante não deve ser normalizada junto com eles.

6.2.18 Pseudocódigo

```
entrada: amostra D, taxa   > 0, limite de épocas
inicialize w e b

para cada época:
    número_de_atualizações ← 0
    para cada (x_n, y_n) em D:
        margem ← y_n (w  x_n + b)
        se margem   0:
            w ← w +   y_n x_n
            b ← b +   y_n
            número_de_atualizações ← número_de_atualizações + 1
    se número_de_atualizações = 0:
        encerre: todos os exemplos têm margem positiva

devolva w e b
```

Uma **época** é uma passagem completa pela amostra. O contador de atualizações fornece um critério de parada mais confiável que verificar se o vetor mudou "pouco": no Perceptron, ausência de atualizações durante uma época significa que todos os exemplos foram classificados corretamente segundo o critério adotado.

6.2.19 Ordem dos exemplos

A solução encontrada não é única e pode depender da ordem em que os exemplos são apresentados. Em implementações práticas, é comum embaralhar a amostra a cada época. Isso evita padrões artificiais na ordem dos dados e pode acelerar a convergência, embora não altere a hipótese matemática de separabilidade.

Há duas variantes de seleção frequentes:

- **on-line ou estocástica:** atualiza imediatamente ao encontrar cada erro;
- **por erro escolhido:** localiza algum exemplo incorreto, atualiza e reinicia a busca.

Ambas seguem a mesma regra fundamental. A primeira costuma ser mais natural para dados

recebidos em fluxo.

6.2.20 Taxa de aprendizado e inicialização

Para dados separáveis e taxa positiva constante, mudar η ou a escala da inicialização frequentemente altera o representante numérico, mas não a essência da busca por uma fronteira. Ainda assim, valores extremos podem causar problemas de ponto flutuante. Inicializar em zero é válido para um único Perceptron, diferentemente de redes neurais com várias unidades simétricas.

> ⚠ Sempre imponha um limite de execução
>
> Se a amostra não for separável, o critério "pare quando não houver erros" pode nunca ser satisfeito. Uma implementação robusta recebe um número máximo de épocas ou atualizações e registra a melhor solução observada.

6.2.21 Convergência não é generalização

Quando o algoritmo encerra com zero erros, sabemos apenas que encontrou uma fronteira consistente com a amostra. O desempenho em exemplos novos depende da representatividade dos dados e da complexidade da classe. A convergência é uma propriedade do procedimento de otimização; a generalização é uma propriedade estatística avaliada fora do conjunto de treinamento.

6.2.22 Código

Uma implementação deve tornar explícitas decisões que o pseudocódigo matemático costuma omitir: formato dos dados, critério para empates, limite de épocas, embaralhamento e informações devolvidas ao usuário. O código abaixo usa NumPy e armazena o viés separadamente dos pesos.

```
from dataclasses import dataclass

import numpy as np


@dataclass
class ResultadoPerceptron:
    pesos: np.ndarray
    vies: float
    epocas: int
    atualizacoes: int
```

```
    convergiu: bool
    erros_por_epoca: list[int]


def treinar_perceptron(
    X,
    y,
    *,
    taxa=1.0,
    max_epocas=1_000,
    embaralhar=True,
    semente=0,
):
    """Treina um classificador Perceptron binário.

    X deve ter forma (n_exemplos, n_atributos) e y deve conter
    somente os rótulos -1 e +1.
    """
    X = np.asarray(X, dtype=float)
    y = np.asarray(y, dtype=int)

    if X.ndim != 2:
        raise ValueError("X deve ser uma matriz bidimensional")
    if y.ndim != 1 or len(y) != len(X):
        raise ValueError("y deve ter um rótulo para cada linha de X")
    if not np.all(np.isin(y, (-1, 1))):
        raise ValueError("os rótulos devem pertencer a {-1, +1}")
    if taxa <= 0 or max_epocas <= 0:
        raise ValueError("taxa e max_epocas devem ser positivos")

    rng = np.random.default_rng(semente)
    pesos = np.zeros(X.shape[1], dtype=float)
    vies = 0.0
    atualizacoes = 0
    erros_por_epoca = []

    for epoca in range(1, max_epocas + 1):
```

```
        indices = np.arange(len(X))
        if embaralhar:
            rng.shuffle(indices)

        erros = 0
        for i in indices:
            margem = y[i] * (X[i] @ pesos + vies)
            if margem <= 0:
                pesos += taxa * y[i] * X[i]
                vies += taxa * y[i]
                erros += 1
                atualizacoes += 1

        erros_por_epoca.append(erros)
        if erros == 0:
            return ResultadoPerceptron(
                pesos, vies, epoca, atualizacoes, True,
                erros_por_epoca
            )

    return ResultadoPerceptron(
        pesos, vies, max_epocas, atualizacoes, False,
        erros_por_epoca
    )
```

O uso de argumentos nomeados após `*` reduz enganos como trocar a taxa de aprendizado pelo número de épocas. A semente torna o embaralhamento reproduzível, algo importante em experimentos e depuração.

6.2.23 Predição

Treinamento e predição devem ser funções separadas. Isso permite aplicar os mesmos parâmetros a conjuntos de treino, validação e teste.

```
def prever_perceptron(X, pesos, vies):
    X = np.asarray(X, dtype=float)
    pontuacoes = X @ pesos + vies
    # Mesma convenção do capítulo: empate recebe +1.
```

```
    return np.where(pontuacoes >= 0, 1, -1)


def taxa_de_erro(y_real, y_predito):
    y_real = np.asarray(y_real)
    y_predito = np.asarray(y_predito)
    return np.mean(y_real != y_predito)
```

6.2.24 Exemplo reproduzível

```
X = np.array([
    [-2.0, 4.0],
    [ 4.0, 1.0],
    [ 1.0, 6.0],
    [ 2.0, 4.0],
    [ 6.0, 2.0],
])
y = np.array([-1, -1, 1, 1, 1])

resultado = treinar_perceptron(
    X,
    y,
    taxa=1.0,
    max_epocas=1_000,
    embaralhar=True,
    semente=42,
)

predicoes = prever_perceptron(
    X, resultado.pesos, resultado.vies
)

print("pesos:", resultado.pesos)
print("viés:", resultado.vies)
print("convergiu:", resultado.convergiu)
print("erro de treino:", taxa_de_erro(y, predicoes))
```

Se `resultado.convergiu` for falso, não se deve interpretar o último vetor como necessariamente o melhor. Ele é apenas o estado no instante em que o limite foi atingido. A seção sobre o algoritmo *Pocket* mostrará como guardar os parâmetros de menor erro.

6.2.25 Visualização em duas dimensões

Quando há exatamente dois atributos e $w_2 \neq 0$, a fronteira $w_1 x_1 + w_2 x_2 + b = 0$ pode ser desenhada como $x_2 = -(w_1 x_1 + b)/w_2$:

```
import matplotlib.pyplot as plt


def desenhar_fronteira_2d(X, y, pesos, vies):
    fig, ax = plt.subplots()

    positivos = y == 1
    ax.scatter(
        X[positivos, 0], X[positivos, 1],
        marker="o", label="+1"
    )
    ax.scatter(
        X[~positivos, 0], X[~positivos, 1],
        marker="x", label="-1"
    )

    limite_x = np.array(ax.get_xlim())
    if not np.isclose(pesos[1], 0.0):
        limite_y = -(pesos[0] * limite_x + vies) / pesos[1]
        ax.plot(limite_x, limite_y, label="fronteira")
    else:
        ax.axvline(-vies / pesos[0], label="fronteira")

    ax.set_xlabel("atributo 1")
    ax.set_ylabel("atributo 2")
    ax.legend()
    return fig, ax
```

```
desenhar_fronteira_2d(
    X, y, resultado.pesos, resultado.vies
)
plt.show()
```

O caso `pesos[1] == 0` merece tratamento próprio, pois a fronteira é vertical e a fórmula de inclinação dividiria por zero.

> **Evite vazamento no pré-processamento**
>
> Se os atributos forem padronizados, ajuste médias e desvios apenas nos dados de treinamento. Aplique depois a mesma transformação aos dados de validação, teste e produção.

6.2.25.1 Questões para experimentar

1. Execute o treinamento com sementes diferentes. Os pesos finais são iguais? As previsões na amostra são iguais?
2. Remova `max_epocas` de uma implementação ingênua e insira dois exemplos idênticos com rótulos opostos. O que acontece?
3. Compare o histórico `erros_por_epoca` com a taxa de erro calculada depois de cada época. Por que o número de atualizações durante uma época não precisa ser igual ao erro dos parâmetros ao final dela?

6.2.26 Exemplo em dois passos

Vamos acompanhar um caso pequeno o bastante para verificar cada conta. Considere os pontos bidimensionais

$$p = (0, 2), \qquad y_p = +1,$$

$$q = (1, 1), \qquad y_q = -1.$$

Na representação homogênea,

$$\tilde{p} = (1, 0, 2), \qquad \tilde{q} = (1, 1, 1).$$

Escolha como estado inicial $\widetilde{w}^{(0)} = (0, 0, 2)$. A primeira coordenada é o viés; as duas restantes são os pesos dos atributos. A fronteira inicial satisfaz

$$0 + 0x_1 + 2x_2 = 0,$$

isto é, $x_2 = 0$.

6.2.26.1 Passo 1: localizar o erro

As pontuações iniciais são

$$\left(\widetilde{w}^{(0)}\right)^{\mathsf{T}} \tilde{p} = 0 + 0(0) + 2(2) = 4,$$

$$\left(\widetilde{w}^{(0)}\right)^{\mathsf{T}} \tilde{q} = 0 + 0(1) + 2(1) = 2.$$

O ponto p é positivo e recebeu pontuação positiva, portanto está correto. O ponto q é negativo, mas também recebeu pontuação positiva. Sua margem confirma o erro:

$$m_q^{(0)} = y_q \left(\widetilde{w}^{(0)}\right)^{\mathsf{T}} \tilde{q} = (-1)(2) = -2 < 0.$$

6.2.26.2 Passo 2: atualizar e verificar

Com taxa $\eta = 1$, usamos q para atualizar:

$$\begin{aligned} \widetilde{w}^{(1)} &= \widetilde{w}^{(0)} + y_q \tilde{q} \\ &= (0, 0, 2) - (1, 1, 1) \\ &= (-1, -1, 1). \end{aligned}$$

A nova fronteira é

$$-1 - x_1 + x_2 = 0 \quad \Longleftrightarrow \quad x_2 = x_1 + 1.$$

Agora, as pontuações são

$$\left(\widetilde{w}^{(1)}\right)^{\mathsf{T}} \tilde{p} = -1 - 1(0) + 1(2) = 1,$$

$$\left(\widetilde{w}^{(1)}\right)^{\mathsf{T}} \tilde{q} = -1 - 1(1) + 1(1) = -1.$$

Logo, p recebe $+1$ e q recebe -1. Uma única atualização foi suficiente para separar a amostra.

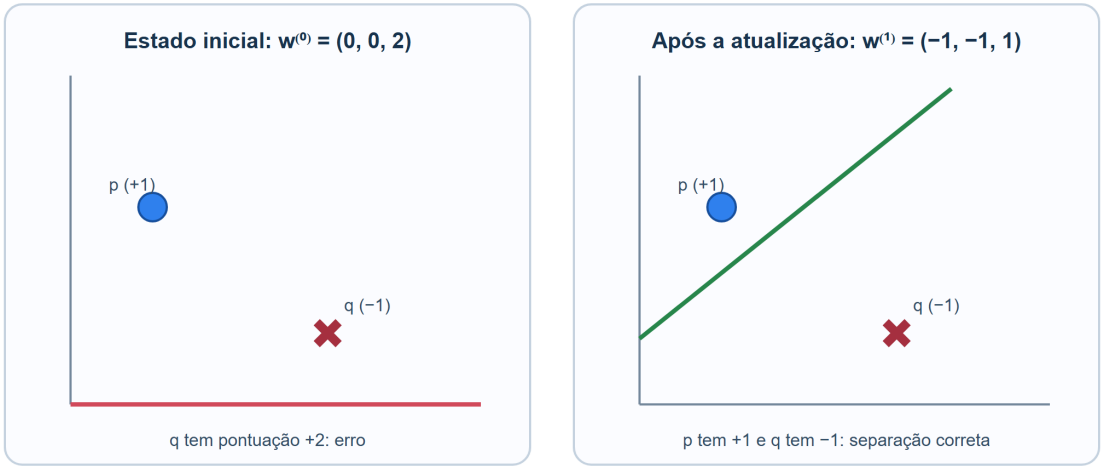


Figura 6.5: A correção causada por q transforma a fronteira horizontal em uma reta que separa os dois pontos.

Podemos conferir diretamente a identidade da melhora de margem. Como $\|\tilde{q}\|^2 = 1^2+1^2+1^2 = 3$,

$$m_q^{(1)} = m_q^{(0)} + \|\tilde{q}\|^2 = -2 + 3 = 1,$$

que coincide com $y_q(\widetilde{w}^{(1)})^\top \tilde{q} = (-1)(-1) = 1$.

> **i** Por que o título diz dois passos?
>
> Os dois passos são diagnosticar o exemplo incorreto e aplicar a correção. Há apenas **uma atualização de pesos**. Distinguir iteração, atualização e época evita confusão ao analisar o tempo de execução.

6.2.26.3 Exercício de extensão

Refaça as contas começando com $\widetilde{w}^{(0)} = (0, 0, 0)$ e percorrendo primeiro p e depois q. Registre, após cada atualização, as pontuações, as margens e a equação da fronteira.

6.2.27 Tempo de Execução

Há dois custos diferentes a considerar: o custo de **uma operação** e o número de operações até a convergência.

Para um exemplo com d atributos, calcular $w^\top x + b$ custa $O(d)$. Uma passagem por N exemplos custa $O(Nd)$; portanto, executar K épocas tem custo $O(KNd)$ e requer $O(d)$ de memória para os parâmetros, além da memória usada para armazenar ou transmitir os dados.

Esse cálculo não informa quanto vale K. Para dados separáveis, o **teorema de convergência do Perceptron** fornece uma cota para o número de atualizações, baseada na geometria da amostra.

6.2.28 Raio e margem

Usaremos os vetores homogêneos $\tilde{x}_n$ para incluir o viés. Suponha que exista um vetor unitário u, com $\|u\|_2 = 1$, que separe a amostra com margem $\gamma > 0$:

$$y_n u^\top \tilde{x}_n \geq \gamma \qquad \text{para todo } n. \tag{6.11}$$

Defina também o raio

$$R = \max_{1 \leq n \leq N} \|\tilde{x}_n\|_2. \tag{6.12}$$

Como u é unitário, $|u^\top \tilde{x}|$ é a distância assinada do ponto ao hiperplano $u^\top \tilde{x} = 0$. A margem γ é a **menor** distância assinada, depois de corrigir o lado com y_n; é o ponto mais próximo que limita a folga da separação.

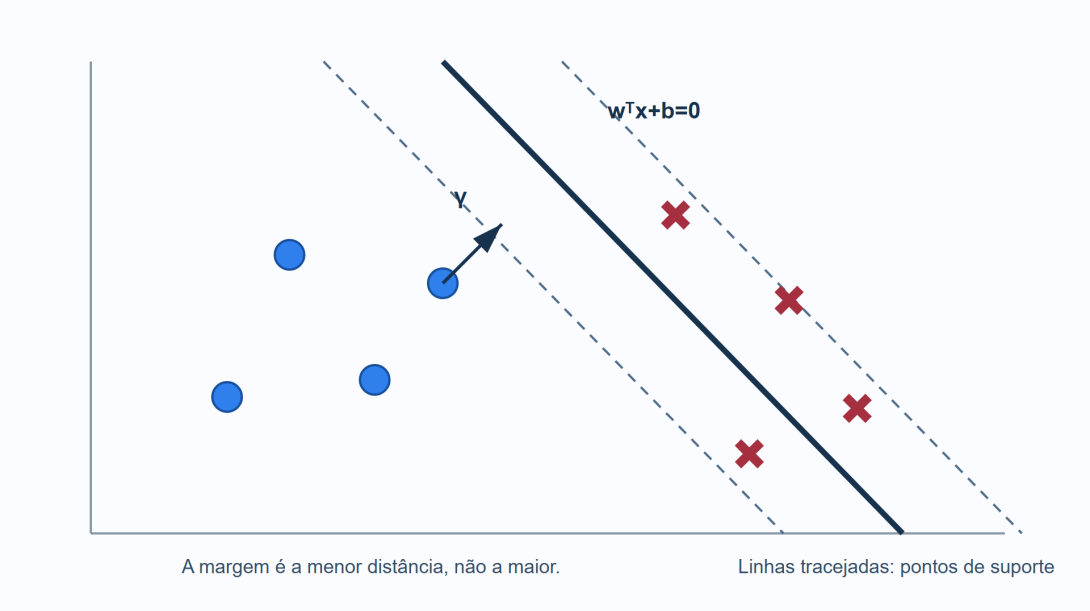


Figura 6.6: A margem é determinada pelos exemplos mais próximos da fronteira.

Para uma fronteira na forma não normalizada $w^\top x + b = 0$, a distância geométrica de um ponto

x é

$$\operatorname{dist}(x, H) = \frac{|w^{\mathsf{T}} x + b|}{\|w\|_2}.$$

O denominador é $\|w\|_2$, e não $\|w\|_2^2$. Essa fórmula decorre da projeção sobre a direção normal $w/\|w\|_2$. Recordando o produto escalar,

$$v^{\mathsf{T}} z = \|v\|_2 \, \|z\|_2 \cos\theta,$$

em que θ é o ângulo entre os vetores.

6.2.29 Teorema de convergência

Teorema de convergência do Perceptron

Considere $\widetilde{w}^{(0)} = 0$, taxa $\eta = 1$ e uma amostra que satisfaz Equação 6.11 e Equação 6.12. O Perceptron comete no máximo

$$T \leq \left(\frac{R}{\gamma}\right)^2$$

atualizações antes de encontrar uma separação correta.

A cota depende da razão R/γ. Uma margem grande facilita o problema; exemplos com normas muito grandes aumentam a cota. Isso também ajuda a entender por que escalonar atributos pode melhorar o comportamento numérico.

6.2.30 Ideia da demonstração

Depois de T atualizações, cada uma provocada por $(\tilde{x}_t, y_t)$, temos

$$\widetilde{w}^{(T)} = \sum_{t=1}^{T} y_t \tilde{x}_t.$$

Projetando o vetor acumulado sobre a direção da solução u,

$$u^{\mathsf{T}} \widetilde{w}^{(T)} = \sum_{t=1}^{T} y_t u^{\mathsf{T}} \tilde{x}_t \geq T\gamma. \tag{6.13}$$

Por Cauchy–Schwarz e $\|u\| = 1$,

$$u^{\mathsf{T}} \widetilde{w}^{(T)} \leq \|\widetilde{w}^{(T)}\|_2.$$

Agora limitamos o crescimento da norma. Como uma atualização ocorre quando $y_t(\widetilde{w}^{(t)})^{\mathsf{T}} \tilde{x}_t \leq 0$,

$$\begin{aligned} \|\widetilde{w}^{(t+1)}\|_2^2 &= \|\widetilde{w}^{(t)} + y_t \tilde{x}_t\|_2^2 \\ &= \|\widetilde{w}^{(t)}\|_2^2 + 2y_t(\widetilde{w}^{(t)})^{\mathsf{T}} \tilde{x}_t + \|\tilde{x}_t\|_2^2 \\ &\leq \|\widetilde{w}^{(t)}\|_2^2 + R^2. \end{aligned}$$

Somando as T atualizações,

$$\|\widetilde{w}^{(T)}\|_2^2 \leq TR^2, \qquad \|\widetilde{w}^{(T)}\|_2 \leq \sqrt{T}R. \tag{6.14}$$

Combinando Equação 6.13 e Equação 6.14,

$$T\gamma \leq \sqrt{T}R \quad \Longrightarrow \quad T \leq \frac{R^2}{\gamma^2}.$$

O argumento mostra um contraste: o alinhamento com a solução cresce linearmente em T, enquanto a norma cresce no máximo como $\sqrt{T}$. Se as atualizações continuassem além da cota, as duas desigualdades seriam incompatíveis.

6.2.31 O que a cota não diz

- Ela limita **atualizações**, não necessariamente épocas. Uma época pode conter nenhuma, uma ou várias atualizações.
- É uma cota de pior caso; a execução real pode terminar muito antes.
- Ela exige separabilidade com margem positiva. Com dados não separáveis, não há garantia de término.
- Ela não afirma que o Perceptron encontra a fronteira de margem máxima. Métodos como máquinas de vetores de suporte perseguem esse objetivo de forma explícita.
- Ela não é uma garantia de generalização; apenas prova convergência no conjunto de treinamento.

6.2.32 Relação com o custo total

No modo on-line, cada exemplo custa $O(d)$ e o teorema limita a quantidade de correções a $(R/\gamma)^2$. Porém, encontrar uma época sem erros ainda requer verificar os exemplos. Para uma implementação por épocas, a medida mais transparente continua sendo

$$O(KNd),$$

registrando separadamente K e o número total de atualizações. Essa distinção evita interpretar a cota teórica como se fosse diretamente o tempo de relógio do programa.

6.2.32.1 Exercícios

1. O que acontece com a cota se todos os vetores de entrada forem multiplicados por uma constante positiva c? Considere também como a margem se transforma.
2. Mostre em qual passo da demonstração a separabilidade é usada.
3. Uma execução fez 80 atualizações em 12 épocas. Quais desses números podem ser comparados diretamente com $(R/\gamma)^2$?

6.2.33 Animação

Uma animação ajuda a desfazer uma interpretação enganosa: o Perceptron não move a fronteira continuamente em direção a uma reta ideal. A cada erro, ele salta para um novo vetor de parâmetros. A fronteira pode girar, transladar e até voltar a classificar incorretamente um ponto que estava correto.

Para acompanhar o processo, cada quadro deve mostrar pelo menos:

- os pontos e seus rótulos verdadeiros;
- a fronteira atual $w^{\mathsf{T}}x + b = 0$;
- o exemplo que provocou a atualização;
- o número da época e da atualização;
- a taxa de erro ou o número de erros dos parâmetros atuais.

6.2.34 Registrando os quadros

O gerador abaixo é uma versão compacta do treinamento. Ele produz um estado depois de cada correção, sem armazenar todos os quadros simultaneamente.

```
import numpy as np


def quadros_perceptron(X, y, max_epocas=100):
    X = np.asarray(X, dtype=float)
    y = np.asarray(y, dtype=int)
    w = np.zeros(X.shape[1])
    b = 0.0
    atualizacao = 0

    # Quadro inicial.
    yield {
        "epoca": 0,
        "atualizacao": 0,
        "indice": None,
        "w": w.copy(),
        "b": b,
    }

    for epoca in range(1, max_epocas + 1):
        houve_atualizacao = False

        for i, (x_i, y_i) in enumerate(zip(X, y)):
            if y_i * (w @ x_i + b) <= 0:
                w += y_i * x_i
                b += y_i
                atualizacao += 1
                houve_atualizacao = True

                yield {
                    "epoca": epoca,
                    "atualizacao": atualizacao,
                    "indice": i,
                    "w": w.copy(),
                    "b": b,
                }
```

```
        if not houve_atualizacao:
            return
```

O método `copy()` é importante. Sem ele, todos os dicionários poderiam referenciar o mesmo vetor mutável e parecer conter apenas o estado final. Esse é um erro frequente ao registrar históricos numéricos em Python.

6.2.35 Construindo a animação

Para dados bidimensionais, podemos transformar os estados em uma animação com `matplotlib.animation.FuncAnimation`:

```
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation


def animar_perceptron(X, y, max_epocas=100, intervalo=700):
    estados = list(quadros_perceptron(X, y, max_epocas))
    fig, ax = plt.subplots()

    positivos = y == 1
    ax.scatter(X[positivos, 0], X[positivos, 1], label="+1")
    ax.scatter(
        X[~positivos, 0], X[~positivos, 1],
        marker="x", label="-1"
    )
    destaque = ax.scatter([], [], s=180, facecolors="none",
                          edgecolors="black")
    fronteira, = ax.plot([], [], color="black")
    titulo = ax.set_title("")
    ax.legend()

    xmin, xmax = ax.get_xlim()
    ymin, ymax = ax.get_ylim()
    eixo_x = np.array([xmin, xmax])

    def atualizar(estado):
        w, b = estado["w"], estado["b"]
```

```
        if not np.isclose(w[1], 0.0):
            eixo_y = -(w[0] * eixo_x + b) / w[1]
            fronteira.set_data(eixo_x, eixo_y)
        elif not np.isclose(w[0], 0.0):
            x_vertical = -b / w[0]
            fronteira.set_data([x_vertical, x_vertical], [ymin, ymax])
        else:
            fronteira.set_data([], [])

        i = estado["indice"]
        destaque.set_offsets(
            np.empty((0, 2)) if i is None else X[[i]]
        )
        titulo.set_text(
            f"época {estado['epoca']} · "
            f"atualização {estado['atualizacao']}"
        )
        return fronteira, destaque, titulo

    animacao = FuncAnimation(
        fig,
        atualizar,
        frames=estados,
        interval=intervalo,
        repeat=False,
    )
    return fig, animacao
```

Em um notebook, o resultado pode ser exibido como HTML:

```
from IPython.display import HTML

fig, animacao = animar_perceptron(X, y)
plt.close(fig)
HTML(animacao.to_jshtml())
```

Para salvar em GIF ou vídeo, o Matplotlib precisa de um escritor compatível instalado no ambiente. A exibição em `to_jshtml()` evita essa dependência e costuma ser suficiente para

experimentos didáticos.

6.2.36 O que observar

Ao executar a animação, responda:

1. Toda atualização reduz o número total de erros?
2. O exemplo destacado fica correto imediatamente após sua atualização?
3. Quantas atualizações ocorrem em cada época?
4. A fronteira final muda quando a ordem dos exemplos muda?
5. O comportamento se torna mais estável após padronizar os atributos?

> **Animação não é prova**
>
> A visualização produz intuição e ajuda a depurar uma implementação, mas mostra apenas uma amostra e uma ordem de apresentação. A garantia de convergência vem do argumento matemático da seção anterior.

6.2.37 O Algoritmo `Pocket` (= bolso)

O Perceptron clássico foi projetado para encontrar uma separação perfeita. Quando a amostra não é linearmente separável, suas atualizações continuam, e o último vetor visitado pode ser pior que vetores encontrados anteriormente. O algoritmo **Pocket** acrescenta memória ao processo: ele mantém no "bolso" os parâmetros com menor erro observados até então.

Essa modificação é útil quando há ruído de medição, rótulos incorretos, exemplos contraditórios ou uma fronteira verdadeira não linear. Nesses casos, exigir $E_{\text{in}} = 0$ é impossível ou indesejável.

6.2.37.1 Estado corrente e estado guardado

O Pocket mantém dois vetores distintos:

- $\widetilde{w}_{\text{atual}}$, que continua recebendo as atualizações do Perceptron;
- $\widetilde{w}_{\text{melhor}}$, que representa o menor erro de treinamento encontrado até aquele instante.

Depois de cada atualização do estado corrente, calculamos

$$E_{\text{in}}(\widetilde{w}) = \frac{1}{N} \sum_{n=1}^{N} \mathbb{1}[h_{\widetilde{w}}(x_n) \neq y_n] \, .$$

Se o novo erro for menor que o erro guardado, copiamos os parâmetros para o bolso. Como o melhor estado só é substituído por outro melhor, sua sequência de erros é monotonicamente não

crescente, mesmo que o estado corrente oscile.

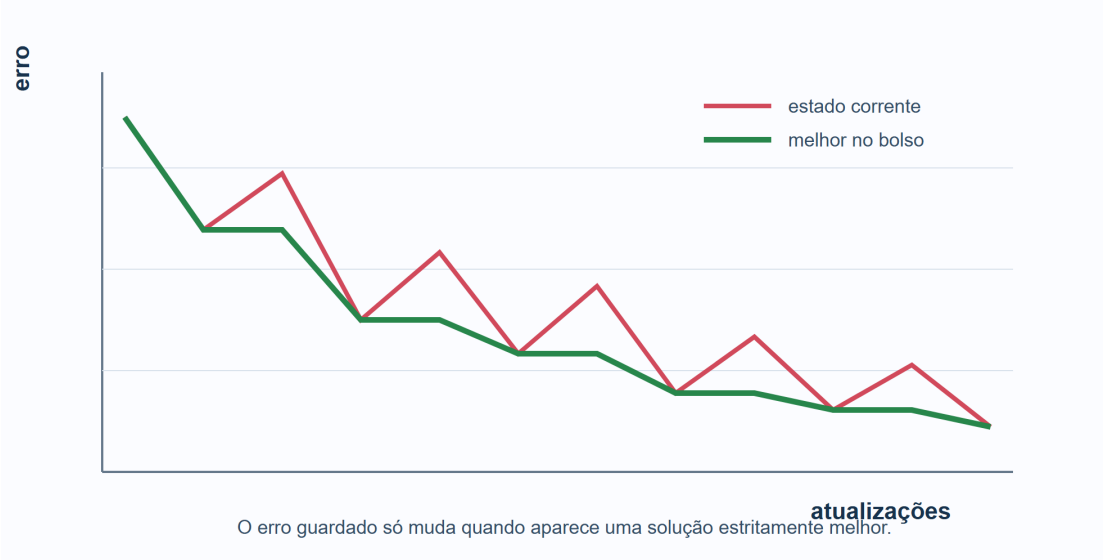


Figura 6.7: O estado corrente pode piorar, enquanto o menor erro guardado pelo Pocket nunca aumenta.

É indispensável **copiar** o vetor. Em Python, a atribuição `melhor_w = w` pode criar apenas outra referência para o mesmo array; atualizações futuras alterariam também o conteúdo do bolso.

6.2.37.2 Pseudocódigo

```
inicialize w_atual
w_melhor ← cópia de w_atual
erro_melhor ← erro_de_treino(w_melhor)

repita até atingir o limite de atualizações:
    escolha um exemplo com margem não positiva
    se nenhum existir:
        encerre: erro zero foi alcançado

    aplique uma atualização do Perceptron em w_atual
    erro_atual ← erro_de_treino(w_atual)

    se erro_atual < erro_melhor:
        w_melhor ← cópia de w_atual
        erro_melhor ← erro_atual
```

```
devolva w_melhor
```

Um limite de atualizações é parte da especificação, não apenas uma proteção acidental: em dados não separáveis, não existe um instante natural no qual o Perceptron necessariamente pare.

6.2.37.3 Implementação

```
import numpy as np


def erro_classificacao(X, y, w, b):
    predicoes = np.where(X @ w + b >= 0, 1, -1)
    return np.mean(predicoes != y)


def treinar_pocket(
    X,
    y,
    *,
    max_atualizacoes=10_000,
    taxa=1.0,
    semente=0,
):
    X = np.asarray(X, dtype=float)
    y = np.asarray(y, dtype=int)
    rng = np.random.default_rng(semente)

    w = np.zeros(X.shape[1])
    b = 0.0
    melhor_w = w.copy()
    melhor_b = b
    melhor_erro = erro_classificacao(X, y, w, b)

    for atualizacao in range(1, max_atualizacoes + 1):
        margens = y * (X @ w + b)
        candidatos = np.flatnonzero(margens <= 0)
```

```
        if len(candidatos) == 0:
            return melhor_w, melhor_b, melhor_erro, atualizacao - 1

        i = rng.choice(candidatos)
        w += taxa * y[i] * X[i]
        b += taxa * y[i]

        erro_atual = erro_classificacao(X, y, w, b)
        if erro_atual < melhor_erro:
            melhor_w = w.copy()
            melhor_b = b
            melhor_erro = erro_atual

    return melhor_w, melhor_b, melhor_erro, max_atualizacoes
```

Essa versão escolhe aleatoriamente um exemplo problemático. O uso de uma semente permite reproduzir a trajetória. Em caso de empate no erro, poderíamos manter o vetor mais antigo, preferir a maior margem média ou usar um critério de validação; a decisão deve ser documentada.

6.2.37.4 Custo computacional

Uma atualização custa $O(d)$, mas recalcular o erro sobre todos os exemplos custa $O(Nd)$. Se essa avaliação ocorrer depois de cada uma das T atualizações, o custo é $O(TNd)$. Por isso, implementações para bases grandes podem avaliar o bolso a cada certo número de passos ou manter estruturas auxiliares. Essas otimizações alteram a frequência com que bons estados podem ser registrados.

6.2.37.5 O que o Pocket garante — e o que não garante

O algoritmo devolve uma solução **não pior que qualquer estado que ele tenha guardado**, segundo o critério usado. Entretanto:

- ele não garante encontrar o mínimo global da perda zero–um;
- mais iterações ampliam a busca, mas não estabelecem quando o ótimo foi encontrado;
- escolher pelo erro de treinamento não garante o menor erro fora da amostra;
- testar repetidamente no conjunto de teste para decidir o bolso causa vazamento de dados.

Se desejarmos escolher o número de atualizações ou desempatar modelos com base em desempenho externo, devemos usar um conjunto de validação e reservar o teste para a avaliação final.

6.2.37.6 Perceptron versus Pocket

Aspecto	Perceptron clássico	Pocket
estado devolvido	último estado	melhor estado observado
separável	converge para erro zero	também pode alcançar erro zero
não separável	pode oscilar	devolve uma aproximação encontrada
memória extra	$O(d)$	$O(d)$ para a cópia adicional
avaliação de erro	não é necessária a cada passo	usualmente custa $O(Nd)$
garantia de ótimo global	não	não

> **!** O bolso preserva, mas não otimiza sozinho
>
> O mérito do Pocket é não perder uma boa solução durante uma trajetória oscilante. A qualidade final ainda depende dos estados explorados pelo Perceptron, do orçamento de atualizações e da ordem dos exemplos.

6.2.38 Recapitulação

O Perceptron reúne em um único exemplo várias ideias fundamentais do aprendizado de máquina. Os dados fornecem exemplos rotulados; uma classe de hipóteses delimita as fronteiras permitidas; uma medida de erro avalia as previsões; e um algoritmo ajusta parâmetros com base nos exemplos em que a hipótese atual falha.

6.2.38.1 Mapa da seção

Elemento	Formulação no Perceptron
entrada	$x \in \mathbb{R}^d$
rótulo	$y \in \{-1, +1\}$
pontuação	$s(x) = w^\top x + b$
hipótese	$h(x) = \text{sinal}_+(s(x))$
fronteira	$w^\top x + b = 0$
margem funcional	$y(w^\top x + b)$
condição de atualização	$y(w^\top x + b) \leq 0$
atualização	$w \leftarrow w + \eta y x$, $b \leftarrow b + \eta y$
erro empírico	média de $\mathbb{1}[h(x_n) \neq y_n]$

Elemento	Formulação no Perceptron
hipótese de convergência	amostra linearmente separável
cota de atualizações	$T \leq (R/\gamma)^2$

O vetor w é perpendicular à fronteira, enquanto b controla seu deslocamento. Adicionar uma coordenada constante permite escrever ambos como um único vetor homogêneo $\widetilde{w}$.

6.2.38.2 O fluxo do treinamento

1. Inicialize os parâmetros.
2. Calcule a margem de um exemplo.
3. Se a margem não for positiva, atualize na direção yx.
4. Repita até completar uma época sem atualizações ou atingir o limite de execução.
5. Avalie a hipótese resultante em dados que não participaram do ajuste.

Para dados separáveis, o Perceptron termina com erro de treinamento zero. Para dados não separáveis, o estado corrente pode oscilar; o Pocket preserva o estado de menor erro encontrado durante uma execução finita.

⚠ Quatro conclusões que não são válidas

- Uma atualização não precisa diminuir o erro total da amostra.
- Convergir não significa encontrar a fronteira de maior margem.
- Erro de treinamento zero não implica erro de teste zero.
- Um peso de grande módulo não prova que o atributo é causalmente importante.

6.2.38.3 Questões conceituais

1. Qual é a diferença entre a perda zero–um e o critério de atualização baseado em margem não positiva?
2. Por que $y(w^{\mathsf{T}}x + b) > 0$ representa corretamente as duas classes em uma única desigualdade?
3. O que acontece geometricamente quando alteramos apenas b? E quando alteramos a direção de w?
4. Por que a atribuição `melhor_w = w` pode invalidar uma implementação do Pocket em NumPy?
5. Em qual hipótese o teorema de convergência depende? Dê um exemplo de amostra que viola essa hipótese.

6. O número de erros necessariamente diminui após cada atualização? Justifique com a natureza local da regra.

6.2.38.4 Problemas de aplicação

1. **Uma atualização.** Para $x = (2, -1)$, $y = -1$, $w = (1, 3)$, $b = 0$ e $\eta = 0{,}5$, calcule a margem. Determine se há atualização e, se houver, obtenha os novos parâmetros.
2. **Fronteira.** Escreva a equação da reta definida por $w = (2, -4)$ e $b = 8$. Identifique inclinação e intercepto.
3. **XOR.** Represente os quatro pontos binários $(0, 0), (0, 1), (1, 0), (1, 1)$, rotulando como positiva a paridade ímpar. Explique geometricamente por que uma única reta não separa as classes.
4. **Complexidade.** Uma base possui $N = 50\,000$ exemplos e $d = 200$ atributos. Quantos produtos peso–atributo são necessários, aproximadamente, para dez épocas completas?
5. **Cota.** Se $R = 5$ e $\gamma = 0{,}25$, calcule a cota de atualizações. Explique por que o valor não prediz exatamente o tempo observado.

6.2.38.5 Respostas breves para conferência

1. No primeiro problema, a pontuação é -1, a margem é 1 e não há atualização.
2. A reta é $2x_1 - 4x_2 + 8 = 0$, ou $x_2 = 0{,}5x_1 + 2$.
3. No XOR, classes iguais ocupam vértices opostos; qualquer reta deixa pelo menos um vértice no lado incompatível.
4. Dez épocas exigem aproximadamente $10 \cdot 50\,000 \cdot 200 = 10^8$ multiplicações, além de somas e outros custos.
5. A cota é $(5/0{,}25)^2 = 400$ atualizações e representa um pior caso sob as hipóteses do teorema.

Com esse vocabulário consolidado, podemos passar da decisão binária para a previsão de valores reais. A regressão linear manterá a pontuação $w^{\mathsf{T}}x + b$, mas substituirá a função sinal e a regra de atualização por uma função de erro diferenciável.

6.3 4.2 A Regressão Linear

A classificação responde a perguntas categóricas, como “aprovar ou rejeitar?”. Muitos sistemas, porém, precisam prever uma **quantidade**: o consumo de energia na próxima hora, o tempo de execução de um programa, o preço de um imóvel ou o limite de crédito adequado a um cliente. A regressão linear é o modelo mais simples para esse tipo de saída.

Em um problema de regressão, cada exemplo é um par

$$(x_n, y_n), \qquad x_n \in \mathbb{R}^d, \quad y_n \in \mathbb{R},$$

em que x_n contém os atributos e y_n é o valor-alvo observado. A amostra é

$$D = \{(x_1, y_1), \dots, (x_N, y_N)\}.$$

O modelo atribui a cada entrada uma previsão real

$$\hat{y} = h_{w,b}(x) = w^{\mathsf{T}} x + b. \tag{6.15}$$

O símbolo "chapéu" distingue a previsão $\hat{y}$ do valor observado y. A diferença

$$e_n = y_n - \hat{y}_n$$

é o **resíduo** do exemplo n. Resíduo positivo significa que o modelo previu abaixo do valor observado; resíduo negativo significa que previu acima.

Figura 6.8: A regressão escolhe uma reta que aproxima os valores observados; os segmentos tracejados são resíduos.

6.3.1 Da reta ao hiperplano

Com um único atributo, Equação 6.15 descreve uma reta:

$$\hat{y} = b + wx.$$

O peso w é a inclinação: aumentar x em uma unidade altera a previsão em w unidades. O viés b é a previsão quando $x = 0$, embora essa interpretação só seja relevante se zero fizer sentido no domínio.

Com dois atributos, o gráfico da previsão é um plano; com d atributos, é um hiperplano em dimensão $d + 1$. Não conseguimos visualizá-lo diretamente quando d é grande, mas a álgebra permanece a mesma.

6.3.2 Um exemplo: tempo de execução

Suponha que desejemos prever o tempo de um algoritmo a partir do tamanho da entrada n e de uma medida de densidade r. Um modelo possível é

$$\hat{t} = 0{,}4 + 0{,}015n + 1{,}8r.$$

Para $n = 100$ e $r = 0{,}5$, a previsão é

$$\hat{t} = 0{,}4 + 0{,}015(100) + 1{,}8(0{,}5) = 2{,}8.$$

Se o tempo observado for $3{,}1$ segundos, o resíduo é $3{,}1 - 2{,}8 = 0{,}3$ segundo. Uma única diferença não determina a qualidade do modelo; precisamos resumir os resíduos de toda a amostra.

6.3.3 O que significa “melhor reta”?

Em geral, nenhuma reta passa por todos os pontos. Precisamos definir como comparar erros positivos e negativos e quanto penalizar grandes desvios. A regressão linear clássica escolhe os parâmetros que minimizam a soma ou a média dos **quadrados dos resíduos**:

$$\mathrm{MSE}(w, b) = \frac{1}{N} \sum_{n=1}^{N} \left(y_n - w^{\mathsf{T}} x_n - b \right)^2 .$$

O quadrado impede cancelamento entre sinais e penaliza fortemente erros grandes. Além disso, produz uma função diferenciável e convexa dos parâmetros, permitindo obter uma solução global por álgebra linear ou por métodos iterativos.

> **Linear nos parâmetros**
>
> Uma regressão pode incluir atributos transformados e continuar linear nos parâmetros. Por exemplo, $\hat{y} = w_0 + w_1 x + w_2 x^2$ é curva em x, mas é linear no vetor (w_0, w_1, w_2) quando usamos os atributos $(1, x, x^2)$.

6.3.4 Previsão, associação e causalidade

Um coeficiente positivo indica que, dentro do modelo e mantendo os outros atributos constantes, valores maiores daquele atributo estão associados a previsões maiores. Isso não demonstra causalidade. Uma regressão pode usar correlações úteis para prever mesmo quando não identifica o processo que gera o resultado.

Em decisões de crédito, saúde ou políticas públicas, essa distinção é crítica: dados históricos podem refletir desigualdades, decisões humanas anteriores e variáveis omitidas. Desempenho preditivo deve ser analisado junto com procedência dos dados, métricas por subgrupo e consequências do uso do modelo.

6.3.5 Suposições e limites iniciais

A regressão linear é uma aproximação apropriada quando uma combinação linear captura parte relevante do sinal. Problemas comuns incluem:

- **não linearidade:** os resíduos exibem um padrão sistemático;
- **valores atípicos:** o quadrado dá influência elevada a erros grandes;
- **extrapolação:** previsões longe da faixa observada podem ser implausíveis;
- **multicolinearidade:** atributos quase redundantes tornam coeficientes instáveis;
- **mudança de distribuição:** relações aprendidas no passado podem não persistir em produção.

As próximas subseções formalizam a classe de hipóteses, a função de erro e os dois caminhos principais para calcular os parâmetros: solução algébrica por mínimos quadrados e otimização iterativa.

6.3.6 Hipóteses

Nesta subseção, **hipótese** significa uma função candidata dentro da classe de modelos. Não deve ser confundida com as hipóteses estatísticas sobre ruído, independência ou variância que podem ser usadas para fazer inferência sobre os coeficientes.

Para d atributos, a classe da regressão linear com intercepto é

$$\mathcal{H}_{mathrmlin} = \left\{ h_{w,b} : \mathbb{R}^d \to \mathbb{R} \;\middle|\; h_{w,b}(x) = w^{\mathsf{T}} x + b,\ w \in \mathbb{R}^d,\ b \in \mathbb{R} \right\}. \tag{6.16}$$

Cada escolha de (w, b) define uma hipótese diferente. O algoritmo de aprendizado selecionará uma delas usando a amostra.

6.3.7 Representação homogênea

Como no Perceptron, podemos introduzir

$$\tilde{x} = (1, x_1, \dots, x_d), \qquad \widetilde{w} = (b, w_1, \dots, w_d),$$

e escrever

$$h_{\widetilde{w}}(x) = \tilde{x}^{\mathsf{T}} \widetilde{w}. \tag{6.17}$$

A ordem do produto é apenas uma convenção quando ambos representam vetores e o resultado é escalar. Para manter coerência com a matriz de projeto, trataremos cada exemplo homogêneo como um vetor linha. Assim, $\tilde{x}^{\mathsf{T}} \widetilde{w}$ na notação abstrata corresponde a uma linha multiplicada pelo vetor coluna de parâmetros na implementação.

Mais precisamente, se $\tilde{x}$ for definido como vetor coluna, usamos $\tilde{x}^{\mathsf{T}} \widetilde{w}$; se for armazenado como linha, a transposição já está implícita. O importante é verificar as dimensões, não decorar a posição de um sobrescrito.

6.3.8 Da hipótese individual à matriz de projeto

Organizamos os N exemplos nas linhas da **matriz de projeto**:

$$\widetilde{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1d} \\ 1 & x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & x_{N2} & \cdots & x_{Nd} \end{bmatrix} \in \mathbb{R}^{N \times (d+1)}.$$

O vetor de alvos é

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \in \mathbb{R}^N,$$

e o vetor de previsões de uma hipótese é

$$\hat{y} = \widetilde{X}\widetilde{w} \in \mathbb{R}^N. \tag{6.18}$$

As dimensões confirmam a operação:

$$\underbrace{\widetilde{X}}_{N\times(d+1)} \quad \underbrace{\widetilde{w}}_{(d+1)\times 1} = \underbrace{\hat{y}}_{N\times 1}.$$

Cada linha da multiplicação reproduz $\hat{y}_n = b + w_1 x_{n1} + \cdots + w_d x_{nd}$.

6.3.9 Vetor de resíduos

Reunindo os resíduos de todos os exemplos,

$$e = y - \hat{y} = y - \widetilde{X}\widetilde{w}. \tag{6.19}$$

A regressão por mínimos quadrados escolherá $\widetilde{w}$ para tornar a norma $\|e\|_2$ pequena. Geometricamente, ela mede desvios na direção do eixo de saída y, não a menor distância perpendicular entre cada ponto e a reta. Essa distinção é importante: a regressão linear usual supõe que os atributos são dados e o erro está associado à variável de saída.

6.3.10 O papel de cada coeficiente

Para

$$h(x) = b + w_1 x_1 + \cdots + w_d x_d,$$

o coeficiente w_j é a variação na previsão causada por aumentar x_j em uma unidade **mantendo os demais atributos constantes**. Suas unidades são

$$[w_j] = \frac{[y]}{[x_j]}.$$

Se y é medido em reais e x_j em metros quadrados, por exemplo, w_j é medido em reais por metro quadrado. Padronizar atributos muda a escala numérica dos coeficientes, mas não precisa mudar as previsões após a transformação inversa correta.

> ⚠ Intercepto não deve ser removido automaticamente
>
> Forçar $b = 0$ obriga o hiperplano a passar pela origem. Isso só é justificável quando a teoria do problema exige saída zero para entrada zero. Caso contrário, a restrição pode introduzir viés sistemático nos resíduos.

6.3.11 Linearidade e engenharia de atributos

A classe pode ser enriquecida substituindo x por um vetor de características $\phi(x)$:

$$h_w(x) = w^{\mathsf{T}}\phi(x).$$

Com $\phi(x) = (1, x, x^2)$, obtemos uma regressão polinomial quadrática. Com indicadores binários, podemos representar categorias; com produtos $x_i x_j$, podemos representar interações. O modelo continua linear nos parâmetros w, embora a relação com a entrada original possa ser curva.

Essa flexibilidade tem custo: adicionar muitas características aumenta a capacidade do modelo, o consumo de memória e o risco de sobreajuste. Transformações devem ser definidas usando apenas informações disponíveis no instante da previsão.

6.3.11.1 Verificação computacional das formas

```
import numpy as np

X = np.array([
    [100.0, 0.2],
    [250.0, 0.7],
    [400.0, 0.9],
])
y = np.array([1.8, 5.2, 7.1])
```

```
X_tilde = np.column_stack([np.ones(len(X)), X])
w_tilde = np.array([0.4, 0.015, 1.8])
y_predito = X_tilde @ w_tilde
residuos = y - y_predito

print(X_tilde.shape)   # (3, 3)
print(w_tilde.shape)   # (3,)
print(y_predito.shape) # (3,)
```

Em NumPy, um array com `shape == (3,)` não é explicitamente linha nem coluna. O operador `@` aplica as regras adequadas, mas, ao combinar matrizes mais complexas, declarar formas como `(3, 1)` pode tornar a intenção mais clara.

6.3.12 Erro

A função de erro transforma um vetor de resíduos em um único número. Na regressão linear clássica, usamos o **erro quadrático médio** (*mean squared error*, MSE):

$$E_{\text{in}}(\widetilde{w}) = \text{MSE}(\widetilde{w}) = \frac{1}{N}\sum_{n=1}^{N}\left(y_n - \widetilde{x}_n^{\mathsf{T}}\widetilde{w}\right)^2. \tag{6.20}$$

A palavra "médio" corresponde ao fator $1/N$. Não há uma integral sobre a amostra finita; há uma média empírica das perdas individuais.

O quadrado desempenha três papéis:

- torna todos os termos não negativos, impedindo o cancelamento entre resíduos positivos e negativos;
- penaliza resíduos grandes mais intensamente;
- produz uma função suave e convexa dos parâmetros.

6.3.13 Soma, média e raiz do erro quadrático

A **soma dos quadrados dos resíduos** é

$$\text{SSE}(\widetilde{w}) = \sum_{n=1}^{N} e_n^2.$$

Como $\text{MSE} = \text{SSE}/N$ e N não depende dos parâmetros, SSE e MSE possuem exatamente os mesmos minimizadores. Seus valores, porém, não são intercambiáveis ao relatar resultados.

A raiz do MSE é

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{n=1}^{N} e_n^2}.$$

MSE tem unidades de y^2, enquanto RMSE tem as mesmas unidades do alvo. Se o alvo é medido em segundos, por exemplo, o RMSE também é expresso em segundos e costuma ser mais fácil de interpretar.

6.3.14 Forma matricial

Pela Equação 6.19,

$$e = y - \widetilde{X}\widetilde{w}.$$

A norma euclidiana satisfaz

$$\|e\|_2^2 = e^{\mathsf{T}} e = \sum_{n=1}^{N} e_n^2.$$

Assim, Equação 6.20 pode ser escrita de modo compacto:

$$E_{\text{in}}(\widetilde{w}) = \frac{1}{N}\left\|y - \widetilde{X}\widetilde{w}\right\|_2^2. \tag{6.21}$$

Expandindo o produto,

$$E_{\text{in}}(\widetilde{w}) = \frac{1}{N}\left(y^{\mathsf{T}} y - 2\widetilde{w}^{\mathsf{T}}\widetilde{X}^{\mathsf{T}} y + \widetilde{w}^{\mathsf{T}}\widetilde{X}^{\mathsf{T}}\widetilde{X}\widetilde{w}\right). \tag{6.22}$$

Essa é uma função quadrática de $\widetilde{w}$. Sua matriz Hessiana é proporcional a $\widetilde{X}^{\mathsf{T}}\widetilde{X}$, que é positiva semidefinida. Portanto, todo mínimo local é global. A unicidade dependerá do posto da matriz de projeto, assunto da subseção sobre o algoritmo.

6.3.15 MSE versus MAE

Outra medida comum é o **erro absoluto médio** (*mean absolute error*, MAE):

$$\mathrm{MAE} = \frac{1}{N}\sum_{n=1}^{N} |e_n|.$$

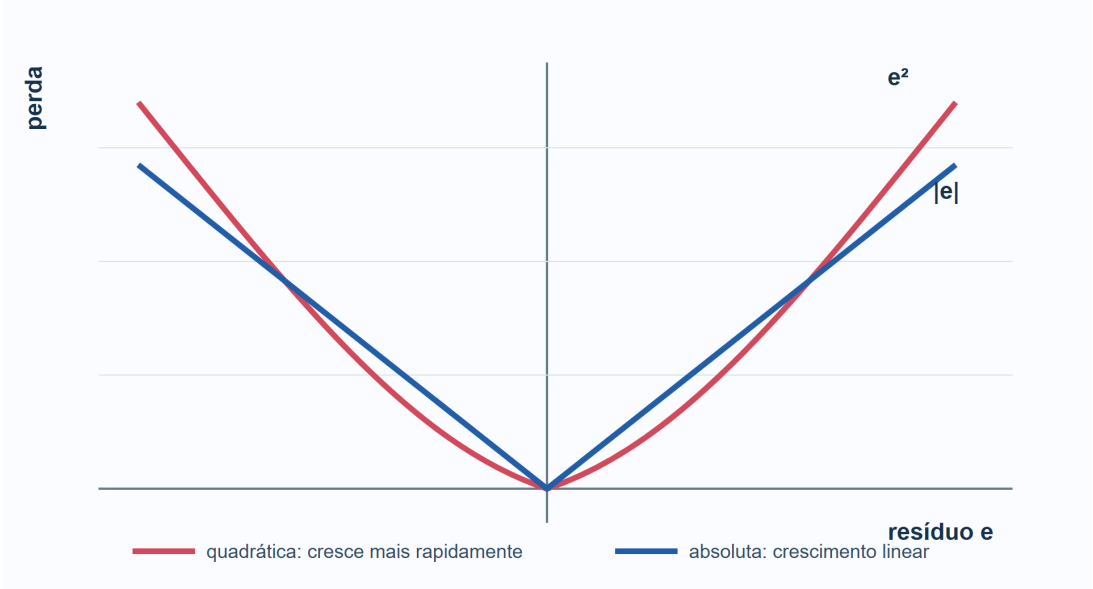


Figura 6.9: A perda quadrática atribui influência crescente aos resíduos de grande módulo.

Propriedade	MSE/RMSE	MAE
crescimento da perda	quadrático	linear
sensibilidade a valores atípicos	maior	menor
diferenciabilidade em zero	sim	não
estimativa central associada	média condicional	mediana condicional
unidade reportada	MSE: quadrada; RMSE: original	original

Nenhuma métrica é universalmente superior. A escolha deve refletir o custo real dos erros. Se dobrar o erro deve custar quatro vezes mais, o quadrado é coerente; se o custo cresce aproximadamente de modo linear, o MAE pode representar melhor a aplicação.

6.3.16 Efeito de um valor atípico

Considere resíduos $(1, -1, 2, 10)$. Temos

$$\mathrm{MAE} = \frac{1+1+2+10}{4} = 3{,}5,$$

enquanto

$$\text{MSE} = \frac{1 + 1 + 4 + 100}{4} = 26{,}5.$$

O último exemplo responde por $100/106 \approx 94{,}3\%$ da soma quadrática. Isso pode ser desejável quando grandes erros são realmente graves, ou prejudicial quando o ponto resulta de falha de medição. Valores atípicos devem ser investigados, não removidos automaticamente.

6.3.17 Coeficiente de determinação

Uma métrica complementar é

$$R^2 = 1 - \frac{\sum_n (y_n - \hat{y}_n)^2}{\sum_n (y_n - \overline{y})^2},$$

em que $\overline{y}$ é a média dos alvos do conjunto avaliado. O numerador é o erro do modelo e o denominador é o erro do preditor constante $\overline{y}$.

- $R^2 = 1$ indica previsões perfeitas naquele conjunto;
- $R^2 = 0$ indica desempenho igual ao preditor da média;
- $R^2 < 0$ significa que o modelo é pior que esse preditor de referência.

Logo, R^2 não é uma porcentagem de previsões corretas e não precisa ficar entre zero e um em dados de teste.

6.3.18 Avaliação correta

Métricas de treinamento medem ajuste, não generalização. MSE, RMSE, MAE e R^2 devem ser calculados também em dados não usados no treinamento. Ao comparar modelos, use a mesma partição e o mesmo pré-processamento.

> ⚠ Não escolha a métrica depois de ver o resultado
>
> Selecionar a métrica que faz um modelo parecer melhor introduz viés na avaliação. Defina antecipadamente a medida principal com base no custo da aplicação e use métricas secundárias para diagnóstico.

6.3.18.1 Verificação computacional

```
import numpy as np

y = np.array([3.0, 5.0, 7.0, 20.0])
y_predito = np.array([2.0, 6.0, 5.0, 10.0])
residuos = y - y_predito

mse = np.mean(residuos ** 2)
rmse = np.sqrt(mse)
mae = np.mean(np.abs(residuos))

print(f"MSE = {mse:.2f}")
print(f"RMSE = {rmse:.2f}")
print(f"MAE = {mae:.2f}")
```

No próximo passo, minimizaremos Equação 6.21 para obter os coeficientes da regressão.

6.3.19 Algoritmo

Treinar a regressão linear significa resolver o problema de mínimos quadrados

$$\widetilde{w}^* \in \arg\min_{\widetilde{w}\in\mathbb{R}^{d+1}} \frac{1}{N} \left\| y - \widetilde{X}\widetilde{w} \right\|_2^2 . \tag{6.23}$$

Escrevemos arg min porque o resultado é o conjunto de parâmetros que atingem o menor valor. Quando a solução é única, podemos tratá-lo como um único vetor $\widetilde{w}^*$.

O fator $1/N$ não altera o minimizador, por isso as derivações podem trabalhar com a soma dos quadrados. Ele continua importante ao comparar o valor do erro entre amostras de tamanhos diferentes.

6.3.20 Visão geral da solução

Sob condições que garantem a inversibilidade de $\widetilde{X}^\intercal \widetilde{X}$, a solução pode ser escrita como

$$\widetilde{w}^* = \left(\widetilde{X}^\intercal \widetilde{X}\right)^{-1} \widetilde{X}^\intercal y. \tag{6.24}$$

Essa expressão é chamada solução das **equações normais**. Ela será obtida de duas maneiras:

1. **geometricamente:** $\widetilde{X}\widetilde{w}^*$ é a projeção ortogonal de y sobre o espaço gerado pelas colunas de $\widetilde{X}$;
2. **por diferenciação:** no mínimo, o gradiente do erro em relação aos parâmetros é zero.

Os dois argumentos produzem a mesma condição:

$$\widetilde{X}^{\mathsf{T}} \left(y - \widetilde{X}\widetilde{w}^* \right) = 0. \tag{6.25}$$

Essa equação diz que o vetor de resíduos é ortogonal a cada coluna da matriz de projeto. Em particular, quando há uma coluna de uns para o intercepto, a soma dos resíduos de treinamento é zero, salvo efeitos de arredondamento numérico.

6.3.21 Fórmula matemática versus algoritmo numérico

É tentador implementar diretamente a inversa de $\widetilde{X}^{\mathsf{T}}\widetilde{X}$. Em software numérico, essa raramente é a melhor opção. Formar o produto pode piorar o condicionamento, e calcular uma inversa explícita realiza trabalho desnecessário.

As estratégias usuais incluem:

- resolver o sistema linear das equações normais sem formar a inversa;
- usar fatoração QR, geralmente mais estável;
- usar decomposição em valores singulares (SVD), que também trata posto deficiente e produz a pseudoinversa;
- aplicar gradiente descendente ou métodos estocásticos quando a matriz é grande demais para a solução direta.

Assim, “regressão linear possui fórmula fechada” não significa que toda implementação deva calcular essa fórmula literalmente.

6.3.22 Quando a solução é única?

Se as $d + 1$ colunas de $\widetilde{X}$ são linearmente independentes, então

$$\text{posto}(\widetilde{X}) = d + 1$$

e $\widetilde{X}^{\mathsf{T}}\widetilde{X}$ é positiva definida e invertível. Nesse caso, o minimizador é único.

Se há uma coluna redundante, mais atributos que informação independente ou categorias codificadas de forma redundante, a matriz pode ter posto deficiente. Ainda existem previsões

de mínimos quadrados, mas vários vetores de coeficientes podem produzir o mesmo ajuste. A pseudoinversa de Moore–Penrose seleciona, entre eles, a solução de menor norma.

> **!** Nunca use a inversa como teste de existência
>
> Se `inv(X.T @ X)` falhar, isso não significa que a regressão não possa ser ajustada. Use um solucionador de mínimos quadrados, como `numpy.linalg.lstsq`, que lida com matrizes retangulares e informa o posto estimado.

6.3.23 Complexidade em linhas gerais

Para N exemplos e $p = d + 1$ parâmetros, armazenar a matriz densa custa $O(Np)$. Uma solução direta por QR custa tipicamente $O(Np^2)$ quando $N \geq p$, além de $O(p^2)$ de memória auxiliar. Em problemas com muitos exemplos ou muitos atributos esparsos, métodos iterativos podem ser mais adequados.

Também é preciso considerar o condicionamento. Atributos em escalas muito diferentes ou quase colineares podem tornar os coeficientes extremamente sensíveis a pequenas perturbações, mesmo quando o cálculo termina sem erro.

6.3.24 Transposição e dimensões

Para $A \in \mathbb{R}^{m \times n}$, a transposta $A^{\mathsf{T}} \in \mathbb{R}^{n \times m}$ troca linhas por colunas. No nosso problema,

$$\widetilde{X}^{\mathsf{T}}\widetilde{X} \in \mathbb{R}^{p \times p}, \qquad \widetilde{X}^{\mathsf{T}}y \in \mathbb{R}^{p}.$$

Portanto, as equações normais formam um sistema quadrado com p incógnitas. Anotar essas dimensões é uma verificação simples contra erros de orientação em código e em demonstrações.

As duas subseções seguintes desenvolvem primeiro a interpretação por projeção e depois a derivação pelo gradiente.

6.3.24.1 Aproximação por Projeção

A interpretação por projeção ocorre em $\mathbb{R}^N$, o espaço que possui uma coordenada para cada exemplo. O vetor-alvo

$$y = (y_1, \dots, y_N)^{\mathsf{T}}$$

é um ponto nesse espaço. Para cada escolha de parâmetros, o vetor de previsões é $\widetilde{X}\widetilde{w}$. Quais vetores de previsão são possíveis?

6.3.24.1.1 O espaço coluna

Escreva a matriz de projeto em termos de suas colunas:

$$\widetilde{X} = \begin{bmatrix} | & | & & | \\ c_0 & c_1 & \cdots & c_d \\ | & | & & | \end{bmatrix}.$$

Então

$$\widetilde{X}\widetilde{w} = w_0 c_0 + w_1 c_1 + \cdots + w_d c_d.$$

Logo, o conjunto de todas as previsões possíveis é o espaço gerado pelas colunas de $\widetilde{X}$:

$$\text{Col}(\widetilde{X}) = \{\widetilde{X}\widetilde{w} : \widetilde{w} \in \mathbb{R}^{d+1}\} \subseteq \mathbb{R}^N.$$

Mesmo que o modelo tenha poucos parâmetros, cada previsão sobre a amostra inteira é um vetor com N componentes. Por isso, o hiperplano da figura original no espaço de atributos e o espaço coluna usado nesta derivação são objetos diferentes.

6.3.24.1.2 A previsão ótima é uma projeção

Minimizar

$$\left\| y - \widetilde{X}\widetilde{w} \right\|_2$$

significa encontrar o ponto de $\text{Col}(\widetilde{X})$ mais próximo de y. Esse ponto é a projeção ortogonal $\hat{y} = P_{widetildeX} y$.

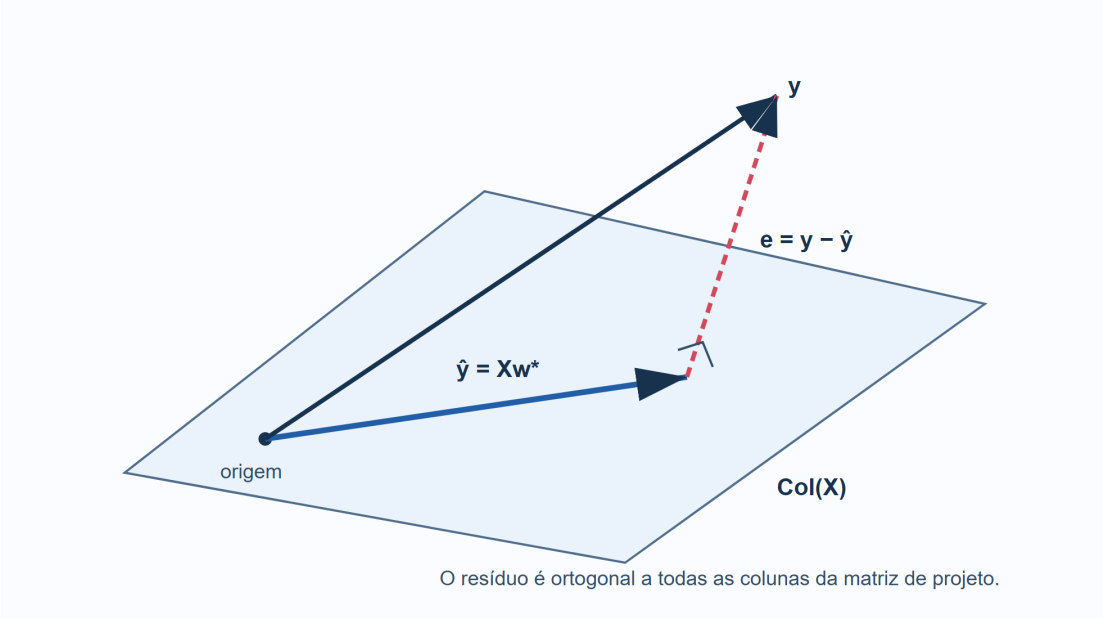


Figura 6.10: O vetor-alvo é decomposto na previsão, que pertence ao espaço coluna, e no resíduo ortogonal.

O resíduo ótimo

$$e^* = y - \hat{y} = y - \widetilde{X}\widetilde{w}^*$$

é perpendicular a todo vetor do espaço coluna. Em particular, é perpendicular a cada coluna de $\widetilde{X}$:

$$\widetilde{X}^\mathsf{T} e^* = 0.$$

Substituindo a definição do resíduo,

$$\widetilde{X}^\mathsf{T}\left(y - \widetilde{X}\widetilde{w}^*\right) = 0,$$

e obtemos as equações normais

$$\widetilde{X}^\mathsf{T}\widetilde{X}\widetilde{w}^* = \widetilde{X}^\mathsf{T} y. \tag{6.26}$$

6.3.24.1.3 Por que a ortogonalidade garante o mínimo?

Considere qualquer outro vetor de parâmetros $\widetilde{w} = \widetilde{w}^* + \Delta$. Seu resíduo é

$$y - \widetilde{X}\widetilde{w} = e^* - \widetilde{X}\Delta.$$

O primeiro termo é perpendicular ao espaço coluna, e o segundo pertence a esse espaço. Pelo teorema de Pitágoras,

$$\left\|e^* - \widetilde{X}\Delta\right\|_2^2 = \|e^*\|_2^2 + \|\widetilde{X}\Delta\|_2^2 \geq \|e^*\|_2^2.$$

Portanto, nenhuma outra previsão possível fica mais perto de y. A igualdade pode ocorrer para $\Delta \neq 0$ somente se $\widetilde{X}\Delta = 0$, isto é, se a matriz possui um núcleo não trivial e os parâmetros não são únicos.

6.3.24.1.4 Caso de posto completo

Se as colunas de $\widetilde{X}$ são linearmente independentes, $\widetilde{X}^\mathsf{T}\widetilde{X}$ é invertível. Isolando os parâmetros em Equação 6.26,

$$\widetilde{w}^* = \left(\widetilde{X}^\mathsf{T}\widetilde{X}\right)^{-1}\widetilde{X}^\mathsf{T}y.$$

A matriz de projeção sobre o espaço coluna é, nesse caso,

$$P_{widetildeX} = \widetilde{X}\left(\widetilde{X}^\mathsf{T}\widetilde{X}\right)^{-1}\widetilde{X}^\mathsf{T},$$

pois $\hat{y} = \widetilde{X}\widetilde{w}^* = P_{\widetilde{X}}y$. Ela satisfaz

$$P_{\widetilde{X}}^\mathsf{T} = P_{\widetilde{X}}, \qquad P_{\widetilde{X}}^2 = P_{\widetilde{X}},$$

propriedades de simetria e idempotência características de uma projeção ortogonal.

6.3.24.1.5 Posto deficiente e pseudoinversa

Ter $N > d + 1$ não garante independência das colunas. Dois atributos podem ser duplicados ou um pode ser combinação dos demais. Quando $\widetilde{X}^\mathsf{T}\widetilde{X}$ não é invertível, usamos a pseudoinversa de Moore–Penrose:

$$\widetilde{w}^* = \widetilde{X}^+y.$$

Nesse caso, $\widetilde{X}^{+}$ não deve ser definida automaticamente por $(\widetilde{X}^{\mathsf{T}}\widetilde{X})^{-1}\widetilde{X}^{\mathsf{T}}$, pois essa expressão exige justamente a inversibilidade. A definição geral é construída, por exemplo, pela SVD e devolve a solução de mínimos quadrados com menor norma euclidiana.

6.3.24.1.6 Exemplo numérico

Considere um atributo $x = (0, 1, 2)^{\mathsf{T}}$, alvos $y = (1, 2, 2)^{\mathsf{T}}$ e intercepto:

$$\widetilde{X} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix}.$$

As equações normais produzem

$$\widetilde{w}^{*} = \begin{bmatrix} 7/6 \\ 1/2 \end{bmatrix}, \qquad \widehat{y} = \begin{bmatrix} 7/6 \\ 5/3 \\ 13/6 \end{bmatrix}.$$

O vetor de resíduos é

$$e^{*} = y - \widehat{y} = \begin{bmatrix} -1/6 \\ 1/3 \\ -1/6 \end{bmatrix}.$$

Podemos verificar a ortogonalidade com as duas colunas:

$$\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} e^{*} = 0, \qquad \begin{bmatrix} 0 & 1 & 2 \end{bmatrix} e^{*} = 0.$$

O primeiro resultado diz que os resíduos somam zero; o segundo, que não resta componente residual na direção do atributo x.

> **Diagnóstico útil**
>
> Em uma regressão com intercepto ajustada por mínimos quadrados, uma soma de resíduos de treinamento muito diferente de zero pode indicar erro de implementação, falta da coluna de intercepto ou precisão numérica muito ruim.

6.3.24.2 Aproximação por Derivação

A mesma solução pode ser obtida tratando o MSE como uma função diferenciável dos parâmetros. O ponto de partida é

$$E(\widetilde{w}) = \frac{1}{N}\sum_{n=1}^{N}\left(y_n - \tilde{x}_n^{\mathsf{T}}\widetilde{w}\right)^2.$$

Antes de usar notação matricial, derivaremos cada coordenada. Isso torna visível a origem de todos os fatores e sinais.

6.3.24.2.1 Derivação por coordenadas

Para o parâmetro w_j, a regra da cadeia fornece

$$\begin{aligned}\frac{\partial E}{\partial w_j} &= \frac{1}{N}\sum_{n=1}^{N} 2\left(y_n - \tilde{x}_n^{\mathsf{T}}\widetilde{w}\right)\frac{\partial}{\partial w_j}\left(y_n - \tilde{x}_n^{\mathsf{T}}\widetilde{w}\right)\\ &= -\frac{2}{N}\sum_{n=1}^{N}\tilde{x}_{nj}\left(y_n - \tilde{x}_n^{\mathsf{T}}\widetilde{w}\right).\end{aligned}$$

Se definirmos $e = y - \widetilde{X}\widetilde{w}$, reunir todas as derivadas parciais produz

$$\nabla E(\widetilde{w}) = -\frac{2}{N}\widetilde{X}^{\mathsf{T}}e = \frac{2}{N}\widetilde{X}^{\mathsf{T}}\left(\widetilde{X}\widetilde{w} - y\right). \tag{6.27}$$

As dimensões são consistentes: $\widetilde{X}^{\mathsf{T}}$ é $(d+1)\times N$ e o vetor de resíduos é $N\times 1$, portanto o gradiente tem $d+1$ componentes.

6.3.24.2.2 Derivação pela forma quadrática

Também podemos expandir o MSE:

$$\begin{aligned}E(\widetilde{w}) &= \frac{1}{N}\left(y - \widetilde{X}\widetilde{w}\right)^{\mathsf{T}}\left(y - \widetilde{X}\widetilde{w}\right)\\ &= \frac{1}{N}\left(y^{\mathsf{T}}y - 2\widetilde{w}^{\mathsf{T}}\widetilde{X}^{\mathsf{T}}y + \widetilde{w}^{\mathsf{T}}\widetilde{X}^{\mathsf{T}}\widetilde{X}\widetilde{w}\right).\end{aligned}$$

Usamos as identidades

$$(AB)^{\mathsf{T}} = B^{\mathsf{T}}A^{\mathsf{T}}, \qquad (A^{\mathsf{T}})^{\mathsf{T}} = A,$$

e o fato de que um escalar é igual à sua transposta. Para uma matriz simétrica A,

$$\nabla_w(w^\mathsf{T} A w) = 2Aw.$$

Como $\widetilde{X}^\mathsf{T}\widetilde{X}$ é simétrica, chegamos novamente a Equação 6.27.

6.3.24.2.3 Condição de primeira ordem

Em qualquer mínimo diferenciável não restrito, o gradiente deve ser nulo. Igualando Equação 6.27 a zero,

$$\widetilde{X}^\mathsf{T}\left(\widetilde{X}\widetilde{w}^* - y\right) = 0,$$

ou

$$\widetilde{X}^\mathsf{T}\widetilde{X}\widetilde{w}^* = \widetilde{X}^\mathsf{T} y.$$

São exatamente as equações normais obtidas pela projeção. Se $\widetilde{X}^\mathsf{T}\widetilde{X}$ for invertível,

$$\widetilde{w}^* = \left(\widetilde{X}^\mathsf{T}\widetilde{X}\right)^{-1}\widetilde{X}^\mathsf{T} y.$$

Se ela não for invertível, a igualdade do gradiente ainda caracteriza os minimizadores, mas não podemos isolar uma solução única por meio da inversa.

6.3.24.2.4 Por que um ponto estacionário é mínimo global?

Zerar o gradiente, sozinho, não basta para qualquer função: pontos estacionários também podem ser máximos ou selas. Aqui, a Hessiana é

$$\nabla^2 E(\widetilde{w}) = \frac{2}{N}\widetilde{X}^\mathsf{T}\widetilde{X}. \tag{6.28}$$

Para qualquer vetor v,

$$v^\mathsf{T}\nabla^2 E(\widetilde{w})v = \frac{2}{N}v^\mathsf{T}\widetilde{X}^\mathsf{T}\widetilde{X}v = \frac{2}{N}\|\widetilde{X}v\|_2^2 \geq 0.$$

Logo, a Hessiana é positiva semidefinida e E é convexa. Toda solução das equações normais é um mínimo global. Se $\widetilde{X}$ tem posto completo de colunas, a desigualdade é estrita para $v \neq 0$, a

Hessiana é positiva definida e o mínimo é único.

6.3.24.2.5 O gradiente como soma de contribuições

A fórmula

$$\nabla E(\widetilde{w}) = -\frac{2}{N}\sum_{n=1}^{N} e_n \tilde{x}_n$$

mostra que cada exemplo contribui com seu resíduo multiplicado pelos atributos. Um erro grande exerce influência maior; atributos de grande escala também podem produzir componentes grandes no gradiente. Essa é uma razão prática para padronizar entradas antes de usar métodos iterativos.

Se não quisermos resolver o sistema diretamente, podemos usar

$$\widetilde{w}^{(t+1)} = \widetilde{w}^{(t)} - \eta \nabla E(\widetilde{w}^{(t)}),$$

o gradiente descendente. Diferentemente do Perceptron, a atualização agora deriva explicitamente de uma função de erro suave.

6.3.24.2.6 Verificação no caso de uma reta

Para $\widehat{y}_n = b + wx_n$,

$$E(b, w) = \frac{1}{N}\sum_{n}(y_n - b - wx_n)^2.$$

As duas equações de primeira ordem são

$$\frac{\partial E}{\partial b} = -\frac{2}{N}\sum_{n}(y_n - b - wx_n) = 0,$$

$$\frac{\partial E}{\partial w} = -\frac{2}{N}\sum_{n} x_n(y_n - b - wx_n) = 0.$$

A primeira afirma que os resíduos somam zero; a segunda, que os resíduos são ortogonais ao vetor dos valores x_n. São precisamente as duas verificações feitas no exemplo da projeção.

> **Duas linguagens para o mesmo fato**
>
> Na linguagem geométrica, o resíduo é ortogonal ao espaço coluna. Na linguagem do cálculo, o gradiente é zero. Como o gradiente é $-2\widetilde{X}^{\mathsf{T}}e/N$, as afirmações são equivalentes.

6.3.24.3 Exercícios

1. Derive Equação 6.27 usando a convenção $E = \|\widetilde{X}\widetilde{w} - y\|^2/(2N)$. O que muda?
2. Mostre que $\widetilde{X}^{\mathsf{T}}\widetilde{X}$ é sempre simétrica.
3. Explique por que posto deficiente produz uma direção $v \neq 0$ com curvatura zero em Equação 6.28.

6.3.25 Computação

Na implementação, o objetivo é encontrar $\widetilde{w}$ que minimize $\|\widetilde{X}\widetilde{w} - y\|_2$. A forma matemática das equações normais é útil para a teoria, mas o cálculo explícito

$$\left(\widetilde{X}^{\mathsf{T}}\widetilde{X}\right)^{-1}\widetilde{X}^{\mathsf{T}}y$$

não é a estratégia numérica recomendada.

Há duas razões principais. Primeiro, não precisamos da matriz inversa: precisamos apenas resolver um sistema para um vetor. Segundo, formar $\widetilde{X}^{\mathsf{T}}\widetilde{X}$ eleva ao quadrado, aproximadamente, o número de condição da matriz, ampliando erros de arredondamento.

6.3.25.0.1 Implementação recomendada com NumPy

`numpy.linalg.lstsq` resolve diretamente o problema de mínimos quadrados, normalmente por métodos baseados em SVD. Ele aceita matrizes retangulares e também funciona quando as colunas não são independentes.

```
from dataclasses import dataclass

import numpy as np


@dataclass
class ModeloLinear:
    coeficientes: np.ndarray
    intercepto: float
```

```
    posto: int
    valores_singulares: np.ndarray


def ajustar_regressao_linear(X, y):
    X = np.asarray(X, dtype=float)
    y = np.asarray(y, dtype=float)

    if X.ndim != 2:
        raise ValueError("X deve ter forma (n_exemplos, n_atributos)")
    if y.ndim != 1 or len(y) != len(X):
        raise ValueError("y deve ter um valor para cada linha de X")
    if not np.all(np.isfinite(X)) or not np.all(np.isfinite(y)):
        raise ValueError("X e y devem conter somente valores finitos")

    X_tilde = np.column_stack([np.ones(len(X)), X])
    w_tilde, _, posto, singulares = np.linalg.lstsq(
        X_tilde, y, rcond=None
    )

    return ModeloLinear(
        coeficientes=w_tilde[1:],
        intercepto=float(w_tilde[0]),
        posto=int(posto),
        valores_singulares=singulares,
    )


def prever_regressao(modelo, X):
    X = np.asarray(X, dtype=float)
    return X @ modelo.coeficientes + modelo.intercepto
```

O segundo valor devolvido por `lstsq` contém somas de resíduos apenas em certas configurações de forma e posto; por isso, é mais claro calcular as métricas explicitamente a partir das previsões.

```
def metricas_regressao(y, y_predito):
    y = np.asarray(y, dtype=float)
    y_predito = np.asarray(y_predito, dtype=float)
```

```
    residuos = y - y_predito

    mse = np.mean(residuos**2)
    rmse = np.sqrt(mse)
    mae = np.mean(np.abs(residuos))

    denominador = np.sum((y - np.mean(y))**2)
    r2 = np.nan if denominador == 0 else (
        1 - np.sum(residuos**2) / denominador
    )
    return {"mse": mse, "rmse": rmse, "mae": mae, "r2": r2}
```

Quando todos os alvos são iguais, o denominador de R^2 é zero; retornar `NaN` torna explícito que a métrica não está definida nessa amostra.

6.3.25.0.2 Exemplo completo

```
X_treino = np.array([
    [50.0, 0.10],
    [90.0, 0.25],
    [140.0, 0.40],
    [200.0, 0.65],
    [280.0, 0.80],
])
y_treino = np.array([1.1, 1.8, 2.9, 4.1, 5.4])

modelo = ajustar_regressao_linear(X_treino, y_treino)
predicoes = prever_regressao(modelo, X_treino)

print("intercepto:", modelo.intercepto)
print("coeficientes:", modelo.coeficientes)
print("posto:", modelo.posto)
print(metricas_regressao(y_treino, predicoes))
```

Com dois atributos e intercepto, esperamos posto 3. Um valor menor indica dependência linear detectada numericamente. Isso não torna as previsões automaticamente inúteis, mas impede interpretar os coeficientes como uma solução única.

6.3.25.0.3 Entendendo a fatoração QR

Se $\widetilde{X}$ tem posto completo de colunas e sua QR reduzida é

$$\widetilde{X} = QR,$$

então

$$Q \in \mathbb{R}^{N \times p}, \qquad R \in \mathbb{R}^{p \times p}, \qquad Q^{\mathsf{T}} Q = I_p,$$

com $p = d + 1$ e R triangular superior. Substituindo na condição de mínimos quadrados, a solução satisfaz

$$R\widetilde{w} = Q^{\mathsf{T}} y. \tag{6.29}$$

Resolvemos Equação 6.29 por substituição retroativa, sem calcular R^{-1}. Em NumPy:

```
Q, R = np.linalg.qr(X_tilde, mode="reduced")
w_tilde = np.linalg.solve(R, Q.T @ y)
```

A QR é mais estável que resolver equações normais e costuma ser uma boa opção para matrizes densas com posto completo. Bibliotecas maduras usam transformações de Householder ou técnicas equivalentes; implementar Gram–Schmidt clássico como solucionador de produção é desaconselhável por sua sensibilidade a arredondamentos.

6.3.25.0.4 SVD, pseudoinversa e posto

A decomposição em valores singulares escreve

$$\widetilde{X} = U \Sigma V^{\mathsf{T}}.$$

Valores singulares muito pequenos revelam direções quase redundantes. A pseudoinversa é

$$\widetilde{X}^{+} = V \Sigma^{+} U^{\mathsf{T}},$$

onde Σ^{+} inverte apenas os valores singulares tratados como não nulos. Então,

$$\widetilde{w}^{*} = \widetilde{X}^{+} y.$$

A tolerância usada para considerar um valor singular nulo é uma decisão numérica. O argumento `rcond` de `lstsq` controla essa decisão; `None` usa o padrão apropriado da biblioteca.

6.3.25.0.5 Condicionamento

O número de condição em norma 2 é, para posto completo,

$$\kappa_2(\widetilde{X}) = \frac{\sigma_{\max}}{\sigma_{\min}}.$$

Quanto maior κ_2, mais sensível a solução fica a pequenas perturbações. Podemos estimá-lo por

```
condicao = np.linalg.cond(X_tilde)
print("número de condição:", condicao)
```

Um valor elevado não possui um limiar universal de "falha"; sua gravidade depende da precisão numérica e da aplicação. Padronização pode reduzir diferenças de escala, mas não remove redundância estrutural. Regularização, como regressão ridge, pode estabilizar coeficientes ao aceitar um pequeno viés.

6.3.25.0.6 Qual método escolher?

Situação	Estratégia típica
matriz densa moderada	`lstsq` ou QR
posto deficiente ou quase deficiente	SVD/pseudoinversa
muitos exemplos e poucos atributos	QR ou solucionador especializado
dados muito grandes ou em fluxo	gradiente estocástico ou métodos iterativos
matriz esparsa	rotinas próprias para matrizes esparsas
necessidade de estabilização	regularização ridge

6.3.25.0.7 Treino, validação e teste

Ajuste os coeficientes somente no treino. Use validação para escolher transformações, regularização ou outros hiperparâmetros e teste para a avaliação final. Se houver padronização, ajuste média e desvio no treino e reutilize esses valores nos demais conjuntos.

> ⚠ Evite inv(X.T @ X) @ X.T @ y
>
> Além de falhar em posto deficiente, essa expressão forma uma matriz com condicionamento pior e calcula uma inversa desnecessária. Prefira `np.linalg.lstsq(X, y,`

`rcond=None)` ou um solucionador adequado à estrutura dos dados.

6.3.25.1 Verificações após o ajuste

1. Compare o posto devolvido com o número de parâmetros.
2. Examine valores singulares e número de condição.
3. Verifique resíduos e métricas em treino e validação.
4. Inspecione padrões dos resíduos em função das previsões e atributos.
5. Confirme que o mesmo pré-processamento será aplicado em produção.

6.3.26 Resumo

A regressão linear aprende uma função de saída real

$$h_{w,b}(x) = w^{\mathsf{T}} x + b$$

minimizando a média dos quadrados dos resíduos. Embora o modelo seja simples, seu estudo reuniu representação matricial, projeção ortogonal, cálculo multivariável e estabilidade numérica.

6.3.26.1 Mapa conceitual

Elemento	Expressão
matriz de projeto	$\widetilde{X} \in \mathbb{R}^{N\times(d+1)}$
parâmetros	$\widetilde{w} = (b, w_1, \dots, w_d)^{\mathsf{T}}$
previsões	$\hat{y} = \widetilde{X}\widetilde{w}$
resíduos	$e = y - \hat{y}$
objetivo	$\|e\|_2^2/N$
gradiente	$2\widetilde{X}^{\mathsf{T}}(\widetilde{X}\widetilde{w} - y)/N$
condição ótima	$\widetilde{X}^{\mathsf{T}} e = 0$
equações normais	$\widetilde{X}^{\mathsf{T}}\widetilde{X}\widetilde{w} = \widetilde{X}^{\mathsf{T}} y$
solução geral de menor norma	$\widetilde{w}^{*} = \widetilde{X}^{+} y$

A condição $\widetilde{X}^{\mathsf{T}} e = 0$ possui duas leituras equivalentes:

- **geometria:** a previsão é a projeção de y no espaço coluna e o resíduo é perpendicular a esse espaço;
- **cálculo:** o gradiente do MSE é zero no mínimo.

6.3.26.2 Fluxo recomendado de implementação

1. Separe treino, validação e teste.
2. Ajuste transformações de atributos somente no treino.
3. Adicione o intercepto ou configure a biblioteca para estimá-lo.
4. Resolva mínimos quadrados com `lstsq`, QR, SVD ou método apropriado à estrutura dos dados.
5. Examine posto, valores singulares e condicionamento.
6. Calcule MSE ou RMSE e métricas complementares em dados não usados no ajuste.
7. Analise resíduos e verifique padrões, valores atípicos e mudança de distribuição.

> **!** A solução não é apenas uma fórmula
>
> Calcular explicitamente $(\widetilde{X}^{\mathsf{T}}\widetilde{X})^{-1}$ é desnecessário e pode ser instável. A pseudoinversa é um conceito matemático; em código, use um solucionador numérico confiável.

6.3.26.3 O que cada métrica responde

- **MSE:** qual é a perda quadrática média?
- **RMSE:** qual é a escala típica do erro com penalização quadrática, nas unidades do alvo?
- **MAE:** qual é o módulo médio do erro com influência linear?
- R^2**:** quanto o modelo melhora em relação ao preditor constante da média naquele conjunto?

Resultados de treino respondem sobre ajuste; resultados de validação e teste fornecem evidência de generalização. Nenhuma dessas métricas, por si só, estabelece causalidade ou segurança da aplicação.

6.3.26.4 Erros comuns

- esquecer a coluna de intercepto;
- transpor a matriz de projeto e trocar exemplos por atributos;
- usar a fórmula da inversa sem verificar posto;
- interpretar coeficientes sem considerar unidades e correlação entre atributos;
- calcular pré-processamento com dados de teste;
- extrapolar muito além da região observada;
- relatar somente R^2 e ocultar a escala dos resíduos.

6.3.26.5 Questões conceituais

1. Por que SSE e MSE possuem o mesmo minimizador?
2. Por que o resíduo ótimo é ortogonal às colunas da matriz de projeto?

3. Ter mais exemplos que parâmetros garante solução única? Justifique.
4. Por que formar $X^\mathsf{T}X$ pode piorar o condicionamento?
5. Em que sentido uma regressão com atributo x^2 ainda é linear?
6. Como um modelo pode apresentar R^2 negativo no conjunto de teste?

6.3.26.6 Exercícios

1. Para $\widetilde{X} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix}$ e $y = (1, 2, 2)^\mathsf{T}$, verifique por multiplicação as equações normais usando $\widetilde{w} = (7/6, 1/2)^\mathsf{T}$.
2. Construa uma matriz de projeto com duas colunas idênticas. Calcule seu posto e explique por que os coeficientes não são únicos.
3. Compare `np.linalg.lstsq` com a solução explícita das equações normais em uma matriz cujas colunas sejam quase colineares.
4. Ajuste uma reta a uma amostra, adicione um valor-alvo extremamente distante e observe como coeficientes, RMSE e MAE mudam.
5. Examine um gráfico de resíduos contra previsões. Que padrão sugeriria que uma relação não linear permanece sem modelagem?

6.3.26.7 Critério de domínio

Ao concluir esta seção, o estudante deve ser capaz de:

- montar $\widetilde{X}$, y, $\hat{y}$ e e com dimensões corretas;
- derivar as equações normais por projeção e pelo gradiente;
- explicar posto, pseudoinversa e unicidade;
- escolher uma rotina numérica apropriada sem calcular uma inversa explícita;
- avaliar o modelo fora da amostra e interpretar seus resíduos.

6.3.27 Animação

Uma animação pode mostrar simultaneamente duas visões do mesmo processo:

- no espaço dos dados, a reta muda à medida que b e w são atualizados;
- no espaço dos parâmetros, o ponto (b, w) percorre a superfície do MSE em direção ao mínimo.

Para uma regressão com um atributo, o MSE é uma função quadrática de dois parâmetros. Suas curvas de nível são elipses quando a solução é única.

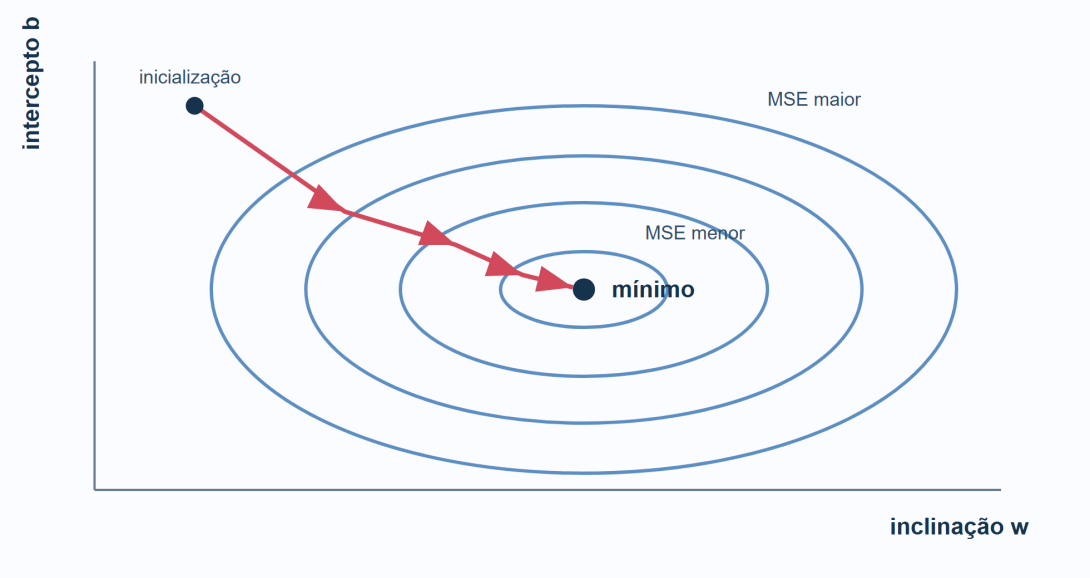


Figura 6.11: O gradiente descendente atravessa curvas de nível do MSE em direção ao mínimo.

Elipses muito alongadas indicam escalas diferentes ou forte correlação entre as direções dos parâmetros. Nesse cenário, uma taxa de aprendizado alta pode fazer a trajetória oscilar de um lado para outro do vale.

6.3.27.1 Registrando a trajetória

O código a seguir ajusta uma reta por gradiente descendente e guarda uma cópia dos parâmetros a cada passo.

```
import numpy as np


def trajetoria_gradiente(X, y, taxa=0.05, passos=100):
    X = np.asarray(X, dtype=float).reshape(-1)
    y = np.asarray(y, dtype=float)
    b, w = 0.0, 0.0
    historico = []

    for passo in range(passos + 1):
        predicoes = b + w * X
        residuos = predicoes - y
        mse = np.mean(residuos**2)
        historico.append((passo, b, w, mse))
```

```
        grad_b = 2 * np.mean(residuos)
        grad_w = 2 * np.mean(residuos * X)
        b -= taxa * grad_b
        w -= taxa * grad_w

    return historico
```

Usamos `residuos = predicoes - y`, por isso o gradiente aparece com sinal positivo antes da subtração. Se definíssemos resíduos como $y - \hat{y}$, o sinal da fórmula intermediária mudaria, mas a atualização final seria equivalente.

6.3.27.2 Animando a reta

```
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation


def animar_ajuste_linear(X, y, taxa=0.05, passos=100):
    X = np.asarray(X, dtype=float).reshape(-1)
    y = np.asarray(y, dtype=float)
    historico = trajetoria_gradiente(X, y, taxa, passos)

    fig, (ax_dados, ax_erro) = plt.subplots(1, 2, figsize=(10, 4))
    ax_dados.scatter(X, y, label="observações")
    eixo_x = np.linspace(X.min(), X.max(), 100)
    reta, = ax_dados.plot([], [], label="modelo")
    ax_dados.set_xlim(X.min(), X.max())
    margem_y = 0.1 * max(np.ptp(y), 1.0)
    ax_dados.set_ylim(y.min() - margem_y, y.max() + margem_y)
    ax_dados.legend()

    passos_hist = [estado[0] for estado in historico]
    erros_hist = [estado[3] for estado in historico]
    ax_erro.plot(passos_hist, erros_hist, color="lightgray")
    ponto_erro, = ax_erro.plot([], [], "o")
    ax_erro.set_xlabel("passo")
    ax_erro.set_ylabel("MSE")
    titulo = fig.suptitle("")
```

```
    def atualizar(estado):
        passo, b, w, mse = estado
        reta.set_data(eixo_x, b + w * eixo_x)
        ponto_erro.set_data([passo], [mse])
        titulo.set_text(
            f"passo={passo} · b={b:.3f} · w={w:.3f} · MSE={mse:.4f}"
        )
        return reta, ponto_erro, titulo

    animacao = FuncAnimation(
        fig,
        atualizar,
        frames=historico,
        interval=100,
        repeat=False,
    )
    return fig, animacao
```

Em um notebook:

```
from IPython.display import HTML

X = np.array([0.0, 1.0, 2.0, 3.0, 4.0])
y = np.array([1.1, 2.8, 5.2, 6.9, 9.2])

fig, animacao = animar_ajuste_linear(
    X, y, taxa=0.03, passos=100
)
plt.close(fig)
HTML(animacao.to_jshtml())
```

6.3.27.3 Experimentos guiados

1. Aumente a taxa de aprendizado gradualmente. Em que valor o MSE deixa de diminuir de modo estável?
2. Multiplique X por $1\,000$ sem alterar a taxa. Explique a mudança usando a expressão do gradiente.
3. Padronize X e repita o experimento. Compare a trajetória.

4. Adicione um valor atípico a y. Observe como a reta ótima muda.
5. Compare os parâmetros finais ao resultado de `np.linalg.lstsq`.

> **A solução direta não percorre a animação**
>
> QR ou SVD calcula a solução por operações de álgebra linear e não gera naturalmente uma sequência de retas intermediárias. A trajetória acima pertence ao gradiente descendente, um algoritmo alternativo para o mesmo objetivo quadrático.

6.4 Regressão Logística

A regressão logística é um modelo de **classificação probabilística binária**. Embora o nome contenha "regressão", o alvo observado pertence a duas classes. Usaremos a codificação

$$y \in \{0, 1\},$$

em que $y = 1$ representa o evento de interesse e $y = 0$, seu complemento. Exemplos incluem inadimplência ou pagamento, fraude ou transação legítima, falha ou funcionamento e presença ou ausência de uma condição clínica.

O objetivo não é apenas devolver uma classe. Queremos modelar

$$p(x) = P(Y = 1 \mid X = x),$$

a probabilidade condicional do evento positivo dados os atributos. Como as classes são complementares,

$$P(Y = 0 \mid X = x) = 1 - p(x).$$

6.4.1 Por que a regressão linear não basta?

Poderíamos codificar os rótulos como zero e um e ajustar $w^{\mathsf{T}} x + b$ por mínimos quadrados. Esse procedimento, chamado modelo de probabilidade linear, possui limitações importantes:

- a saída pode ser menor que zero ou maior que um;
- o efeito de uma variação na pontuação permanece constante, mesmo próximo aos extremos probabilísticos;
- a variância de uma variável binária depende de sua probabilidade;

- o MSE não corresponde diretamente à verossimilhança de uma observação Bernoulli.

A regressão logística mantém a pontuação linear

$$z = w^{\mathsf{T}} x + b,$$

mas aplica uma transformação que leva qualquer número real ao intervalo $(0, 1)$:

$$p(x) = \sigma(z) = \frac{1}{1 + e^{-z}}. \tag{6.30}$$

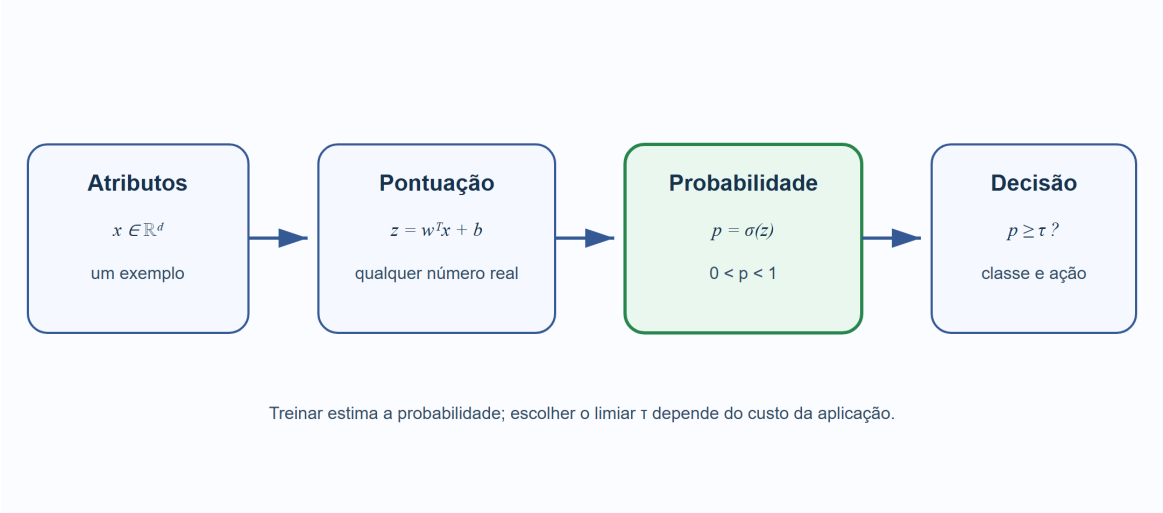


Figura 6.12: A regressão logística separa a estimativa de probabilidade da decisão tomada com um limiar.

6.4.2 Probabilidade não é decisão

Depois de estimar $p(x)$, uma aplicação pode convertê-la em classe usando um limiar τ:

$$\hat{y} = \begin{cases} 1, & p(x) \geq \tau, \\ 0, & p(x) < \tau. \end{cases}$$

O valor $\tau = 0{,}5$ é comum, mas não obrigatório. Se deixar de detectar uma fraude custa muito mais que investigar um alarme falso, um limiar menor pode ser adequado. O modelo estima probabilidades; a política de decisão combina essas probabilidades com custos e restrições.

Como $\sigma(0) = 0{,}5$ e a sigmoide é crescente, o limiar $0{,}5$ produz a fronteira

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

a mesma forma geométrica do Perceptron. A diferença está na saída e no treinamento: o Perceptron fornece um sinal e usa correções de erros; a regressão logística fornece um valor contínuo e otimiza uma perda probabilística.

6.4.3 Interpretação condicional

Não existe, para cada entrada x, um rótulo determinístico que o modelo precise descobrir. Duas observações com atributos iguais podem ter resultados diferentes. A incerteza é representada por uma distribuição Bernoulli:

$$Y \mid X = x \sim \text{Bernoulli}(p(x)).$$

Isso significa

$$P(Y = y \mid X = x) = p(x)^y [1 - p(x)]^{1-y}, \qquad y \in \{0, 1\}.$$

Ao observar muitos casos semelhantes com $p(x) = 0{,}8$, esperamos uma proporção próxima de 80% de resultados positivos, não a certeza de que cada caso será positivo.

6.4.4 Exemplo de leitura

Considere um modelo de falha de servidores:

$$z = -4 + 0{,}03\,\text{temperatura} + 1{,}2\,\text{carga}.$$

Para temperatura 80 e carga $0{,}75$,

$$z = -4 + 0{,}03(80) + 1{,}2(0{,}75) = -0{,}7,$$

e

$$p = \sigma(-0{,}7) \approx 0{,}332.$$

Esse número é uma estimativa condicional ao vetor de atributos e ao modelo ajustado. Não é uma frequência observada diretamente para aquele servidor individual.

6.4.5 Três níveis que não devem ser confundidos

Quantidade	Domínio	Significado
pontuação z	$\mathbb{R}$	evidência linear antes da transformação
probabilidade $p = \sigma(z)$	$(0, 1)$	estimativa de $P(Y = 1 \mid X = x)$
decisão $\hat{y}$	$\{0, 1\}$	ação após aplicar um limiar

Uma boa acurácia não garante probabilidades confiáveis. Se previsões de $0{,}8$ correspondem a resultados positivos em apenas 60% dos casos, o modelo está mal calibrado, ainda que muitas classes estejam corretas. Calibração e discriminação são propriedades diferentes.

> ⚠ **Probabilidade do modelo não é certeza científica**
>
> A saída depende dos dados, atributos, população e suposições do modelo. Mudança de distribuição, grupos pouco representados e variáveis omitidas podem tornar probabilidades inadequadas. Em aplicações de alto impacto, avalie calibração e erros por subgrupo e defina supervisão humana.

6.4.6 Roteiro da seção

Nas próximas subseções, estudaremos:

1. a função logística e sua relação com chances e *log-odds*;
2. a classe de hipóteses probabilísticas;
3. a perda de entropia cruzada derivada da Bernoulli;
4. gradiente, Hessiana e métodos de otimização;
5. implementação numericamente estável e avaliação.

6.4.7 Função Logística

Precisamos transformar uma pontuação $z \in \mathbb{R}$ em uma probabilidade. A **função logística**, também chamada sigmoide, é

$$\sigma(z) = \frac{1}{1 + e^{-z}} = \frac{e^{z}}{1 + e^{z}}. \tag{6.31}$$

Seu domínio é toda a reta real e sua imagem é o intervalo aberto $(0, 1)$. Ela nunca produz exatamente zero ou um para um argumento real finito, mas se aproxima desses valores quando z tende a $-\infty$ ou $+\infty$.

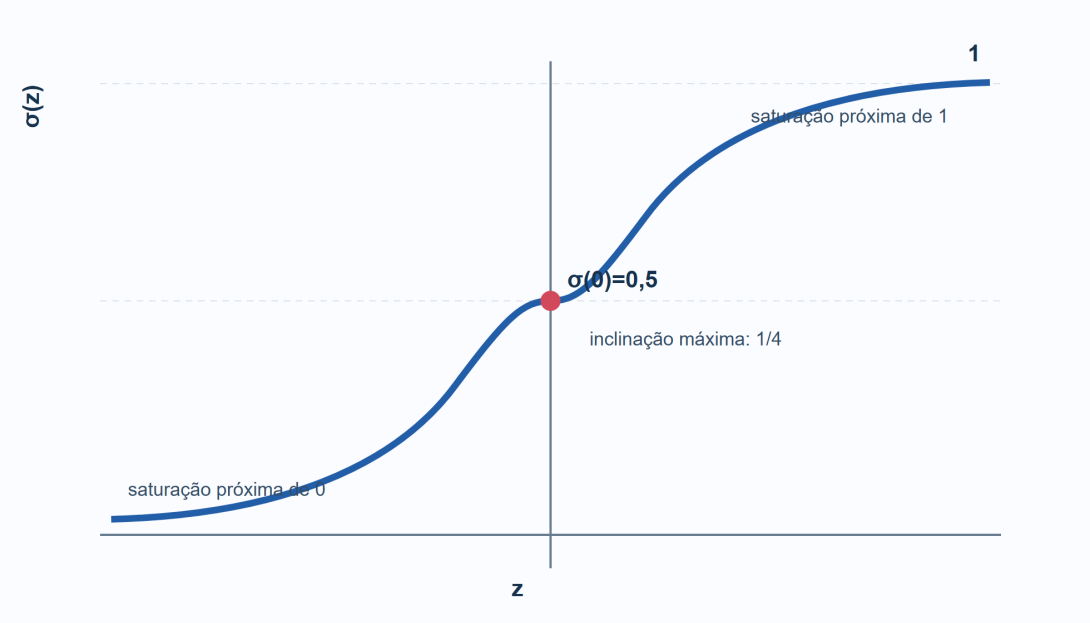


Figura 6.13: A sigmoide converte uma pontuação real em probabilidade e satura próximo aos extremos.

6.4.7.0.1 Propriedades fundamentais

A sigmoide satisfaz

$$\sigma(0) = \frac{1}{2}, \qquad \sigma(-z) = 1 - \sigma(z).$$

Ela é estritamente crescente. Sua derivada possui uma forma especialmente conveniente:

$$\begin{aligned} \sigma'(z) &= \frac{e^{-z}}{(1 + e^{-z})^{2}} \\ &= \sigma(z)[1 - \sigma(z)]. \end{aligned} \tag{6.32}$$

Como $0 < \sigma(z) < 1$, a derivada é positiva. Seu valor máximo é $1/4$, atingido em $z = 0$. Nas caudas, a derivada se aproxima de zero: dizemos que a função **satura**.

Para pequenas variações Δz próximas de zero,

$$\Delta p \approx \sigma'(0)\Delta z = \frac{1}{4}\Delta z.$$

Longe do centro, a mesma variação de pontuação causa uma alteração menor na probabilidade.

6.4.7.0.2 Chances e chances logarítmicas

Se um evento tem probabilidade p, suas **chances** (*odds*) são

$$\text{odds}(p) = \frac{p}{1-p}.$$

Probabilidade $p = 0{,}8$, por exemplo, corresponde a chances $0{,}8/0{,}2 = 4$: lemos "quatro para um" a favor do evento. Probabilidade $0{,}5$ corresponde a chances 1.

O logaritmo das chances é a função **logit**:

$$\text{logit}(p) = \ln\left(\frac{p}{1-p}\right). \tag{6.33}$$

Aplicando-a à sigmoide,

$$\frac{\sigma(z)}{1-\sigma(z)} = e^{z} \quad \Longrightarrow \quad \text{logit}(\sigma(z)) = z.$$

Portanto, sigmoide e logit são funções inversas:

$$p = \sigma(z) \quad \Longleftrightarrow \quad z = \ln\left(\frac{p}{1-p}\right).$$

Na regressão logística, a quantidade linear não é a probabilidade, mas o logaritmo das chances:

$$\ln\left(\frac{p(x)}{1-p(x)}\right) = w^{\mathsf{T}}x + b. \tag{6.34}$$

6.4.7.0.3 Interpretação de um coeficiente

Mantendo os outros atributos fixos, aumentar x_j em uma unidade soma w_j aos log-odds. Exponenciando, as chances são multiplicadas por e^{w_j}:

$$\frac{\text{odds}(x_j + 1)}{\text{odds}(x_j)} = e^{w_j}.$$

Se $w_j = \ln 2$, uma unidade adicional duplica as chances. Isso **não** significa dobrar a probabilidade. Se ela passa de $0{,}2$ para chances $0{,}25$, dobrar as chances produz $0{,}5$, que corresponde a probabilidade $1/3$, não $0{,}4$.

6.4.7.0.4 Relação com a distribuição Bernoulli

Uma variável Bernoulli com parâmetro p satisfaz

$$P(Y = y \mid p) = p^y(1-p)^{1-y}, \qquad y \in \{0, 1\}.$$

Defina o parâmetro natural

$$\eta = \ln\left(\frac{p}{1-p}\right).$$

Como $p = \sigma(\eta)$, podemos reescrever a massa de probabilidade:

$$P(Y = y \mid \eta) = \exp\left[y\eta - \ln(1 + e^{\eta})\right]. \tag{6.35}$$

Essa é a forma de família exponencial da Bernoulli. A regressão logística modela o parâmetro natural por $\eta = w^{\mathsf{T}}x + b$. Essa ligação explica por que a função logística e a perda de entropia cruzada aparecem juntas; elas não são apenas escolhas gráficas convenientes.

6.4.7.0.5 Função logística, probit e tangente hiperbólica

O modelo **probit** substitui a sigmoide pela função de distribuição acumulada normal $\Phi(z)$. Ambas produzem curvas em S e frequentemente geram ajustes semelhantes após uma mudança de escala, mas correspondem a funções de ligação diferentes.

A tangente hiperbólica satisfaz

$$\tanh(z) = 2\sigma(2z) - 1$$

e possui imagem $(-1, 1)$. Ela é comum como função de ativação em redes neurais, mas não representa diretamente uma probabilidade Bernoulli sem transformação adicional.

> **Logístico não é logarítmico**
>
> O nome “regressão logística” vem da função logística. O logaritmo aparece porque o modelo torna os **log-odds** lineares nos atributos.

6.4.7.0.6 Intercepto e probabilidade de referência

No vetor homogêneo,

$$\tilde{x} = (1, x_1, \dots, x_d), \qquad \widetilde{w} = (b, w_1, \dots, w_d),$$

temos $z = \widetilde{w}^{\mathsf{T}} \tilde{x}$. Quando todos os atributos originais valem zero,

$$p_0 = \sigma(b),$$

de modo que b representa os log-odds de referência. Centralizar os atributos pode tornar essa referência mais interpretável.

6.4.7.0.7 Computação numericamente estável

A fórmula direta pode causar transbordamento ao calcular e^{-z} para z muito negativo. Uma implementação por ramos evita exponenciais gigantes:

```
import numpy as np


def sigmoide_estavel(z):
    z = np.asarray(z, dtype=float)
    saida = np.empty_like(z)
    positivos = z >= 0

    saida[positivos] = 1 / (1 + np.exp(-z[positivos]))
    exp_z = np.exp(z[~positivos])
    saida[~positivos] = exp_z / (1 + exp_z)
    return saida
```

Para perdas logísticas, bibliotecas usam identidades como `logaddexp` e `logsumexp`, evitando calcular primeiro probabilidades arredondadas a zero ou um.

6.4.7.1 Exercícios

1. Demonstre a identidade Equação 6.32.
2. Converta probabilidades $0{,}1$, $0{,}5$ e $0{,}9$ em chances e log-odds.
3. Se $w_j = 0{,}4$, por qual fator as chances mudam quando x_j aumenta três unidades?
4. Mostre algebricamente que a inversa da função logit é a sigmoide.
5. Explique por que uma alteração fixa nos log-odds não corresponde a uma alteração fixa na probabilidade.

6.4.8 Hipóteses

A classe de hipóteses da regressão logística é

$$\mathcal{H}_{\text{log}} = \left\{ h_{w,b}(x) = \sigma(w^{\mathsf{T}} x + b) : w \in \mathbb{R}^d,\ b \in \mathbb{R} \right\}. \tag{6.36}$$

Cada hipótese associa a uma entrada uma probabilidade estimada:

$$h_{w,b}(x) = \widehat{P}(Y = 1 \mid X = x).$$

Como Y é binária,

$$\widehat{P}(Y = 0 \mid X = x) = 1 - h_{w,b}(x).$$

Essa função é uma **probabilidade condicional**, não uma densidade de variável contínua. Para $Y \in \{0, 1\}$, usamos uma função massa de probabilidade Bernoulli:

$$\widehat{P}(Y = y \mid X = x) = h(x)^y [1 - h(x)]^{1-y}. \tag{6.37}$$

6.4.9 Representação homogênea e matricial

Com

$$\widetilde{x} = (1, x_1, \dots, x_d), \qquad \widetilde{w} = (b, w_1, \dots, w_d),$$

escrevemos

$$h_{\widetilde{w}}(x) = \sigma(\widetilde{w}^{\mathsf{T}} \widetilde{x}).$$

Para uma amostra inteira,

$$z = \widetilde{X}\widetilde{w}, \qquad \hat{p} = \sigma(z), \tag{6.38}$$

em que a sigmoide é aplicada componente a componente. Assim, $z, \hat{p} \in \mathbb{R}^N$: há uma pontuação e uma probabilidade para cada exemplo.

6.4.10 Codificação dos rótulos

A codificação $\{0, 1\}$ é conveniente para a Bernoulli e a entropia cruzada. Se os dados usam $\{-1, +1\}$, podemos converter por

$$y_{01} = \frac{y_{\pm 1} + 1}{2}, \qquad y_{\pm 1} = 2y_{01} - 1.$$

As duas convenções são válidas, mas suas fórmulas de perda diferem. Misturá-las silenciosamente é uma fonte comum de erros de sinal.

6.4.11 Superfícies de probabilidade

Todos os pontos com a mesma probabilidade $c \in (0, 1)$ satisfazem

$$\sigma(w^{\mathsf{T}} x + b) = c.$$

Aplicando a função logit,

$$w^{\mathsf{T}} x + b = \ln\left(\frac{c}{1-c}\right).$$

Portanto, as superfícies de probabilidade constante são hiperplanos paralelos. Para $c = 0{,}5$, o lado direito é zero e obtemos a fronteira

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

Para um limiar geral τ, a fronteira de decisão é

$$w^{\mathsf{T}} x + b = \operatorname{logit}(\tau). \tag{6.39}$$

Mudar o limiar desloca a fronteira paralelamente; não é necessário treinar novamente o modelo apenas para testar outra política de decisão.

6.4.12 Interpretação dos parâmetros

O modelo supõe linearidade nos log-odds:

$$\text{logit}(h(x)) = b + w_1 x_1 + \cdots + w_d x_d.$$

Cada w_j representa a mudança nos log-odds para uma unidade adicional de x_j, mantendo os outros atributos constantes. O fator e^{w_j} é a razão de chances correspondente.

Essa interpretação depende da especificação do modelo. Se uma relação é curva, omitir termos não lineares pode tornar o coeficiente enganoso. Se dois atributos são fortemente correlacionados, a frase "mantendo os outros constantes" pode descrever uma comparação rara ou impossível nos dados.

6.4.13 Engenharia de atributos

Assim como na regressão linear, podemos usar um mapa de características:

$$h_w(x) = \sigma(w^\mathsf{T} \phi(x)) \,.$$

Com $\phi(x) = (1, x_1, x_2, x_1 x_2, x_1^2)$, a fronteira no espaço original pode ser curva, embora os log-odds continuem lineares nos parâmetros. Isso aumenta a capacidade e deve ser acompanhado por validação e, muitas vezes, regularização.

6.4.14 O que a hipótese assume — e o que não assume

A regressão logística não exige que os atributos sigam distribuição normal. A especificação central é que os log-odds condicionais sejam modelados pela combinação escolhida de características. Para inferência estatística tradicional, outras condições podem ser necessárias; para predição, avaliamos sobretudo desempenho, calibração e estabilidade fora da amostra.

> ⚠ Separação perfeita
>
> Se uma combinação linear separa perfeitamente as classes, a verossimilhança sem regularização pode continuar melhorando enquanto a norma dos coeficientes tende ao infinito. As probabilidades ficam extremas e não há estimativa finita usual de máxima verossimilhança.

> Regularização ou métodos específicos tratam esse caso.

6.4.15 Exemplo vetorizado

```
import numpy as np

X = np.array([
    [70.0, 0.40],
    [85.0, 0.75],
    [60.0, 0.25],
])
w = np.array([0.03, 1.20])
b = -4.0

z = X @ w + b
p = 1 / (1 + np.exp(-z))
classes = (p >= 0.5).astype(int)

print("pontuações:", z)
print("probabilidades:", p)
print("classes:", classes)
```

O vetor p é a saída probabilística da hipótese; `classes` depende do limiar escolhido e pertence à camada de decisão.

6.4.15.1 Questões

1. Derive Equação 6.39 para $\tau = 0{,}8$.
2. Converta uma base rotulada em $\{-1, +1\}$ para $\{0, 1\}$.
3. Explique por que todas as superfícies de probabilidade constante são paralelas no modelo sem transformações não lineares.
4. Dê um mapa $\phi(x)$ que permita uma fronteira circular em duas dimensões.

6.4.16 Erro

A amostra não contém as probabilidades verdadeiras; contém realizações binárias. Para relacionar essas observações aos parâmetros, usamos o modelo Bernoulli e o princípio da **máxima verossimilhança**.

Se

$$p_n = \sigma(\widetilde{x}_n^{\mathsf{T}} \widetilde{w}),$$

a probabilidade atribuída ao rótulo $y_n \in \{0, 1\}$ é

$$P(Y_n = y_n \mid x_n, \widetilde{w}) = p_n^{y_n} (1 - p_n)^{1 - y_n}.$$

Assumindo que as observações são condicionalmente independentes dados os atributos, a verossimilhança da amostra é

$$L(\widetilde{w}) = \prod_{n=1}^{N} p_n^{y_n} (1 - p_n)^{1 - y_n}. \tag{6.40}$$

Essa expressão é uma função dos parâmetros para os dados fixados. Ela não é a probabilidade de que os parâmetros sejam verdadeiros.

6.4.16.0.1 Do produto à soma

Produtos de muitas probabilidades pequenas podem sofrer subfluxo numérico. Como o logaritmo é crescente, maximizar L equivale a maximizar a log-verossimilhança:

$$\ell(\widetilde{w}) = \ln L(\widetilde{w}) = \sum_{n=1}^{N} \left[y_n \ln p_n + (1 - y_n) \ln(1 - p_n) \right].$$

Algoritmos de aprendizado costumam minimizar uma perda. Tomamos a média negativa:

$$E_{\text{in}}(\widetilde{w}) = -\frac{1}{N} \sum_{n=1}^{N} \left[y_n \ln p_n + (1 - y_n) \ln(1 - p_n) \right]. \tag{6.41}$$

Essa função é chamada **entropia cruzada binária**, **log-loss** ou perda logística.

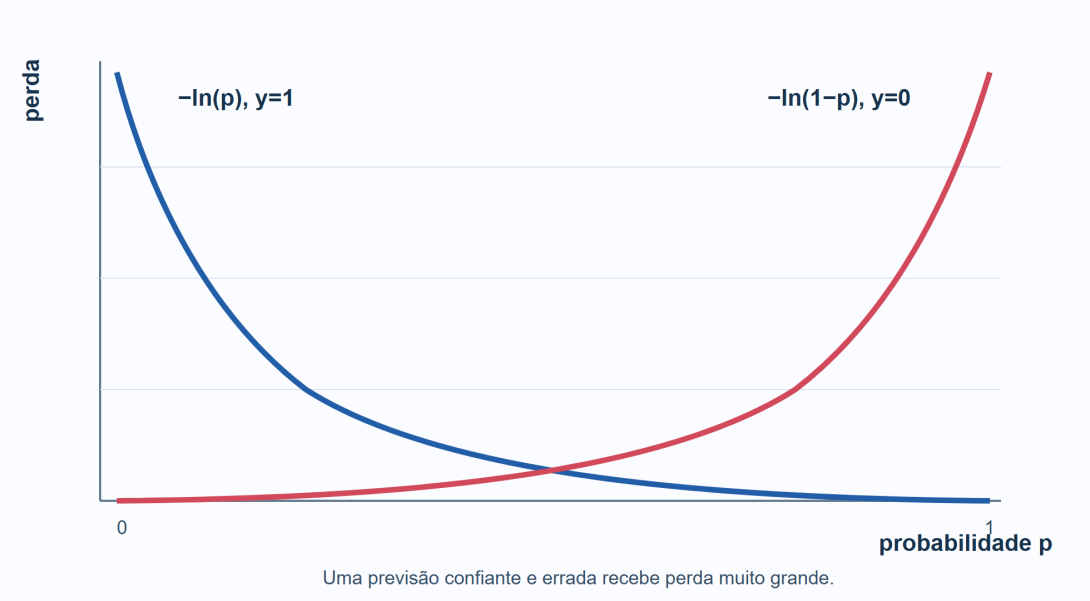


Figura 6.14: A entropia cruzada penaliza fortemente probabilidades confiantes atribuídas à classe errada.

Para $y = 1$, a perda individual é $-\ln p$; para $y = 0$, é $-\ln(1 - p)$. Uma previsão correta e confiante recebe perda próxima de zero. Uma previsão errada e confiante recebe perda muito grande.

6.4.16.0.2 Forma com rótulos $\{-1, +1\}$

Se usarmos $t_n \in \{-1, +1\}$ e a pontuação $z_n = \tilde{x}_n^{\mathsf{T}} \widetilde{w}$, a probabilidade do rótulo observado é

$$P(T_n = t_n \mid x_n) = \sigma(t_n z_n).$$

Logo, a perda individual é

$$-\ln \sigma(t_n z_n) = \ln\left(1 + e^{-t_n z_n}\right).$$

O erro médio torna-se

$$E_{\text{in}}(\widetilde{w}) = \frac{1}{N} \sum_{n=1}^{N} \ln\left(1 + e^{-t_n \tilde{x}_n^{\mathsf{T}} \widetilde{w}}\right). \tag{6.42}$$

As formas Equação 6.41 e Equação 6.42 são equivalentes após converter os rótulos. A expressão $\ln(1 + e^a)$ é chamada **softplus**.

6.4.16.0.3 Relação com a margem

Na codificação $\{-1, +1\}$, a margem é

$$m_n = t_n z_n.$$

A perda $\ln(1 + e^{-m_n})$ é decrescente na margem:

- margem grande e positiva: classificação correta e perda próxima de zero;
- margem zero: probabilidade $0{,}5$ e perda $\ln 2$;
- margem negativa: classificação errada, com perda que cresce à medida que o erro se torna mais confiante.

Ao contrário da perda zero–um, a perda logística é suave e fornece um gradiente útil em quase todas as situações.

6.4.16.0.4 Por que não usar somente acurácia?

Acurácia considera apenas o lado do limiar. Duas previsões de $0{,}51$ e $0{,}99$ para um caso positivo são ambas corretas, mas a segunda atribui mais probabilidade ao resultado observado. A log-loss distingue essas situações e é uma **regra de pontuação própria**: em expectativa, é minimizada ao relatar a probabilidade verdadeira.

Isso não elimina a necessidade de métricas de decisão. Log-loss avalia qualidade probabilística; precisão, revocação e custos avaliam o que acontece depois de aplicar um limiar.

6.4.16.0.5 Computação estável

Calcular primeiro $p = \sigma(z)$ e depois `log(p)` pode produzir `log(0)` por arredondamento. Com rótulos $\{-1, +1\}$, use `np.logaddexp(0, -t * z)`, que calcula a softplus de modo estável:

```
import numpy as np


def perda_logistica_pm1(X, t, w, b):
    z = X @ w + b
    perdas = np.logaddexp(0.0, -t * z)
    return np.mean(perdas)
```

Com rótulos $\{0, 1\}$, uma forma estável diretamente nos *logits* é

$$\ell(z, y) = \ln(1 + e^z) - yz.$$

```
def entropia_cruzada_logits(X, y, w, b):
    z = X @ w + b
    perdas = np.logaddexp(0.0, z) - y * z
    return np.mean(perdas)
```

6.4.16.0.6 Regularização

É comum adicionar uma penalidade aos pesos:

$$E_\lambda(\widetilde{w}) = E_{\text{in}}(\widetilde{w}) + \frac{\lambda}{2}\|w\|_2^2.$$

O intercepto normalmente não é penalizado. A regularização reduz coeficientes extremos, ajuda em separação perfeita e controla sobreajuste. O valor λ deve ser escolhido por validação, não pelo conjunto de teste.

> ⚠ Log-loss não possui limite superior
>
> Uma probabilidade quase zero para um evento que ocorre produz perda muito grande. Isso é intencional: o modelo é responsabilizado por confiança indevida. Também torna essencial investigar rótulos incorretos e usar operações numericamente estáveis.

6.4.16.1 Exemplo

Para um caso positivo com $p = 0{,}8$,

$$-\ln(0{,}8) \approx 0{,}223.$$

Se o modelo atribui $p = 0{,}01$ ao mesmo resultado,

$$-\ln(0{,}01) \approx 4{,}605.$$

A segunda previsão não é apenas incorreta segundo um limiar de $0{,}5$; é incorreta com alta confiança.

6.4.16.2 Exercícios

1. Derive Equação 6.41 a partir de
   1.
2. Mostre que $-\ln \sigma(a) = \ln(1 + e^{-a})$.
3. Calcule a log-loss para $y = 0$ e previsões $p = 0{,}1$ e $p = 0{,}9$.
4. Explique por que maximizar a verossimilhança equivale a minimizar a log-verossimilhança negativa.
5. Compare a perda logística e a perda zero–um em margens $-2, 0, 2$.

6.4.17 Algoritmo

Treinar a regressão logística significa encontrar parâmetros que minimizem a entropia cruzada, ou, equivalentemente, maximizem a verossimilhança Bernoulli. Para rótulos $y \in \{0, 1\}$, o problema é

$$\widetilde{w}^* \in \arg\min_{\widetilde{w}} \frac{1}{N} \sum_{n=1}^{N} \left[\ln(1 + e^{z_n}) - y_n z_n\right], \qquad z = \widetilde{X}\widetilde{w}. \tag{6.43}$$

Na regressão linear, zerar o gradiente produziu um sistema linear com solução fechada. Aqui, as probabilidades $p_n = \sigma(\tilde{x}_n^{\mathsf{T}} \widetilde{w})$ dependem não linearmente dos parâmetros. A condição de primeira ordem forma um sistema não linear e, em geral, precisa ser resolvida iterativamente.

6.4.17.1 Gradiente e Hessiana em uma visão antecipada

Reunindo as probabilidades no vetor p, o gradiente é

$$\nabla E(\widetilde{w}) = \frac{1}{N} \widetilde{X}^{\mathsf{T}} (p - y). \tag{6.44}$$

O gradiente aponta na direção de maior crescimento local. Para diminuir a perda, movemo-nos na direção oposta, $-\nabla E$.

A Hessiana é

$$\nabla^2 E(\widetilde{w}) = \frac{1}{N} \widetilde{X}^{\mathsf{T}} S \widetilde{X}, \tag{6.45}$$

em que

$$S = \operatorname{diag}\left(p_1(1 - p_1), \ldots, p_N(1 - p_N)\right).$$

Para qualquer vetor v,

$$v^\top \nabla^2 E(\widetilde{w}) v = \frac{1}{N}(\widetilde{X}v)^\top S(\widetilde{X}v) \geq 0,$$

pois os elementos diagonais de S são não negativos. Logo, a perda é convexa: não há mínimos locais ruins. Isso não significa que toda execução seja trivial; condicionamento, escolha do passo e separação perfeita ainda importam.

6.4.17.2 Três estratégias

As subseções seguintes apresentam métodos relacionados:

Método	Informação usada	Custo por iteração	Característica
gradiente descendente	primeira derivada	geralmente $O(Nd)$	simples e escalável
Newton/IRLS	gradiente e Hessiana	inclui sistema em $d+1$ variáveis	poucas iterações perto do mínimo
máximo declive com busca de passo	gradiente e escolha adaptativa de η	várias avaliações da perda	reduz risco de passo inadequado

Em bases muito grandes, o gradiente pode ser estimado com minilotes. Em dimensão moderada, Newton ou métodos quase-Newton, como BFGS e L-BFGS, costumam convergir em menos iterações.

6.4.17.3 Regularização

Com penalidade L_2, minimizamos

$$E_\lambda(\widetilde{w}) = E(\widetilde{w}) + \frac{\lambda}{2}\|w\|_2^2,$$

normalmente excluindo o intercepto b. O gradiente ganha o termo λw e a Hessiana ganha λI nas coordenadas penalizadas. Além de controlar sobreajuste, esse termo melhora o condicionamento e produz solução finita em muitos casos de separação perfeita.

6.4.17.4 Estrutura de uma iteração

Independentemente do método, um ciclo típico contém:

1. calcular os *logits* $z = \widetilde{X}\widetilde{w}$;

2. calcular probabilidades ou quantidades equivalentes de modo estável;
3. avaliar perda e gradiente;
4. determinar uma direção d_t;
5. escolher um tamanho de passo η_t;
6. atualizar $\widetilde{w}^{(t+1)} = \widetilde{w}^{(t)} + \eta_t d_t$;
7. verificar critérios de parada.

Para gradiente descendente, $d_t = -\nabla E$. Para Newton,

$$d_t = -[\nabla^2 E(\widetilde{w}^{(t)})]^{-1} \nabla E(\widetilde{w}^{(t)}),$$

mas uma implementação resolve o sistema linear $\nabla^2 E\, d_t = -\nabla E$ sem calcular a inversa.

6.4.17.5 Critérios de parada

Um único critério pode ser enganoso. Implementações robustas combinam:

- norma do gradiente abaixo de uma tolerância;
- variação relativa da perda pequena;
- variação relativa dos parâmetros pequena;
- número máximo de iterações;
- detecção de valores não finitos;
- desempenho de validação para parada antecipada, quando apropriado.

A perda deve ser monitorada em precisão suficiente. Uma mudança absoluta de 10^{-6} pode ser irrelevante para uma perda de 10^{6} e importante para uma perda de 10^{-8}; por isso, tolerâncias relativas são úteis.

6.4.17.6 Inicialização e escala

Inicializar em zero é válido para uma regressão logística isolada: não há simetria entre várias unidades que precise ser quebrada. Se as classes são muito desbalanceadas, uma inicialização útil do intercepto é

$$b^{(0)} = \ln\left(\frac{\overline{y}}{1-\overline{y}}\right),$$

com pesos inicialmente zero, desde que $0 < \overline{y} < 1$. Isso começa com a prevalência da classe positiva.

Padronizar atributos melhora o condicionamento e torna uma única taxa de aprendizado mais

razoável para todas as coordenadas. As estatísticas de padronização devem ser estimadas somente no treino.

> ⚠ Convergência numérica não garante um bom modelo
>
> Um otimizador pode minimizar perfeitamente a perda de treinamento e ainda generalizar mal, estar descalibrado ou apresentar desempenho desigual entre grupos. Otimização resolve os parâmetros do objetivo especificado; avaliação verifica se esse objetivo serve à aplicação.

6.4.17.7 Diagnósticos básicos

Durante o treinamento, registre pelo menos:

- perda de treino e validação;
- norma do gradiente;
- norma dos parâmetros;
- tamanho do passo;
- número de iterações e motivo da parada.

Crescimento contínuo da norma dos parâmetros com perda decrescente pode indicar separação perfeita sem regularização. Perda `NaN` ou infinita costuma indicar operações instáveis ou passo excessivo.

6.4.17.8 Gradiente

Para uma função escalar $f : \mathbb{R}^p \to \mathbb{R}$, o gradiente reúne as derivadas parciais:

$$\nabla f(w) = \begin{bmatrix} \partial f/\partial w_1 \\ \vdots \\ \partial f/\partial w_p \end{bmatrix}.$$

Adotaremos gradientes como vetores coluna. A derivada direcional na direção unitária v é

$$D_v f(w) = \nabla f(w)^\mathsf{T} v.$$

Por Cauchy–Schwarz, esse valor é máximo quando v aponta na direção do gradiente. Portanto, $-\nabla f(w)$ é a direção de maior declive local.

6.4.17.8.1 Derivação com rótulos $\{0, 1\}$

Para um exemplo, defina

$$z_n = \tilde{x}_n^{\mathsf{T}} \widetilde{w}, \qquad p_n = \sigma(z_n),$$

e use a perda estável nos logits

$$\ell_n(\widetilde{w}) = \ln(1 + e^{z_n}) - y_n z_n.$$

Como a derivada da softplus é a sigmoide,

$$\frac{\partial \ell_n}{\partial z_n} = \sigma(z_n) - y_n = p_n - y_n.$$

Além disso,

$$\nabla_{\widetilde{w}} z_n = \tilde{x}_n.$$

Pela regra da cadeia,

$$\nabla_{\widetilde{w}} \ell_n = (p_n - y_n)\tilde{x}_n. \tag{6.46}$$

Calculando a média e reunindo os exemplos na matriz de projeto,

$$\nabla E(\widetilde{w}) = \frac{1}{N} \sum_{n=1}^{N} (p_n - y_n)\tilde{x}_n = \frac{1}{N} \widetilde{X}^{\mathsf{T}}(p - y). \tag{6.47}$$

A fórmula possui uma interpretação simples: $p - y$ mede o erro probabilístico de cada exemplo, e a multiplicação por $\widetilde{X}^{\mathsf{T}}$ distribui esses erros entre os parâmetros de acordo com os valores dos atributos.

6.4.17.8.2 Derivação com rótulos $\{-1, +1\}$

Para $t_n \in \{-1, +1\}$, a perda é

$$\ell_n = \ln(1 + e^{-t_n z_n}).$$

Sua derivada é

$$\nabla_{\widetilde{w}} \ell_n = -t_n \sigma(-t_n z_n) \tilde{x}_n = -\frac{t_n \tilde{x}_n}{1 + e^{t_n z_n}}. \tag{6.48}$$

Observe o expoente $+t_n z_n$ no denominador. Quando a margem é grande e positiva, a contribuição tende a zero; quando é negativa, a contribuição aproxima-se de $-t_n \tilde{x}_n$.

6.4.17.8.3 Atualização em lote

O gradiente descendente em lote usa todos os exemplos:

$$\widetilde{w}^{(k+1)} = \widetilde{w}^{(k)} - \eta_k \frac{1}{N} \widetilde{X}^{\mathsf{T}} \left(p^{(k)} - y \right). \tag{6.49}$$

Com regularização L_2, adicionamos λw às coordenadas dos pesos, deixando o intercepto sem penalidade.

Um passo muito pequeno converge lentamente. Um passo muito grande pode fazer a perda oscilar ou aumentar, apesar da convexidade. Convexidade descreve a forma do objetivo, não corrige automaticamente uma taxa de aprendizado inadequada.

6.4.17.8.4 Gradiente estocástico e minilotes

O gradiente completo custa $O(Nd)$ por iteração. Para bases grandes, podemos estimá-lo usando um subconjunto B:

$$g_B(\widetilde{w}) = \frac{1}{|B|} \sum_{n \in B} (p_n - y_n) \tilde{x}_n.$$

- **SGD:** $|B| = 1$; atualizações baratas e ruidosas;
- **minilote:** $1 < |B| < N$; aproveita vetorização e reduz variância;
- **lote completo:** $|B| = N$; direção determinística.

Em treinamento por minilotes, embaralhamos os exemplos a cada época e normalmente reduzimos a taxa de aprendizado ao longo do tempo. O ruído faz a perda oscilar, portanto um único aumento entre passos não implica falha.

6.4.17.8.5 Implementação estável

```
import numpy as np



def perda_e_gradiente_logistico(X, y, w, b, lambda_l2=0.0):
```

```
    z = X @ w + b

    # Perda diretamente nos logits: softplus(z) - y*z.
    perdas = np.logaddexp(0.0, z) - y * z
    perda = np.mean(perdas) + 0.5 * lambda_l2 * (w @ w)

    # Sigmoide estável.
    p = np.empty_like(z)
    positivos = z >= 0
    p[positivos] = 1 / (1 + np.exp(-z[positivos]))
    exp_z = np.exp(z[~positivos])
    p[~positivos] = exp_z / (1 + exp_z)

    erro = p - y
    grad_w = X.T @ erro / len(X) + lambda_l2 * w
    grad_b = np.mean(erro)
    return perda, grad_w, grad_b
```

Uma implementação do treinamento pode usar:

```
def ajustar_por_gradiente(
    X,
    y,
    *,
    taxa=0.1,
    max_iter=10_000,
    tolerancia=1e-7,
    lambda_l2=0.0,
):
    X = np.asarray(X, dtype=float)
    y = np.asarray(y, dtype=float)
    w = np.zeros(X.shape[1])
    b = 0.0
    historico = []

    for iteracao in range(1, max_iter + 1):
        perda, grad_w, grad_b = perda_e_gradiente_logistico(
            X, y, w, b, lambda_l2
```

```
        )
        norma_grad = np.sqrt(grad_w @ grad_w + grad_b**2)
        historico.append(perda)

        if not np.isfinite(perda):
            raise FloatingPointError("a perda deixou de ser finita")
        if norma_grad <= tolerancia:
            return w, b, historico, True

        w -= taxa * grad_w
        b -= taxa * grad_b

    return w, b, historico, False
```

O retorno booleano impede que atingir o limite de iterações seja confundido com convergência.

6.4.17.8.6 Verificação por diferenças finitas

Antes de confiar numa derivada implementada manualmente, podemos compará-la com uma aproximação numérica:

$$\frac{\partial E}{\partial w_j} \approx \frac{E(w + \varepsilon e_j) - E(w - \varepsilon e_j)}{2\varepsilon}.$$

```
def gradiente_numerico(funcao, w, epsilon=1e-6):
    aproximacao = np.zeros_like(w)

    for j in range(len(w)):
        deslocamento = np.zeros_like(w)
        deslocamento[j] = epsilon
        aproximacao[j] = (
            funcao(w + deslocamento)
            - funcao(w - deslocamento)
        ) / (2 * epsilon)

    return aproximacao
```

Diferenças finitas também têm erro de arredondamento e truncamento, mas são excelentes para detectar sinais trocados, fatores ausentes e erros de transposição em exemplos pequenos.

6.4.17.8.7 Convergência e unicidade

A perda é convexa, mas o mínimo não é necessariamente único sem condições adicionais. Colunas redundantes produzem direções planas. Em dados perfeitamente separáveis e sem regularização, o ínfimo pode ser aproximado enquanto a norma dos parâmetros diverge, sem um minimizador finito.

Com regularização L_2 positiva nas direções relevantes, o objetivo torna-se mais fortemente convexo e o comportamento numérico melhora.

> **Monitore mais que a perda**
>
> Registre norma do gradiente, perda de validação e norma dos parâmetros. Uma perda que muda pouco pode indicar convergência, taxa pequena demais ou saturação numérica; os outros sinais ajudam a distinguir os casos.

6.4.17.8.8 Exercícios

1. Derive Equação 6.46 pela regra da cadeia.
2. Converta Equação 6.48 para a codificação $\{0, 1\}$.
3. Implemente a verificação por diferenças finitas para w e b.
4. Compare lote completo e minilotes na mesma amostra, usando a mesma quantidade de épocas.
5. Explique por que padronizar atributos permite usar uma taxa comum com mais segurança.

6.4.17.9 Método de Newton

O gradiente usa a inclinação local, mas ignora como essa inclinação muda em cada direção. O método de Newton acrescenta a informação de curvatura contida na Hessiana.

6.4.17.9.1 Da aproximação quadrática ao passo

Perto de um ponto w, a expansão de Taylor de segunda ordem é

$$E(w + d) \approx E(w) + \nabla E(w)^{\mathsf{T}} d + \frac{1}{2} d^{\mathsf{T}} H(w) d,$$

em que $H(w) = \nabla^2 E(w)$. Para minimizar essa aproximação quadrática em relação a d, zeramos seu gradiente:

$$\nabla E(w) + H(w) d = 0.$$

A direção de Newton resolve, portanto,

$$H(w)d = -\nabla E(w), \tag{6.50}$$

e a atualização é

$$w_{\text{novo}} = w + \eta d, \tag{6.51}$$

com $\eta = 1$ no Newton puro. Em código, resolvemos Equação 6.50; não calculamos H^{-1} explicitamente.

Em uma dimensão, essa expressão reduz a

$$w_{\text{novo}} = w - \frac{E'(w)}{E''(w)},$$

que é o método de Newton aplicado à equação $E'(w) = 0$.

6.4.17.9.2 Gradiente e Hessiana logísticos

Para a matriz homogênea $\widetilde{X}$, temos

$$g = \nabla E(\widetilde{w}) = \frac{1}{N}\widetilde{X}^{\mathsf{T}}(p - y)$$

e

$$H = \nabla^2 E(\widetilde{w}) = \frac{1}{N}\widetilde{X}^{\mathsf{T}} S \widetilde{X},$$

onde

$$S = \operatorname{diag}\left(p_1(1-p_1), \dots, p_N(1-p_N)\right).$$

A matriz S atribui peso maior aos exemplos cuja probabilidade está próxima de $0{,}5$ e peso menor aos que estão saturados perto de zero ou um. Se houver regularização L_2, adicionamos λI às coordenadas penalizadas da Hessiana e λw ao gradiente.

6.4.17.9.3 Relação com mínimos quadrados reponderados

A equação de Newton pode ser reescrita como um problema de **mínimos quadrados iterativamente reponderados** (IRLS). Defina a resposta de trabalho

$$r = \widetilde{X}\widetilde{w} + S^{-1}(y - p).$$

O próximo vetor resolve

$$\widetilde{w}_{\text{novo}} = \arg\min_{v} \left\| S^{1/2}(r - \widetilde{X}v) \right\|_2^2 . \tag{6.52}$$

As equações normais de 2 são

$$\widetilde{X}^{\mathsf{T}} S \widetilde{X} \widetilde{w}_{\text{novo}} = \widetilde{X}^{\mathsf{T}} S r,$$

que equivalem ao passo de Newton. O nome “reponderados” vem do fato de S mudar a cada iteração conforme mudam as probabilidades.

Na prática, não é necessário formar S como uma matriz $N \times N$. Se $s = p(1 - p)$, multiplicar por S equivale a multiplicar cada linha ou componente pelo elemento correspondente de s.

6.4.17.9.4 Implementação com Newton amortecido

O código abaixo reúne intercepto e pesos em um vetor homogêneo, inclui regularização apenas nos pesos e usa *backtracking* para evitar passos que aumentem a perda.

```
import numpy as np


def perda_logistica_logits(X, y, w, lambda_l2=0.0):
    z = X @ w
    perda_dados = np.mean(np.logaddexp(0.0, z) - y * z)
    return perda_dados + 0.5 * lambda_l2 * (w[1:] @ w[1:])


def ajustar_logistica_newton(
    X,
    y,
    *,
    lambda_l2=1e-6,
```

```
    max_iter=100,
    tolerancia=1e-8,
):
    X = np.asarray(X, dtype=float)
    y = np.asarray(y, dtype=float)
    Xh = np.column_stack([np.ones(len(X)), X])
    w = np.zeros(Xh.shape[1])
    penalidade = np.diag([0.0] + [lambda_l2] * X.shape[1])
    historico = []

    for iteracao in range(1, max_iter + 1):
        z = Xh @ w

        p = np.empty_like(z)
        positivos = z >= 0
        p[positivos] = 1 / (1 + np.exp(-z[positivos]))
        exp_z = np.exp(z[~positivos])
        p[~positivos] = exp_z / (1 + exp_z)

        g = Xh.T @ (p - y) / len(Xh) + penalidade @ w
        s = p * (1 - p)
        H = Xh.T @ (s[:, None] * Xh) / len(Xh) + penalidade

        norma_grad = np.linalg.norm(g)
        perda_atual = perda_logistica_logits(
            Xh, y, w, lambda_l2
        )
        historico.append(perda_atual)

        if norma_grad <= tolerancia:
            return w, historico, True

        try:
            direcao = np.linalg.solve(H, -g)
        except np.linalg.LinAlgError:
            direcao = np.linalg.lstsq(H, -g, rcond=None)[0]
```

```
        # Backtracking: aceite um passo que reduza a perda.
        passo = 1.0
        derivada_direcional = g @ direcao
        while passo > 1e-12:
            candidato = w + passo * direcao
            perda_nova = perda_logistica_logits(
                Xh, y, candidato, lambda_l2
            )
            if perda_nova <= (
                perda_atual + 1e-4 * passo * derivada_direcional
            ):
                break
            passo *= 0.5
        else:
            return w, historico, False

        w = candidato

    return w, historico, False
```

A condição usada no *backtracking* é uma forma da condição de Armijo. Ela exige redução suficiente, não apenas qualquer diminuição minúscula.

6.4.17.9.5 Por que adicionar amortecimento?

Longe do mínimo, a aproximação quadrática pode ser ruim. Uma Hessiana quase singular também pode produzir um passo enorme. Há três mecanismos comuns:

- usar $\eta < 1$ por busca linear;
- adicionar μI à Hessiana, como em métodos de região de confiança;
- usar regularização estatística, que também melhora o condicionamento.

Newton com busca linear preserva a rapidez local sem presumir que todo passo unitário seja seguro.

6.4.17.9.6 Custo computacional

Se $p = d + 1$ é o número de parâmetros, calcular gradiente e Hessiana densa custa aproximadamente $O(Np^2)$, e resolver o sistema custa $O(p^3)$. A memória da Hessiana é $O(p^2)$.

O gradiente descendente evita a Hessiana e custa aproximadamente $O(Np)$ por iteração. Newton

costuma usar menos iterações, mas cada uma é mais cara. Para muitos parâmetros, métodos quase-Newton como L-BFGS aproximam a curvatura sem armazenar a Hessiana completa.

6.4.17.9.7 Newton, Fisher scoring e IRLS

Em modelos lineares generalizados, **Fisher scoring** substitui a Hessiana observada pela informação de Fisher esperada. Para a regressão logística Bernoulli com ligação canônica, as matrizes relevantes coincidem na forma usual, e o algoritmo é frequentemente descrito tanto como Newton–Raphson quanto como IRLS.

6.4.17.9.8 Convergência local

Perto de um mínimo estrito, com Hessiana não singular e regularidade suficiente, Newton pode apresentar convergência quadrática: o erro é aproximadamente elevado ao quadrado de uma iteração para a seguinte. Essa é uma propriedade local. Ela não garante que uma inicialização arbitrária aceite sempre passos unitários nem resolve separação perfeita.

> ⚠ Probabilidades saturadas exigem cuidado
>
> Quando p_n está numericamente em zero ou um, $p_n(1-p_n)$ pode se tornar zero e a Hessiana perde informação naquela observação. Regularização, cálculo estável e amortecimento ajudam; recortar probabilidades sem documentar pode mascarar o problema.

6.4.17.9.9 Verificações

1. Confirme que $g^\top d < 0$ para uma direção de descida.
2. Verifique que a perda aceita pelo *backtracking* não aumenta.
3. Monitore o número de condição de H.
4. Compare o resultado com um otimizador de biblioteca.
5. Teste uma base perfeitamente separável com e sem regularização.

6.4.17.9.10 Exercícios

1. Derive a Hessiana a partir de Equação 6.47.
2. Mostre a equivalência entre Equação 6.50 e
   2.
3. Explique por que não devemos formar S explicitamente.
4. Compare custos de uma iteração de Newton e de gradiente para $N = 10^6$ e $p = 100$.
5. Modifique o código para penalizar também o intercepto e discuta por que essa não é a convenção usual.

6.4.17.10 Método do Máximo Declive

O **método do máximo declive**, ou gradiente descendente, escolhe em cada iteração a direção oposta ao gradiente. Para uma direção unitária v, a derivada direcional é

$$D_v E(w) = \nabla E(w)^{\mathsf{T}} v.$$

Pela desigualdade de Cauchy–Schwarz,

$$-\|\nabla E(w)\|_2 \leq \nabla E(w)^{\mathsf{T}} v \leq \|\nabla E(w)\|_2.$$

O menor valor ocorre em

$$v = -\frac{\nabla E(w)}{\|\nabla E(w)\|_2}.$$

Assim, **na geometria euclidiana dos parâmetros**, o gradiente negativo é a direção de queda local mais rápida. A atualização usual absorve a normalização na taxa de aprendizado:

$$w^{(k+1)} = w^{(k)} - \eta_k \nabla E(w^{(k)}). \tag{6.53}$$

O gradiente aponta perpendicularmente às curvas de nível. Quando essas curvas são muito alongadas, a trajetória pode atravessar repetidamente o vale e formar um zigue-zague.

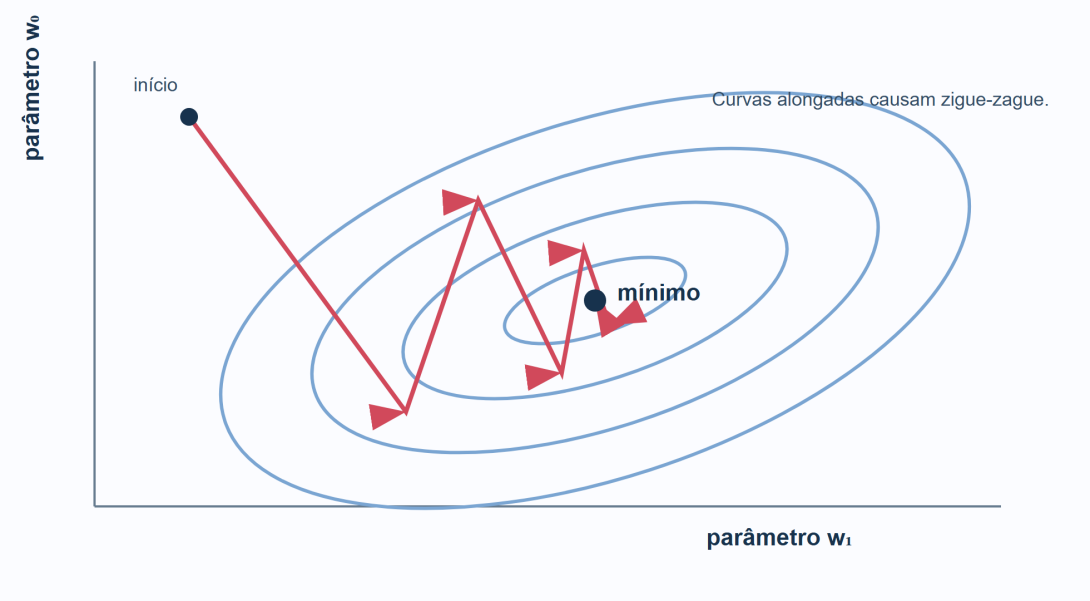


Figura 6.15: O máximo declive cruza as curvas de nível na direção perpendicular local, podendo oscilar em vales alongados.

6.4.17.10.1 “Mais íngreme” depende da escala

Se substituímos um atributo medido em metros pelo mesmo atributo em milímetros, a função preditiva pode representar a mesma relação, mas a geometria do espaço de parâmetros muda. Consequentemente, a direção euclidiana de máximo declive também muda. Padronização e pré-condicionamento reduzem esse problema.

Newton pode ser interpretado como um passo de gradiente corrigido pela curvatura:

$$d_{\text{Newton}} = -H^{-1}g.$$

Em vez de tratar todas as direções com a mesma escala, a Hessiana reescala o gradiente conforme a curvatura local.

6.4.17.10.2 Como escolher o tamanho do passo?

Há três abordagens comuns:

- **taxa fixa:** simples, mas exige ajuste;
- **agenda de decaimento:** reduz η_k ao longo das iterações;
- **busca linear:** escolhe um passo que produza redução suficiente.

Para regressão logística sem regularização, a Hessiana satisfaz

$$\nabla^2 E(w) \preceq \frac{1}{4N}\widetilde{X}^{\mathsf{T}}\widetilde{X},$$

pois $p_n(1-p_n) \le 1/4$. Portanto, a menor constante de Lipschitz L_{grad} satisfaz

$$L_{\text{grad}} \le \frac{\|\widetilde{X}\|_2^2}{4N} =: L_{\text{cota}}.$$

Com penalidade L_2 de intensidade λ, acrescentamos aproximadamente λ à cota nas direções penalizadas. Um passo $\eta \le 1/L_{\text{cota}}$ fornece uma escolha conservadora para redução monotônica no problema convexo e suave.

Calcular $\|\widetilde{X}\|_2$ exatamente pode ser caro; busca linear ou estimativas são alternativas práticas.

6.4.17.10.3 Busca linear por backtracking

Se $g = \nabla E(w)$ e $d = -g$, a condição de Armijo aceita η quando

$$E(w + \eta d) \le E(w) + c\eta g^{\mathsf{T}} d, \qquad 0 < c < 1. \tag{6.54}$$

Como $g^{\mathsf{T}} d = -\|g\|^2 < 0$, o lado direito exige uma redução proporcional ao passo e à inclinação.

```
def passo_backtracking(
    perda,
    w,
    gradiente,
    *,
    passo_inicial=1.0,
    contracao=0.5,
    armijo=1e-4,
):
    direcao = -gradiente
    valor = perda(w)
    derivada_direcional = gradiente @ direcao
    passo = passo_inicial

    while passo > 1e-12:
        candidato = w + passo * direcao
        if perda(candidato) <= (
            valor + armijo * passo * derivada_direcional
```

```
        ):
            return passo
        passo *= contracao

    raise RuntimeError("a busca linear não encontrou um passo")
```

A busca começa com um passo candidato e o reduz geometricamente. Ela realiza avaliações extras da perda, mas evita escolher previamente uma taxa adequada para toda a trajetória.

6.4.17.10.4 Algoritmo completo

```
inicialize w
repita:
    calcule perda E(w) e gradiente g
    se ||g|| satisfaz a tolerância: pare
    escolha  por taxa fixa, agenda ou busca linear
    w ← w − g
até atingir o limite de iterações
```

Para minilotes, a busca linear é menos direta porque a perda estimada é ruidosa. Nesse caso, agendas de taxa e otimizadores adaptativos são mais comuns.

6.4.17.10.5 Convergência

Convexidade garante que um ponto estacionário finito é global, mas não garante unicidade nem existência de minimizador finito em separação perfeita. Sob suavidade e um passo apropriado, o gradiente descendente converge. Para funções fortemente convexas, como frequentemente ocorre com regularização L_2 e posto adequado, a convergência pode ser linear em função do número de iterações.

O termo “linear” aqui descreve a taxa de convergência geométrica do erro, não o fato de o modelo usar uma pontuação linear.

6.4.17.10.6 Máximo declive versus Newton

Aspecto	Máximo declive	Newton
derivadas	gradiente	gradiente e Hessiana
memória adicional	$O(d)$	$O(d^2)$ para Hessiana densa
custo típico por iteração	$O(Nd)$	$O(Nd^2 + d^3)$
sensibilidade à escala	alta	menor, pela correção de curvatura

Aspecto	Máximo declive	Newton
uso com minilotes	natural	menos comum na forma exata
velocidade perto do ótimo	mais lenta	potencialmente quadrática

Descida local, objetivo global

A direção $-g$ é definida por uma aproximação local. A convexidade é o que permite relacionar essa movimentação local ao mínimo global, desde que o tamanho do passo e as demais condições sejam adequados.

6.4.17.10.7 Exercícios

1. Use Cauchy–Schwarz para demonstrar a direção de máximo declive.
2. Mostre que $g^{\mathsf{T}}(-g) = -\|g\|^2$.
3. Explique como multiplicar um atributo por 1 000 altera o caminho do gradiente descendente.
4. Implemente Equação 6.54 e registre quantas reduções de passo ocorrem por iteração.
5. Compare uma taxa fixa, backtracking e Newton na mesma amostra.

6.4.18 Computação

Esta seção transforma as derivações anteriores em um fluxo computacional completo. O objetivo não é apenas obter um vetor de pesos, mas produzir um artefato reproduzível que preserve pré-processamento, parâmetros, limiar e informações de convergência.

Compararemos duas implementações didáticas:

1. Newton amortecido, adequado a bases densas de dimensão moderada;
2. gradiente descendente, simples e adaptável a grandes volumes de dados e minilotes.

Em aplicações reais, bibliotecas consolidadas oferecem métodos quase-Newton, regularização, ponderação de classes e verificações numéricas adicionais. Implementar do zero é útil para compreender o algoritmo, não para substituir automaticamente essas bibliotecas.

6.4.18.1 Contrato dos dados

Adotaremos

$$X \in \mathbb{R}^{N \times d}, \qquad y \in \{0,1\}^N.$$

Cada linha de X é um exemplo; cada coluna é um atributo. Antes do ajuste, verificamos:

- mesmo número de linhas em X e entradas em y;
- somente valores finitos;
- exatamente dois rótulos, codificados como zero e um;
- presença de exemplos suficientes para a avaliação planejada.

Valores ausentes exigem uma política explícita. Remover linhas, preencher valores ou criar indicadores de ausência são decisões de modelagem e devem ser ajustadas somente com os dados de treinamento.

6.4.18.2 Divisão e pré-processamento

Para classificação, a divisão deve preservar aproximadamente a proporção das classes quando possível. O fluxo correto é:

```
dados brutos
    treino → ajustar imputação/padronização → ajustar modelo
    validação → aplicar transformações do treino → escolher hiperparâmetros
    teste → aplicar transformações do treino → avaliação final única
```

Padronização é particularmente útil para regularização e gradiente descendente. O intercepto não é padronizado nem normalmente penalizado.

6.4.18.3 Saídas de um treinamento

Uma rotina deve devolver mais que os coeficientes. Um resultado mínimo pode ser representado por:

```
from dataclasses import dataclass

import numpy as np


@dataclass
class ResultadoLogistico:
    pesos: np.ndarray
    intercepto: float
    convergiu: bool
    iteracoes: int
    motivo_parada: str
```

```
    perdas: list[float]
    normas_gradiente: list[float]
```

O motivo da parada diferencia convergência, limite de iterações, busca linear malsucedida e problema numérico. Silenciar essa diferença pode levar um sistema a publicar parâmetros incompletamente ajustados.

6.4.18.4 API de predição

Devemos separar pontuação, probabilidade e classe:

```
def pontuacao(X, pesos, intercepto):
    return np.asarray(X, dtype=float) @ pesos + intercepto


def probabilidade_positiva(X, pesos, intercepto):
    z = pontuacao(X, pesos, intercepto)
    p = np.empty_like(z)
    positivos = z >= 0
    p[positivos] = 1 / (1 + np.exp(-z[positivos]))
    exp_z = np.exp(z[~positivos])
    p[~positivos] = exp_z / (1 + exp_z)
    return p


def prever_classe(X, pesos, intercepto, limiar=0.5):
    if not 0 < limiar < 1:
        raise ValueError("o limiar deve estar entre zero e um")
    return (
        probabilidade_positiva(X, pesos, intercepto) >= limiar
    ).astype(int)
```

Essa separação impede que uma alteração de limiar seja confundida com novo treinamento e permite avaliar calibração usando as probabilidades.

6.4.18.5 Métricas complementares

A log-loss avalia probabilidades. Após escolher um limiar, a matriz de confusão permite calcular precisão e revocação:

$$\text{precisão} = \frac{VP}{VP + FP}, \qquad \text{revocação} = \frac{VP}{VP + FN}.$$

Quando um denominador é zero, a implementação deve definir o comportamento em vez de produzir silenciosamente um valor enganoso. Para bases desbalanceadas, acurácia isolada pode ocultar falha completa na classe rara.

Também devemos inspecionar:

- log-loss em treino, validação e teste;
- curva ROC ou precisão–revocação, conforme a aplicação;
- calibração das probabilidades;
- métricas por subgrupo relevante;
- estabilidade sob mudança temporal ou operacional.

6.4.18.6 Reprodutibilidade

Registre versão dos dados, divisão, semente, transformações, intensidade de regularização, algoritmo, tolerâncias e limite de iterações. Para os métodos determinísticos em lote apresentados aqui, a semente pode afetar a divisão dos dados, mesmo quando não afeta a otimização.

> **!** O conjunto de teste não escolhe o algoritmo
>
> Comparar Newton, gradiente, regularização ou limiares repetidamente no teste transforma esse conjunto em validação. Escolha configurações na validação e use o teste apenas para a estimativa final.

6.4.18.7 Pelo Método de Newton

A forma computacional correta do passo de Newton é

$$H_t d_t = -g_t, \qquad \widetilde{w}_{t+1} = \widetilde{w}_t + \eta_t d_t,$$

em que g_t e H_t são o gradiente e a Hessiana no estado atual. Observe o sinal negativo: somar $H_t^{-1} g_t$ moveria, em geral, para a direção errada.

6.4.18.7.1 Pseudocódigo

```
entrada: X, y,  , tolerâncias e limite de iterações
adicione a coluna de intercepto a X
inicialize w
```

```
para t = 0, 1, ...:
    z ← Xw
    p ← sigmoide_estável(z)
    g ← X (p − y)/N + gradiente_da_penalidade
    H ← X  diag(p(1−p)) X/N + Hessiana_da_penalidade

    se ||g|| for suficientemente pequena:
        devolva convergência

    resolva H d = −g
    escolha  por backtracking
    w ← w +  d

    se a variação relativa da perda for pequena:
        devolva convergência

devolva limite_de_iterações
```

O intercepto corresponde à primeira coordenada de w. Para não penalizá-lo, usamos uma matriz diagonal de regularização cujo primeiro elemento é zero.

6.4.18.7.2 Inicialização

O vetor nulo é uma escolha válida e determinística. Nesse ponto, todas as probabilidades valem $0{,}5$. Quando a classe positiva é rara, podemos inicializar apenas o intercepto com a prevalência suavizada:

$$b_0 = \ln\left(\frac{n_1 + \alpha}{n_0 + \alpha}\right),$$

em que n_1 e n_0 são as contagens das classes e $\alpha > 0$ evita logaritmo de zero. Os pesos começam em zero.

Inicialização aleatória não é necessária para quebrar simetria em uma única regressão logística e dificulta reproduzir resultados sem oferecer benefício automático.

6.4.18.7.3 Solução do sistema

Nunca forme H^{-1}. Resolva

```
direcao = np.linalg.solve(H, -gradiente)
```

Quando a matriz é simétrica positiva definida, uma fatoração de Cholesky é apropriada. Se o sistema estiver singular ou quase singular, verifique posto, escala, separação e regularização; recorrer silenciosamente à pseudoinversa pode ocultar um problema de modelagem.

O sistema também pode ser resolvido sem materializar a Hessiana em problemas grandes, usando produtos Hessiana–vetor e métodos como gradientes conjugados.

6.4.18.7.4 Critérios combinados

Uma implementação robusta pode declarar convergência quando pelo menos uma destas condições, devidamente escalada, é satisfeita:

$$\|g_t\|_\infty \le \varepsilon_g,$$

$$\frac{|E_{t+1} - E_t|}{\max(1, |E_t|)} \le \varepsilon_E,$$

$$\frac{\|\widetilde{w}_{t+1} - \widetilde{w}_t\|_2}{\max(1, \|\widetilde{w}_t\|_2)} \le \varepsilon_w.$$

O limite máximo permanece obrigatório. Uma perda “suficientemente pequena” não é um bom critério universal: com ruído, o ótimo pode estar longe de zero; em separação perfeita, perseguir perda quase zero pode fazer os coeficientes divergir.

6.4.18.7.5 Exemplo numérico

Considere um único atributo:

```
X = np.array([[-2.0], [-1.0], [0.5], [1.0], [2.0]])
y = np.array([0.0, 0.0, 0.0, 1.0, 1.0])
```

Com $\lambda = 0{,}1$ nos pesos e inicialização zero, Newton produz uma sequência como:

iteração	perda penalizada	norma do gradiente
0	0,6931	0,5590
1	valor menor	norma menor
2	aproximação refinada	próxima de zero

Os valores exatos dependem da normalização da penalidade e da regra de passo. A propriedade importante é que cada passo aceito pelo backtracking reduz a função objetivo.

Podemos verificar a solução final por três condições:

```
z = Xh @ w_final
p = 1 / (1 + np.exp(-z))
gradiente = Xh.T @ (p - y) / len(Xh) + penalidade @ w_final

assert np.linalg.norm(gradiente) < 1e-6
assert all(
    atual <= anterior + 1e-12
    for anterior, atual in zip(perdas, perdas[1:])
)
assert np.all(np.isfinite(w_final))
```

6.4.18.7.6 Diagnóstico de falhas

Sintoma	Causa provável	Ação
sistema singular	colunas redundantes ou pesos saturados	verificar posto e regularizar
perda aumenta	passo unitário inadequado ou erro no gradiente	usar backtracking e checar derivadas
coeficientes crescem sem parar	separação perfeita	regularizar e monitorar norma
valor não finito na perda	exponencial ou logaritmo instável	trabalhar diretamente com logits
convergência lenta	condicionamento ruim	padronizar e revisar atributos
validação piora	sobreajuste	regularização e seleção por validação

6.4.18.7.7 Custo e escolha do método

Newton é atraente quando o número de parâmetros é moderado e a matriz cabe confortavelmente na memória. Se d for grande, armazenar e fatorar uma Hessiana densa pode dominar o custo. Nessa situação, L-BFGS, gradiente estocástico ou Newton truncado podem ser melhores.

> **Compare com uma implementação de referência**
>
> Em testes, compare probabilidades, perda e gradiente com uma biblioteca consolidada usando a mesma regularização e convenção de intercepto. Coeficientes podem diferir se as bibliotecas normalizam a penalidade de maneiras distintas.

6.4.18.7.8 Exercícios

1. Corrija o pseudocódigo se a função objetivo for a soma, e não a média, das perdas.
2. Implemente os três critérios relativos de parada.
3. Force duas colunas idênticas e observe o sistema sem regularização.
4. Compare inicialização nula e intercepto pela prevalência.
5. Registre passo aceito, condição da Hessiana e norma dos pesos em cada iteração.

6.4.18.8 Pelo Método do Máximo Declive

A implementação em lote usa o gradiente completo

$$g_t = \frac{1}{N}\widetilde{X}^{\mathsf{T}}(p_t - y) + \lambda R\widetilde{w}_t,$$

em que $R = \mathrm{diag}(0, 1, \ldots, 1)$ deixa o intercepto sem penalidade. A atualização é

$$\widetilde{w}_{t+1} = \widetilde{w}_t - \eta_t g_t.$$

Não normalizamos obrigatoriamente g_t. Na forma $-g_t/\|g_t\|$, todos os passos teriam comprimento η_t, descartando a informação de que o gradiente pequeno pode indicar proximidade do ótimo. A versão não normalizada é a convenção mais comum.

6.4.18.8.1 Pseudocódigo

```
entrada: X, y,  , taxa ou busca linear, tolerâncias, max_iter
adicione a coluna de intercepto
inicialize w

para t = 0, 1, ...:
    calcule perda E(w) e gradiente g
    registre perda e ||g||

    se ||g|| satisfaz a tolerância:
```

```
        devolva convergência

    escolha
    candidato ← w − g

    se candidato não é finito:
        devolva falha numérica

    atualize w
    verifique variação relativa e limite de iterações

devolva o último estado com motivo da parada
```

6.4.18.8.2 Implementação em lote

```
import numpy as np


def objetivo_e_gradiente(Xh, y, w, lambda_l2):
    z = Xh @ w
    perda = np.mean(np.logaddexp(0.0, z) - y * z)
    perda += 0.5 * lambda_l2 * (w[1:] @ w[1:])

    p = np.empty_like(z)
    positivos = z >= 0
    p[positivos] = 1 / (1 + np.exp(-z[positivos]))
    exp_z = np.exp(z[~positivos])
    p[~positivos] = exp_z / (1 + exp_z)

    gradiente = Xh.T @ (p - y) / len(Xh)
    gradiente[1:] += lambda_l2 * w[1:]
    return perda, gradiente


def ajustar_logistica_gradiente(
    X,
    y,
```

```
    *,
    lambda_l2=1e-4,
    taxa=0.1,
    max_iter=10_000,
    tolerancia_grad=1e-7,
    tolerancia_perda=1e-10,
):
    X = np.asarray(X, dtype=float)
    y = np.asarray(y, dtype=float)
    Xh = np.column_stack([np.ones(len(X)), X])
    w = np.zeros(Xh.shape[1])
    perdas = []
    normas = []
    perda_anterior = None

    for iteracao in range(1, max_iter + 1):
        perda, gradiente = objetivo_e_gradiente(
            Xh, y, w, lambda_l2
        )
        norma = np.linalg.norm(gradiente)
        perdas.append(float(perda))
        normas.append(float(norma))

        if not np.isfinite(perda) or not np.all(
            np.isfinite(gradiente)
        ):
            motivo = "valor não finito"
            break

        if norma <= tolerancia_grad:
            motivo = "norma do gradiente"
            return w, perdas, normas, True, motivo

        if perda_anterior is not None:
            variacao = abs(perda - perda_anterior)
            escala = max(1.0, abs(perda_anterior))
            if variacao / escala <= tolerancia_perda:
```

```
                motivo = "variação relativa da perda"
                return w, perdas, normas, True, motivo

        w -= taxa * gradiente
        perda_anterior = perda
    else:
        motivo = "limite de iterações"

    return w, perdas, normas, False, motivo
```

Essa versão usa taxa fixa. Ela é apropriada para estudo, mas deve verificar se a perda realmente diminui. Podemos substituir a linha de atualização pela busca de Armijo apresentada anteriormente.

6.4.18.8.3 Validação das entradas

Antes do treinamento, acrescente verificações:

```
def validar_classificacao_binaria(X, y):
    if X.ndim != 2:
        raise ValueError("X deve ser bidimensional")
    if y.ndim != 1 or len(X) != len(y):
        raise ValueError("y deve ter um rótulo por exemplo")
    if len(X) == 0:
        raise ValueError("a amostra não pode ser vazia")
    if not np.all(np.isfinite(X)) or not np.all(np.isfinite(y)):
        raise ValueError("os dados devem ser finitos")
    if not np.all(np.isin(y, (0, 1))):
        raise ValueError("os rótulos devem pertencer a {0, 1}")
```

Uma amostra contendo apenas uma classe ainda pode ser aceita matematicamente, mas exige interpretação cuidadosa e geralmente não permite avaliação discriminativa útil.

6.4.18.8.4 Exemplo reproduzível

```
X = np.array([
    [-2.0,  0.2],
    [-1.5, -0.4],
    [-0.5,  0.3],
    [ 0.4, -0.2],
```

```
    [ 1.0,  0.5],
    [ 1.8, -0.1],
])
y = np.array([0, 0, 0, 1, 1, 1], dtype=float)

w_homogeneo, perdas, normas, convergiu, motivo = (
    ajustar_logistica_gradiente(
        X,
        y,
        lambda_l2=0.01,
        taxa=0.2,
        max_iter=5_000,
    )
)

intercepto = w_homogeneo[0]
pesos = w_homogeneo[1:]
probabilidades = probabilidade_positiva(
    X, pesos, intercepto
)

print("convergiu:", convergiu, "-", motivo)
print("perda inicial/final:", perdas[0], perdas[-1])
print("probabilidades:", probabilidades)
```

Verificações úteis:

```
assert np.all(np.isfinite(w_homogeneo))
assert perdas[-1] <= perdas[0]
assert np.all((probabilidades >= 0) & (probabilidades <= 1))
```

Comparar apenas a perda final com a inicial é uma condição fraca: uma execução pode terminar melhor que começou e ainda oscilar muito. Para taxa fixa em lote, inspecione toda a sequência ou use busca linear.

6.4.18.8.5 Minilotes

Para minilotes, selecione índices, calcule o gradiente apenas no bloco e atualize imediatamente:

```
rng = np.random.default_rng(42)

for epoca in range(max_epocas):
    indices = rng.permutation(len(Xh))

    for inicio in range(0, len(Xh), tamanho_lote):
        lote = indices[inicio:inicio + tamanho_lote]
        _, gradiente = objetivo_e_gradiente(
            Xh[lote], y[lote], w, lambda_l2
        )
        w -= taxa * gradiente
```

Há uma sutileza na regularização: se a função de lote adiciona a penalidade completa em cada minilote, a quantidade efetiva de regularização por época depende do número de lotes e da convenção de taxa. Bibliotecas definem cuidadosamente a normalização; ao comparar implementações, confira se λ possui o mesmo significado.

6.4.18.8.6 Monitoramento

Um gráfico útil contém perda de treino e validação por época. Há três padrões típicos:

- ambas diminuem: o ajuste ainda melhora;
- treino diminui e validação aumenta: possível sobreajuste;
- ambas oscilam ou divergem: taxa alta, dados mal escalados ou erro de implementação.

Parada antecipada escolhe o estado de menor perda de validação, não necessariamente o último. Como no Pocket, é preciso copiar os parâmetros ao guardar o melhor estado.

6.4.18.8.7 Escolha do limiar

O treinamento da log-loss não determina automaticamente o limiar de decisão. Depois de ajustar o modelo:

1. gere probabilidades na validação;
2. avalie candidatos a limiar segundo o custo ou métrica desejada;
3. fixe o limiar;
4. avalie uma única vez no teste.

Se o custo de falso positivo é C_{FP} e o de falso negativo é C_{FN}, sob condições simples e probabilidades calibradas, a decisão de menor custo prevê positivo quando

$$p(x) \geq \frac{C_{FP}}{C_{FP} + C_{FN}}.$$

Custos, prevalência e ações reais precisam ser definidos com os responsáveis pela aplicação.

6.4.18.8.8 Comparação final dos métodos

Critério	Newton amortecido	Gradiente em lote
parâmetros moderados	geralmente eficiente	funciona, com mais iterações
muitos atributos	Hessiana pode ser proibitiva	memória menor
taxa de aprendizado	busca de passo ainda útil	escolha central
curvatura	usada explicitamente	ignorada
minilotes	não natural na forma exata	natural
implementação	mais complexa	mais simples
diagnóstico essencial	condição da Hessiana	escala e taxa

6.4.18.8.9 Recapitulação da regressão logística

1. A pontuação linear modela log-odds.
2. A sigmoide converte pontuação em probabilidade.
3. A entropia cruzada vem da verossimilhança Bernoulli.
4. O gradiente é $\widetilde{X}^{\mathsf{T}}(p - y)/N$.
5. A Hessiana é $\widetilde{X}^{\mathsf{T}} S \widetilde{X}/N$.
6. Regularização controla coeficientes e melhora condicionamento.
7. Probabilidade, classe e ação são saídas conceitualmente distintas.
8. Convergência de treino não substitui validação, calibração e análise por subgrupo.

> **!** Do modelo ao sistema
>
> Um classificador em produção inclui dados, transformação, parâmetros, calibração, limiar, tratamento de entradas inválidas e monitoramento. Salvar apenas o vetor de pesos não preserva o comportamento completo.

6.4.18.8.10 Exercícios integradores

1. Execute Newton e gradiente na mesma divisão e compare perdas, iterações e tempo.
2. Produza uma amostra separável e observe os coeficientes sem regularização.
3. Selecione um limiar que maximize revocação sob uma precisão mínima.
4. Construa um diagrama de calibração em dez faixas de probabilidade.

5. Simule mudança de prevalência no teste e observe acurácia, precisão, revocação e calibração.
6. Explique por que escolher o limiar no teste invalida sua função de avaliação final.

6.5 Exercícios de múltipla escolha

Assinale uma alternativa em cada questão e confira depois as justificativas.

1. No perceptron com rótulos $y \in \{-1, +1\}$, um exemplo está corretamente classificado quando:

   a. $y(\mathbf{w}^{\mathsf{T}}\mathbf{x} + b) > 0$.

   b. $y(\mathbf{w}^{\mathsf{T}}\mathbf{x} + b) < 0$.

   c. $\mathbf{w} = \mathbf{0}$ para qualquer entrada.

   d. b é necessariamente positivo.

2. Ao encontrar um exemplo mal classificado, a atualização básica do perceptron para os pesos é:

   a. $\mathbf{w} \leftarrow \mathbf{w} - \eta y \mathbf{x}$.

   b. $\mathbf{w} \leftarrow \mathbf{w} + \eta y \mathbf{x}$.

   c. $\mathbf{w} \leftarrow \eta \mathbf{w}^2$.

   d. $\mathbf{w} \leftarrow \mathbf{x} - y$.

3. O teorema de convergência do perceptron garante término em número finito de atualizações quando os dados são:

   a. linearmente separáveis com margem positiva.

   b. sempre não separáveis.

   c. produzidos por regressão.

   d. normalizados, independentemente dos rótulos.

4. O algoritmo Pocket é útil principalmente porque:

   a. transforma qualquer problema em separável.

   b. guarda a melhor solução observada durante a execução.

   c. calcula probabilidades perfeitamente calibradas.

   d. dispensa um critério de parada.

5. Na regressão linear por mínimos quadrados, a condição de ortogonalidade dos resíduos é:

   a. $X^{\mathsf{T}}(X\mathbf{w} - \mathbf{y}) = \mathbf{0}$.

   b. $X\mathbf{y} = \mathbf{0}$.

   c. $\mathbf{w}^{\mathsf{T}}\mathbf{w} = 0$.

   d. $X^{\mathsf{T}}X = 0$.

6. Por que `solve` ou uma fatoração QR costuma ser preferível a calcular explicitamente $(X^{\mathsf{T}}X)^{-1}$?

   a. Porque evita toda multiplicação.

   b. Porque tende a oferecer maior estabilidade numérica.

   c. Porque elimina a necessidade de dados.

   d. Porque sempre produz coeficientes inteiros.

7. O coeficiente de determinação R^2 compara o erro do modelo com:

   a. um preditor de referência baseado na média da saída.

   b. a quantidade de classes.

   c. o maior valor singular de X.

   d. o erro de classificação zero–um.

8. A função logística transforma uma pontuação real em um valor no intervalo:

a. $(-\infty, +\infty)$.

b. $[-1, 1]$.

c. $(0, 1)$.

d. $[0, +\infty)$.

9. Na regressão logística binária, escolher um limiar de decisão diferente de $0{,}5$:

a. é impossível matematicamente.

b. pode alterar o compromisso entre falsos positivos e falsos negativos.

c. modifica obrigatoriamente os pesos já treinados.

d. torna todas as probabilidades iguais.

10. Em comparação com o máximo declive, o método de Newton utiliza:

a. somente o sinal dos rótulos.

b. informação de segunda ordem por meio da Hessiana.

c. uma taxa de aprendizado sempre igual a zero.

d. apenas uma observação do conjunto.

i Gabarito comentado

1. **a.** O produto positivo indica que a pontuação possui o mesmo sinal do rótulo.
2. **b.** A parcela $\eta y\mathbf{x}$ desloca a pontuação do exemplo na direção correta.
3. **a.** Separabilidade com margem sustenta a cota de atualizações; isso não é garantia de generalização.
4. **b.** Em dados não separáveis, os pesos correntes podem piorar; o Pocket preserva a melhor configuração encontrada.
5. **a.** Nas equações normais, o resíduo é ortogonal ao espaço coluna de X.
6. **b.** Formar a inversa e as equações normais pode ampliar erros de ponto flutuante e piorar o condicionamento.
7. **a.** R^2 mede a redução relativa da soma de quadrados frente à previsão constante pela

média.

8. **c.** Para toda entrada real finita, a sigmoide produz valor estritamente entre 0 e 1.
9. **b.** O limiar converte probabilidade em ação e deve refletir custos e requisitos da aplicação.
10. **b.** A Hessiana descreve a curvatura local e permite construir um passo de Newton.

Capítulo 7

Transformação Linear

Modelos lineares combinam atributos por uma soma ponderada. Essa simplicidade favorece interpretação e otimização, mas uma fronteira linear no espaço original não representa todos os padrões. O XOR e uma classe cercada por outra são exemplos imediatos.

Podemos ampliar a classe de funções sem abandonar os algoritmos lineares transformando a representação de cada entrada:

$$\phi : X \longrightarrow Z, \qquad x \longmapsto \phi(x).$$

O modelo passa a calcular

$$h_w(x) = h_0(w^{\mathsf{T}}\phi(x)), \tag{7.1}$$

em que h_0 pode ser o sinal, a identidade ou a sigmoide. O modelo é linear nos **atributos transformados** e nos parâmetros w, embora seja não linear em relação à entrada original.

> **O nome do capítulo**
>
> Neste contexto, "linearização" significa representar uma relação não linear por um modelo linear em novas características. O mapa ϕ não precisa ser uma transformação linear no sentido estrito da álgebra linear; mapas polinomiais, por exemplo, são não lineares.

7.1 Visão geral da transformação de atributos

7.1.1 Exemplo radial

Considere pontos no plano rotulados pela distância à origem:

$$y = \begin{cases} +1, & x_1^2 + x_2^2 < r^2, \\ -1, & x_1^2 + x_2^2 > r^2. \end{cases}$$

Nenhuma reta separa a região interna da externa. Entretanto, com

$$\phi(x_1, x_2) = x_1^2 + x_2^2,$$

o problema torna-se unidimensional: basta comparar $\phi(x)$ com o limiar r^2.

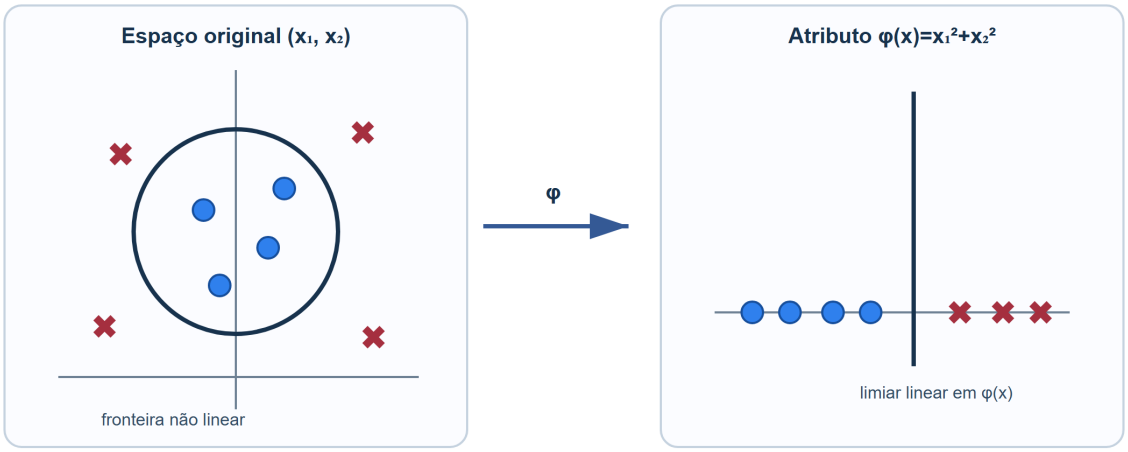


Figura 7.1: Uma fronteira circular no espaço original torna-se um limiar linear após a transformação radial.

Não precisamos inverter ϕ depois do treinamento. Para prever um novo exemplo x, calculamos $\phi(x)$ e aplicamos o modelo aprendido. A saída continua se referindo ao exemplo original.

7.1.2 Transformação da amostra

Para

$$D = \{(x_1, y_1), \ldots, (x_N, y_N)\},$$

a amostra transformada é

$$\phi(D) = \{(\phi(x_1), y_1), \dots, (\phi(x_N), y_N)\}.$$

Os rótulos não mudam. O algoritmo linear recebe $\phi(x_n)$ como seus atributos e escolhe

$$w^* \in \arg\min_w \frac{1}{N} \sum_{n=1}^{N} \ell(h_0(w^\mathsf{T} \phi(x_n)), y_n)\,. \tag{7.2}$$

A hipótese final no domínio original é a composição $h_{w^*} = h_0 \circ (w^{*\mathsf{T}} \phi)$.

7.1.3 Exemplo polinomial

Para duas entradas, um mapa quadrático completo pode ser

$$\phi(x_1, x_2) = (1, x_1, x_2, x_1^2, x_1 x_2, x_2^2)\,.$$

Uma pontuação linear em $\phi(x)$ é

$$w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_1 x_2 + w_5 x_2^2,$$

um polinômio quadrático no espaço original. Dependendo dos coeficientes, sua fronteira de nível pode ser elipse, parábola, hipérbole ou um caso degenerado.

7.1.4 O que permanece linear?

É útil distinguir três espaços:

Espaço	Objeto	Pode ser não linear?
entrada original	x	sim, a fronteira pode ser curva
características	$\phi(x)$	o mapa pode ser não linear
parâmetros	$w^\mathsf{T}\phi(x)$	é linear em w

Essa última linearidade permite reutilizar Perceptron, mínimos quadrados e regressão logística sem mudar sua estrutura matemática.

7.1.5 A transformação faz parte do modelo

Treino, validação, teste e produção devem usar exatamente o mesmo mapa. Se ϕ depende de parâmetros estimados — médias, desvios, categorias, componentes principais — esses parâmetros devem ser aprendidos apenas no treino e armazenados com o modelo.

```
import numpy as np


def mapa_quadratico(X):
    """Transforma uma matriz com duas colunas."""
    x1 = X[:, 0]
    x2 = X[:, 1]
    return np.column_stack([
        np.ones(len(X)),
        x1,
        x2,
        x1**2,
        x1 * x2,
        x2**2,
    ])
```

Aplicar transformações diferentes entre treinamento e predição é um erro de contrato de dados, mesmo quando as dimensões coincidem.

7.1.6 Capacidade e custo

Transformar pode facilitar o ajuste, mas aumenta a capacidade da classe. Para d atributos, o número de monômios de grau até q é

$$\binom{d+q}{q}.$$

Com $d = 100$ e $q = 2$, já são $\binom{102}{2} = 5\,151$ características incluindo o termo constante. Isso aumenta memória, tempo e risco de sobreajuste.

Regularização e validação tornam-se especialmente importantes. Uma transformação que reduz o erro de treinamento pode piorar o erro fora da amostra.

7.1.7 Transformações explícitas e implícitas

Há duas estratégias:

- **explícita:** calcular e armazenar $\phi(x)$;
- **implícita:** calcular apenas produtos $\phi(x)^{\mathsf{T}}\phi(x')$ por meio de um núcleo (*kernel*).

O artifício do núcleo será útil quando a dimensão de Z for muito grande ou até infinita, desde que o algoritmo dependa dos dados por produtos escalares.

7.1.8 Limitações

- A transformação adequada depende da estrutura do problema.
- Muitas características podem amplificar ruído e redundância.
- Polinômios de grau alto podem extrapolar de forma extrema.
- Um mapa fixo pode não capturar invariâncias relevantes.
- Interpretar coeficientes transformados exige considerar toda a composição, não pesos isolados.

7.1.8.1 Questões iniciais

1. Em que sentido Equação 7.1 é linear e em que sentido pode ser não linear?
2. Por que não precisamos calcular ϕ^{-1} para fazer previsões?
3. Quantos monômios de grau até dois existem para $d = 10$?
4. Dê um exemplo de transformação que dependeria de estatísticas do treino e poderia causar vazamento de dados.
5. Explique por que melhorar o ajuste não prova melhora de generalização.

7.2 Linearizar Elipses em torno da origem

Suponha que os exemplos positivos ocupem o interior de uma elipse centrada na origem e os negativos fiquem do lado de fora. No plano original, a fronteira é curva e não pode ser representada por um único Perceptron linear nos atributos (x_1, x_2).

Uma elipse alinhada aos eixos, com semieixos $a, b > 0$, satisfaz

$$\frac{x_1^2}{a^2} + \frac{x_2^2}{b^2} = 1. \tag{7.3}$$

Para um círculo de raio r, temos $a = b = r$ e a equação é $x_1^2 + x_2^2 = r^2$. O quadrado no raio é indispensável.

7.2.1 Da elipse à reta

Defina

$$\phi(x_1, x_2) = (z_1, z_2) = (x_1^2, x_2^2).$$

No espaço transformado, Equação 7.3 torna-se

$$\frac{z_1}{a^2} + \frac{z_2}{b^2} = 1,$$

que é a equação de uma reta. A classificação “interior positivo” pode ser escrita como

$$h(x) = \text{sinal}_+\left(1 - \frac{x_1^2}{a^2} - \frac{x_2^2}{b^2}\right). \tag{7.4}$$

No vetor homogêneo

$$\tilde{\phi}(x) = (1, x_1^2, x_2^2),$$

os pesos são

$$w = \left(1, -\frac{1}{a^2}, -\frac{1}{b^2}\right),$$

e Equação 7.4 assume a forma linear $h(x) = \text{sinal}_+(w^\mathsf{T}\tilde{\phi}(x))$.

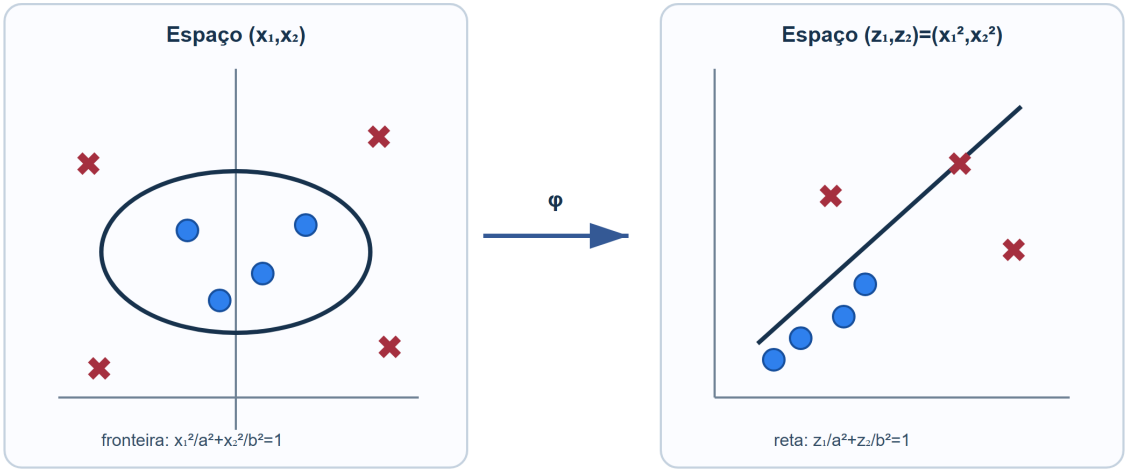


Figura 7.2: A elipse no espaço original corresponde a uma reta no espaço das coordenadas quadradas.

O espaço transformado acessível satisfaz $z_1, z_2 \geq 0$; não ocupamos todo $\mathbb{R}^2$. Isso não impede o classificador, mas lembra que ϕ pode perder informação: os pontos (x_1, x_2), $(-x_1, x_2)$ e outras mudanças de sinal possuem a mesma imagem.

7.2.2 Aprendendo a elipse

No exemplo anterior, a e b pareciam conhecidos. Em aprendizado, fornecemos os atributos $(1, x_1^2, x_2^2)$ e deixamos o algoritmo estimar

$$g(z) = w_0 + w_1 z_1 + w_2 z_2.$$

Quando $w_0 > 0$ e $w_1, w_2 < 0$, a fronteira $g(z) = 0$ corresponde a

$$\frac{x_1^2}{-w_0/w_1} + \frac{x_2^2}{-w_0/w_2} = 1,$$

com

$$a = \sqrt{-\frac{w_0}{w_1}}, \qquad b = \sqrt{-\frac{w_0}{w_2}}.$$

Um classificador linear não impõe automaticamente esses sinais. Para outros valores, a fronteira pode ser vazia, ilimitada ou classificar o exterior como positivo. A interpretação geométrica deve

ser feita após o ajuste.

7.2.3 Elipses giradas

Uma elipse centrada na origem e não necessariamente alinhada aos eixos é descrita por

$$x^\top A x = 1,$$

em que A é uma matriz simétrica positiva definida. Em duas dimensões,

$$x^\top A x = a_{11}x_1^2 + 2a_{12}x_1x_2 + a_{22}x_2^2.$$

Precisamos então do termo cruzado:

$$\phi(x_1, x_2) = (1, x_1^2, x_1x_2, x_2^2)\,.$$

O coeficiente de x_1x_2 permite girar os eixos principais da elipse. Sem ele, a classe contém apenas elipses alinhadas aos eixos.

7.2.4 Elipses fora da origem

Uma elipse centrada em $c = (c_1, c_2)$ satisfaz

$$(x - c)^\top A(x - c) = 1.$$

Ao expandir, surgem termos quadráticos, lineares e constante. Por isso, o mapa quadrático completo

$$\phi(x_1, x_2) = (1, x_1, x_2, x_1^2, x_1x_2, x_2^2)$$

representa cônicas transladadas. A presença desses seis atributos não garante que toda fronteira aprendida seja uma elipse; coeficientes podem produzir parábolas, hipérboles ou formas degeneradas.

7.2.5 Igualdade dos erros após a composição

Se g é uma hipótese no espaço transformado e

$$h = g \circ \phi,$$

então, para todo exemplo, $h(x_n) = g(\phi(x_n))$. Usando a mesma perda,

$$E_D(h) = \frac{1}{N} \sum_n \ell(h(x_n), y_n) = E_{\phi(D)}(g).$$

Não há "ida e volta" geométrica: a igualdade decorre simplesmente de aplicar a composição aos mesmos exemplos e rótulos.

7.2.6 Capacidade da classe transformada

Se $\mathcal{G}$ é a classe de separadores lineares em Z e ϕ é fixado, a classe no domínio original é

$$\mathcal{H}_\phi = \{g \circ \phi : g \in \mathcal{G}\}.$$

Como padrões realizados por $\mathcal{H}_\phi$ correspondem a padrões de $\mathcal{G}$ sobre imagens $\phi(x)$,

$$d_{\mathrm{VC}}(\mathcal{H}_\phi) \leq d_{\mathrm{VC}}(\mathcal{G}).$$

Para $\phi(x) = (x_1^2, x_2^2)$ e separadores afins em duas dimensões, $d_{\mathrm{VC}}(\mathcal{G}) = 3$, portanto $d_{\mathrm{VC}}(\mathcal{H}_\phi) \leq 3$. A desigualdade pode ser estrita porque as imagens ficam restritas ao primeiro quadrante e pontos distintos podem coincidir após o mapeamento.

Se escolhemos entre muitos mapas usando os dados, a classe efetiva é a união das classes associadas. Isso não é proibido, mas sua complexidade deve ser contabilizada e a seleção deve usar validação apropriada.

7.2.7 Implementação

```
import numpy as np


def atributos_elipse(X, girada=False, transladada=False):
    X = np.asarray(X, dtype=float)
    x1, x2 = X[:, 0], X[:, 1]
    colunas = [np.ones(len(X))]
```

```
    if transladada:
        colunas.extend([x1, x2])

    colunas.append(x1**2)
    if girada:
        colunas.append(x1 * x2)
    colunas.append(x2**2)
    return np.column_stack(colunas)
```

A ordem das colunas deve ser armazenada com o modelo. Alterá-la na predição preserva a dimensão, mas associa cada peso ao atributo errado.

7.2.7.1 Exercícios

1. Corrija o classificador se a classe positiva estiver fora da elipse.
2. Derive a e b a partir de (w_0, w_1, w_2) sob os sinais adequados.
3. Expanda $(x - c)^\top A(x - c)$ e identifique os seis coeficientes.
4. Mostre que $x_1 x_2$ muda de sinal sob a reflexão $(x_1, x_2) \mapsto (-x_1, x_2)$, ao contrário dos quadrados.
5. Explique por que selecionar o grau polinomial no conjunto de teste produz uma avaliação otimista.

7.3 Generalização da Transformação

O exemplo da elipse ilustra uma ideia mais ampla: podemos construir um novo vetor com funções das coordenadas originais e, depois, aplicar a esse vetor um algoritmo linear. Se $x \in \mathbb{R}^d$, uma transformação de atributos tem a forma

$$\phi : \mathbb{R}^d \longrightarrow \mathbb{R}^{d'}, \qquad x \longmapsto \phi(x).$$

O número d' não precisa ser igual a d. Ele pode ser menor, como em redução de dimensionalidade, ou maior, quando criamos interações e termos não lineares.

7.3.1 Transformação polinomial

Uma transformação polinomial de grau máximo Q reúne todos os monômios

$$x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_d^{\alpha_d} \quad \text{tais que} \quad \alpha_j \in \mathbb{N}_0 \quad \text{e} \quad \sum_{j=1}^{d} \alpha_j \leq Q.$$

O vetor $\alpha = (\alpha_1, \ldots, \alpha_d)$ é chamado **multiíndice** e $|\alpha| = \sum_j \alpha_j$ é o grau total do monômio. Para $d = 2$ e $Q = 2$, por exemplo,

$$\phi_2(x_1, x_2) = (1, x_1, x_2, x_1^2, x_1 x_2, x_2^2)\,. \tag{7.5}$$

Uma função linear nesse novo vetor produz

$$w^{\mathsf{T}} \phi_2(x) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_1 x_2 + w_5 x_2^2,$$

isto é, um polinômio quadrático no espaço original. O modelo é **não linear em** x, mas continua **linear nos parâmetros** w. Essa distinção permite usar regressão linear, regressão logística ou outros algoritmos lineares após a transformação.

7.3.2 Quantos atributos são criados?

O número de monômios de grau exatamente q em d variáveis é

$$\binom{d+q-1}{q}.$$

Somando os graus de 0 a Q, obtemos

$$m(d, Q) = \sum_{q=0}^{Q} \binom{d+q-1}{q} = \binom{d+Q}{Q}. \tag{7.6}$$

Essa contagem inclui o termo constante. Se o viés for tratado separadamente, a dimensão do vetor sem a constante será

$$d' = \binom{d+Q}{Q} - 1.$$

Para duas variáveis,

$$d' = \binom{Q+2}{Q} - 1 = \frac{Q(Q+3)}{2}.$$

Assim, com $d = 2$, os graus $Q = 1, 2, 3, 4$ geram, respectivamente, 3, 6, 10 e 15 coeficientes quando contamos o termo constante.

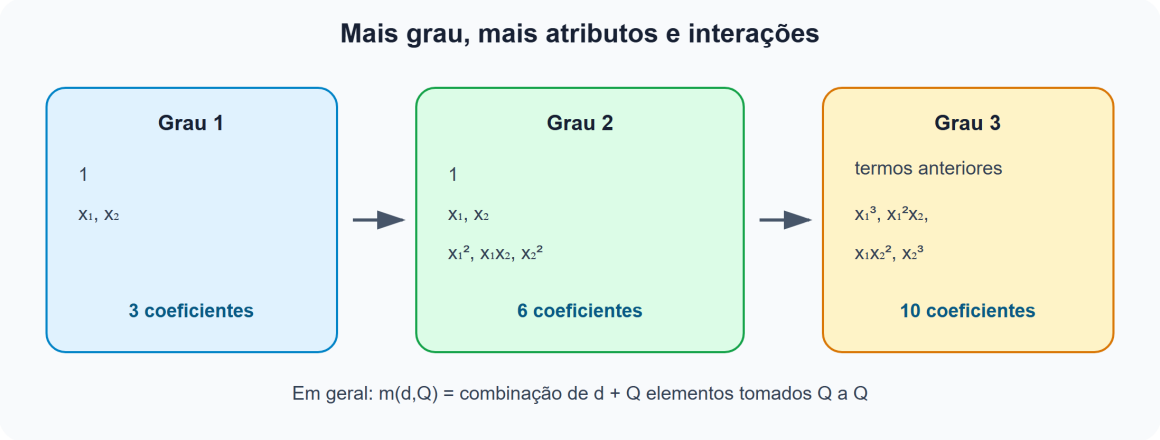


Figura 7.3: O aumento do grau cria novas interações e eleva rapidamente a dimensão do espaço de atributos.

Em dimensões maiores, o crescimento combinatório é mais severo. Para $d = 20$ e $Q = 3$, Equação 7.6 fornece $\binom{23}{3} = 1771$ coeficientes. Uma matriz de projeto com N exemplos pode, portanto, passar de $N \times 20$ para $N \times 1771$.

7.3.3 Expressividade, generalização e custo

Ao aumentar Q, a classe de hipóteses contém fronteiras mais flexíveis. Como as classes polinomiais são aninhadas, o menor erro de treinamento não pode aumentar: um modelo de grau $Q + 1$ pode sempre atribuir peso zero aos novos termos. Isso não significa que o erro fora da amostra sempre diminua.

Três efeitos devem ser considerados:

1. **Aproximação:** graus maiores podem representar relações que um modelo simples não consegue capturar.
2. **Estimação:** mais parâmetros tornam o ajuste mais sensível às particularidades e ao ruído da amostra.
3. **Computação:** armazenar a matriz transformada e calcular produtos, gradientes ou inversas passa a exigir mais memória e tempo.

Para separadores afins em $\mathbb{R}^{d'}$, a dimensão VC da classe linear completa é $d' + 1$. Porém, a classe composta $\{g \circ \phi : g \in \mathcal{G}\}$ pode ter dimensão VC menor, pois os vetores $\phi(x)$ ocupam apenas uma parte estruturada do espaço transformado. Ainda assim, o crescimento de d' é um bom alerta para o risco de sobreajuste.

7.3.4 Como escolher o grau

O grau Q é um hiperparâmetro. Uma prática adequada é:

1. separar os dados de teste e não usá-los na escolha;
2. comparar os graus candidatos por validação ou validação cruzada;
3. ajustar no conjunto de treino qualquer normalização necessária;
4. aplicar exatamente a mesma transformação na validação e no teste;
5. selecionar o grau e a regularização em conjunto.

A padronização merece atenção. Se x_j tiver valores grandes, x_j^Q pode ficar numericamente enorme. Em geral, padronizamos as variáveis originais usando estatísticas do treino e só então geramos os termos polinomiais. Regularização L_2 ou L_1 também ajuda a controlar coeficientes e reduzir variância.

> ⚠ Evite vazamento de dados
>
> Média, desvio-padrão, grau e força de regularização não devem ser escolhidos com informação do conjunto de teste. Em uma validação cruzada, toda a transformação precisa ser ajustada novamente dentro de cada partição de treino.

7.3.5 Implementação sem bibliotecas especializadas

O exemplo a seguir constrói 3 para uma matriz com duas colunas:

```
import numpy as np


def atributos_quadraticos_2d(X):
    X = np.asarray(X, dtype=float)
    if X.ndim != 2 or X.shape[1] != 2:
        raise ValueError("X deve ter formato (n_amostras, 2)")

    x1, x2 = X[:, 0], X[:, 1]
    return np.column_stack([
```

```
        np.ones(len(X)),
        x1,
        x2,
        x1**2,
        x1 * x2,
        x2**2,
    ])
```

Em aplicações reais, uma *pipeline* deve guardar a ordem das colunas e os parâmetros de normalização. Gerar os mesmos atributos em ordem diferente durante a predição produz resultados incorretos sem necessariamente causar erro de dimensão.

7.3.5.1 Exercícios

1. Liste todos os monômios de grau máximo 3 em duas variáveis.
2. Calcule $m(5, 2)$ e interprete a diferença entre contar ou não o termo constante.
3. Mostre por que o erro mínimo de treinamento não aumenta quando se passa do grau Q para $Q + 1$.
4. Para $x = (2, -3)$, calcule manualmente $\phi_2(x)$ na ordem usada em
   3.
5. Explique por que padronizar toda a base antes da validação cruzada constitui vazamento de dados.

7.4 O Artifício do Núcleo

Transformações explícitas podem criar milhares — ou infinitos — atributos. Entretanto, alguns algoritmos nunca precisam acessar cada coordenada de $\phi(x)$ separadamente: eles utilizam apenas produtos escalares entre exemplos transformados. O **artifício do núcleo**, também chamado **truque do kernel**, explora exatamente essa situação.

Uma função núcleo é calculada no espaço de entrada:

$$K : X \times X \longrightarrow \mathbb{R}, \qquad K(x, z) = \langle \phi(x), \phi(z) \rangle_{\mathcal{F}}, \tag{7.7}$$

em que $\mathcal{F}$ é o espaço de atributos. Se existe uma expressão direta e barata para $K(x, z)$, podemos obter o produto escalar sem construir os vetores $\phi(x)$ e $\phi(z)$.

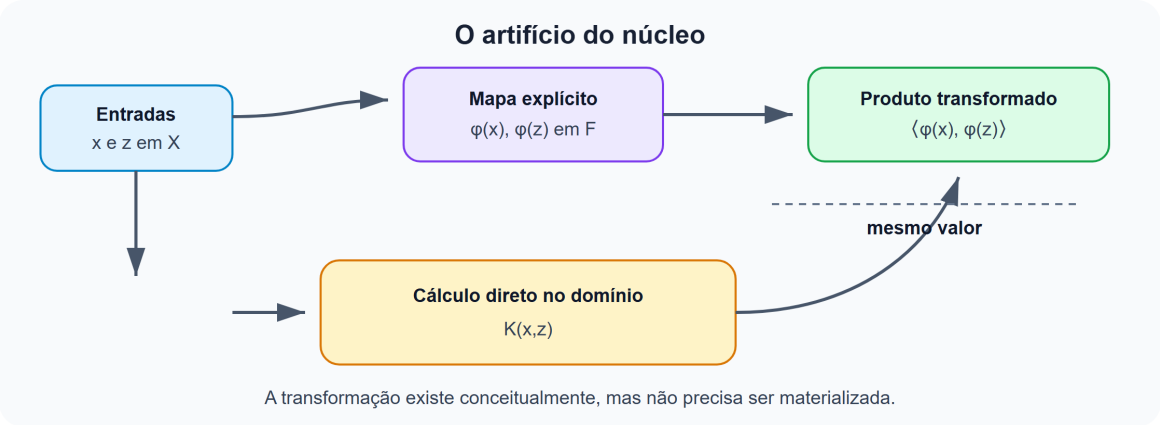


Figura 7.4: O núcleo calcula diretamente, no espaço original, o mesmo produto escalar que seria obtido após a transformação explícita.

O ganho não vem simplesmente do fato de X ter dimensão menor. Ele depende de duas condições: o núcleo deve ser barato de avaliar e o algoritmo deve poder ser formulado apenas em termos desses produtos escalares. Métodos de margens, como máquinas de vetores de suporte, são o exemplo clássico.

7.4.1 Princípio

Considere uma amostra $D = \{(x_i, y_i)\}_{i=1}^{N}$. Em muitos problemas regularizados, a solução no espaço de atributos pode ser escrita como

$$w = \sum_{i=1}^{N} \alpha_i \phi(x_i). \tag{7.8}$$

Para prever em um novo ponto x, calculamos

$$\begin{aligned} f(x) &= \langle w, \phi(x) \rangle + b \\ &= \sum_{i=1}^{N} \alpha_i \langle \phi(x_i), \phi(x) \rangle + b \\ &= \sum_{i=1}^{N} \alpha_i K(x_i, x) + b. \end{aligned} \tag{7.9}$$

Observe a mudança de representação. No método primal, armazenamos as coordenadas de w; na representação dual, armazenamos coeficientes α_i associados aos exemplos de treino. O núcleo recebe **dois pontos do domínio**, e não um vetor de pesos e um ponto.

Durante o treinamento, os produtos entre todos os pares formam a **matriz de Gram**:

$$G_{ij} = K(x_i, x_j), \qquad G \in \mathbb{R}^{N \times N}.$$

Essa matriz resume a geometria da amostra no espaço de atributos. Sua entrada G_{ij} mede a similaridade induzida pelo núcleo entre os exemplos x_i e x_j.

7.4.2 Exemplo: núcleo polinomial quadrático

Em duas dimensões, tome

$$K(x, z) = (1 + x^\mathsf{T} z)^2.$$

Expandindo,

$$K(x, z) = 1 + 2x_1 z_1 + 2x_2 z_2 + x_1^2 z_1^2 + 2x_1 x_2 z_1 z_2 + x_2^2 z_2^2.$$

Essa expressão é o produto escalar associado ao mapa

$$\phi(x) = \left(1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1 x_2, x_2^2\right).$$

Logo, podemos calcular um produto em seis dimensões usando somente $x^\mathsf{T} z$ no espaço original. Os fatores $\sqrt{2}$ são necessários para que o coeficiente dos termos cruzados seja exatamente 2.

7.4.3 Quando o artifício se aplica?

Não basta que a função de perda contenha $w^\mathsf{T} x$. É preciso reescrever o procedimento de treinamento e a predição em termos de produtos entre exemplos, como em Equação 7.8 e Equação 7.9. SVMs, regressão *ridge* em forma dual e PCA com núcleo possuem formulações desse tipo. Um código primal comum de regressão logística não aceita um núcleo automaticamente; seria necessária uma formulação ou um algoritmo apropriado.

Há também uma troca computacional. A transformação explícita costuma custar em função de d', enquanto o método de núcleo armazena uma matriz $N \times N$. Para amostras muito grandes, o custo de memória $O(N^2)$ e o custo do treinamento podem superar a economia obtida ao não materializar $\phi(x)$. Portanto, núcleos são especialmente atraentes quando d' é enorme e N ainda é administrável.

```python
import numpy as np


def matriz_gram_polinomial(X, grau=2, constante=1.0):
    X = np.asarray(X, dtype=float)
    return (constante + X @ X.T) ** grau
```

Para uma matriz `X` de formato `(N, d)`, o resultado tem formato `(N, N)`. Em uma implementação robusta, os mesmos parâmetros do núcleo usados no treino devem ser preservados para a predição.

7.4.3.1 Exercícios

1. Verifique manualmente o valor de $K(x, z)$ para $x = (1, 2)$ e $z = (3, -1)$ usando tanto a fórmula direta quanto o mapa explícito.
2. Explique a diferença entre as dimensões de X, $\phi(X)$ e da matriz de Gram.
3. Mostre que a matriz de Gram de um núcleo simétrico também é simétrica.
4. Compare a memória de uma matriz explícita $N \times d'$ com a da matriz de Gram $N \times N$ quando $N = 10\,000$ e $d' = 500$.

7.4.4 Teorema de Mercer

Nem toda função de similaridade pode substituir um produto escalar. Para ser um núcleo real válido, $K : X \times X \to \mathbb{R}$ deve satisfazer duas propriedades fundamentais.

Primeiro, deve ser simétrico:

$$K(x, z) = K(z, x).$$

Segundo, deve ser **semidefinido positivo**: para qualquer quantidade finita de pontos $x_1, \ldots, x_N \in X$ e quaisquer coeficientes reais $c_1, \ldots, c_N$,

$$\sum_{i=1}^{N} \sum_{j=1}^{N} c_i c_j K(x_i, x_j) \geq 0. \tag{7.10}$$

Se G é a matriz de Gram, essa condição equivale a

$$c^{\mathsf{T}} G c \geq 0 \quad \text{para todo } c \in \mathbb{R}^N.$$

Como G é simétrica, isso também equivale a dizer que todos os seus autovalores são não

negativos. Pequenos autovalores negativos podem aparecer numericamente por arredondamento; valores negativos relevantes indicam que a função ou a implementação não produz uma matriz semidefinida positiva naquela amostra.

7.4.5 Por que a positividade é necessária?

Se existe um mapa ϕ tal que $K(x, z) = \langle \phi(x), \phi(z) \rangle$, então

$$\begin{aligned} \sum_{i,j} c_i c_j K(x_i, x_j) &= \sum_{i,j} c_i c_j \langle \phi(x_i), \phi(x_j) \rangle \\ &= \left\| \sum_i c_i \phi(x_i) \right\|^2 \\ &\geq 0. \end{aligned}$$

Portanto, 4 não é um detalhe técnico arbitrário: ela decorre do fato de que o quadrado de uma norma nunca é negativo.

7.4.6 Mercer e o espaço associado

Em linguagem comum de aprendizado de máquina, a condição anterior é frequentemente chamada "condição de Mercer". Convém, porém, separar dois resultados relacionados:

- o teorema de **Moore–Aronszajn** garante que todo núcleo simétrico e semidefinido positivo determina um espaço de Hilbert de funções com núcleo reprodutor e um mapa de atributos, possivelmente infinito-dimensional;
- o **teorema de Mercer**, sob hipóteses adicionais como domínio compacto e núcleo contínuo, simétrico e positivo, fornece uma expansão espectral em autofunções.

Em uma forma simplificada, a expansão de Mercer é

$$K(x, z) = \sum_{r=1}^{\infty} lambda_r e_r(x) e_r(z), \qquad \lambda_r \geq 0,$$

onde e_r são autofunções ortonormais de um operador integral associado ao núcleo. Isso sugere o mapa

$$\phi(x) = \left(\sqrt{\lambda_1} e_1(x), \sqrt{\lambda_2} e_2(x), \ldots \right),$$

pois seu produto escalar recupera $K(x, z)$. O espaço pode ter dimensão infinita, mas o algoritmo acessa apenas avaliações do núcleo.

> **Semidefinido não significa estritamente positivo**
>
> É permitido que $c^\top Gc = 0$ para algum vetor não nulo c. Isso ocorre quando os vetores transformados possuem dependência linear. Um núcleo semidefinido positivo continua sendo válido nesse caso.

7.4.7 Regras para construir novos núcleos

Se K_1 e K_2 são núcleos válidos e $a, b \geq 0$, então também são válidos:

$$aK_1(x, z) + bK_2(x, z), \qquad K_1(x, z)K_2(x, z).$$

Além disso, para qualquer função $q : X \to \mathbb{R}$,

$$K(x, z) = q(x)q(z)$$

é um núcleo válido. Essas regras permitem combinar noções de similaridade sem precisar descobrir explicitamente o mapa resultante. Subtrair núcleos ou aplicar uma transformação arbitrária, por outro lado, pode destruir a semidefinição positiva.

7.4.8 Verificação numérica em uma amostra

```
import numpy as np


def verificar_gram(G, tolerancia=1e-10):
    G = np.asarray(G, dtype=float)
    if G.ndim != 2 or G.shape[0] != G.shape[1]:
        raise ValueError("G deve ser uma matriz quadrada")

    simetrica = np.allclose(G, G.T, atol=tolerancia)
    G_sim = (G + G.T) / 2
    autovalores = np.linalg.eigvalsh(G_sim)
    psd = autovalores.min() >= -tolerancia
    return simetrica, psd, autovalores
```

Esse teste verifica apenas a matriz dos pontos fornecidos. Ele pode encontrar um contraexemplo e provar que uma função não é um núcleo válido, mas resultados positivos em algumas amostras

não demonstram a propriedade para todo conjunto finito possível. A validade geral exige uma prova ou um resultado teórico conhecido.

7.4.8.1 Exercícios

1. Prove diretamente que $K(x, z) = x^{\mathsf{T}} z$ satisfaz
   4.
2. Mostre que a soma de dois núcleos válidos com coeficientes não negativos também é válida.
3. Dê um exemplo de matriz simétrica que não seja semidefinida positiva.
4. Explique por que testar apenas uma matriz de Gram não prova que uma função é núcleo em todo o domínio.
5. Qual é o mapa de atributos associado a $K(x, z) = 1 + x^{\mathsf{T}} z$?

7.4.9 Vantagem

O artifício do núcleo permite trabalhar com uma geometria não linear sem armazenar explicitamente todos os atributos de $\phi(x)$. Suas principais vantagens são:

- representar interações de dimensão muito alta com uma fórmula curta;
- reutilizar algoritmos baseados em produtos escalares;
- expressar diferentes noções de similaridade por meio da escolha de K;
- obter fronteiras não lineares mantendo uma otimização convexa em métodos como a SVM.

Essas vantagens não eliminam os custos. O treinamento normalmente usa a matriz de Gram $N \times N$, e uma predição por Equação 7.9 pode exigir uma avaliação do núcleo para cada coeficiente α_i não nulo. Em uma SVM, apenas os exemplos com $\alpha_i \neq 0$ são vetores de suporte, o que pode tornar a predição esparsa; ainda assim, seu número depende dos dados e dos hiperparâmetros.

7.4.10 Núcleo linear

$$K(x, z) = x^{\mathsf{T}} z.$$

Ele não cria uma fronteira não linear: corresponde ao mapa identidade. Serve como referência e costuma ser competitivo quando os atributos já são numerosos, como em representações esparsas de texto.

7.4.11 Núcleo polinomial

$$K(x, z) = (\gamma x^{\mathsf{T}} z + c_0)^Q,$$

com $\gamma > 0$, $c_0 \geq 0$ e grau inteiro $Q \geq 1$. O núcleo representa monômios e interações até o grau Q quando $c_0 > 0$. Se $c_0 = 0$, contém somente termos de grau exatamente Q. Os parâmetros γ e c_0 alteram a escala relativa das contribuições e devem ser tratados como parte do modelo.

7.4.12 Núcleo gaussiano ou RBF

$$K(x, z) = \exp(-\gamma \|x - z\|^2) = \exp\left(-\frac{\|x - z\|^2}{2\sigma^2}\right),$$

onde

$$\gamma = \frac{1}{2\sigma^2}.$$

O núcleo RBF vale 1 quando $x = z$ e se aproxima de zero à medida que a distância cresce. Seu mapa de atributos é infinito-dimensional, mas a avaliação do núcleo depende apenas da distância euclidiana.

O parâmetro controla a escala da vizinhança:

- γ **pequeno** (ou σ grande): a similaridade decai lentamente e a fronteira tende a variar suavemente;
- γ **grande** (ou σ pequeno): a influência de cada ponto fica localizada e a fronteira pode se tornar muito irregular.

Na SVM, γ interage com o parâmetro de regularização C. Valores altos de C penalizam fortemente violações na amostra; valores menores aceitam mais violações para favorecer uma margem regularizada. Ambos devem ser escolhidos com validação, nunca pelo desempenho no teste.

> **!** A escala dos atributos muda o núcleo RBF
>
> Uma variável medida em milhares pode dominar $\|x - z\|^2$ e anular a influência de variáveis medidas entre zero e um. Ajuste a padronização somente no treino e aplique os mesmos parâmetros aos demais dados.

7.4.13 Escolha prática

Uma sequência razoável é começar pelo núcleo linear, estabelecer uma linha de base e só então testar uma geometria não linear. Para o RBF, buscamos C e γ em escala logarítmica por validação cruzada. Para o polinomial, grau, C, γ e c_0 ampliam o espaço de busca e exigem maior cuidado.

Também é necessário considerar o tamanho da amostra. Quando N é muito grande e um mapa explícito moderado existe, um método linear sobre os atributos transformados ou uma aproximação de núcleo pode ser mais eficiente que armazenar a matriz de Gram completa.

```
import numpy as np


def kernel_rbf(X, Z, gamma):
    X = np.asarray(X, dtype=float)
    Z = np.asarray(Z, dtype=float)
    if gamma <= 0:
        raise ValueError("gamma deve ser positivo")

    norma_X = np.sum(X**2, axis=1)[:, None]
    norma_Z = np.sum(Z**2, axis=1)[None, :]
    dist2 = np.maximum(norma_X + norma_Z - 2 * X @ Z.T, 0.0)
    return np.exp(-gamma * dist2)
```

A correção com `maximum` remove pequenos valores negativos de distância quadrática causados por arredondamento. O resultado possui formato `(len(X), len(Z))`, útil tanto para a matriz de treino quanto para comparar novos exemplos aos exemplos armazenados.

7.4.13.1 Exercícios

1. Calcule o núcleo RBF para pontos iguais e interprete o resultado.
2. Descreva o efeito de $\gamma \to 0$ e de $\gamma \to \infty$ para pontos distintos.
3. Explique por que a padronização influencia o RBF, mas não corrige por si só sobreajuste.
4. Compare o custo de armazenar $N \times d'$ com $N \times N$ em dois cenários escolhidos por você.
5. Por que um bom resultado com núcleo linear é uma referência útil antes de testar núcleos mais flexíveis?

7.4.14 Aplicações

O uso sistemático de funções núcleo antecede as SVMs modernas, mas ganhou grande visibilidade com os classificadores de margem na década de 1990. A ideia, entretanto, não pertence a um único algoritmo: sempre que um método admite uma formulação baseada em produtos escalares, podemos investigar uma versão com núcleo.

7.4.15 Máquinas de vetores de suporte

Na forma dual de uma SVM, o treinamento utiliza os exemplos por meio de

$$y_i y_j K(x_i, x_j).$$

Depois do ajuste, a função de decisão é

$$f(x) = \sum_{i \in \mathcal{S}} \alpha_i y_i K(x_i, x) + b,$$

onde $\mathcal{S}$ é o conjunto de vetores de suporte. A classificação é sinal$(f(x))$. O núcleo altera a geometria na qual a margem é maximizada, enquanto C controla a penalização das violações.

7.4.16 Regressão com núcleo

Na regressão *ridge* com núcleo, buscamos uma função da forma

$$f(x) = \sum_{i=1}^{N} \alpha_i K(x_i, x).$$

Com perda quadrática e regularização, os coeficientes satisfazem

$$(G + \lambda I)\alpha = y,$$

em que G é a matriz de Gram e $\lambda > 0$ controla a regularização. A adição de λI também melhora a estabilidade numérica quando G é singular ou quase singular.

7.4.17 PCA com núcleo

A análise de componentes principais comum encontra direções lineares de maior variância. A **PCA com núcleo** efetua a análise no espaço $\mathcal{F}$ sem formar explicitamente $\phi(x)$. Para isso, trabalha com uma matriz de Gram centralizada:

$$G_c = HGH, \qquad H = I - \frac{1}{N}\mathbf{1}\mathbf{1}^\top.$$

Centralizar os atributos originais não substitui essa operação: o que precisa ter média zero são os vetores $\phi(x_i)$ no espaço de atributos. Os autovetores de G_c fornecem coordenadas não lineares úteis para visualização e redução de dimensionalidade.

7.4.18 Dados estruturados e núcleos especializados

O domínio não precisa ser $\mathbb{R}^d$. É possível definir núcleos para sequências, árvores, grafos, conjuntos ou histogramas, desde que a função resultante seja simétrica e semidefinida positiva. Nesses casos, o núcleo incorpora conhecimento sobre a estrutura do objeto e permite aplicar um método geométrico sem converter tudo para um vetor denso de tamanho fixo.

Em texto, por exemplo, o núcleo linear sobre vetores esparsos de frequência é uma base forte. Núcleos mais elaborados podem comparar subsequências, mas seu custo e a dificuldade de escolher hiperparâmetros precisam ser comparados a representações explícitas modernas.

7.4.19 Um roteiro de decisão

Antes de adotar um método com núcleo, responda:

1. O método possui uma formulação baseada em produtos escalares?
2. Há uma noção de similaridade adequada ao domínio e comprovadamente válida como núcleo?
3. O número N de exemplos permite armazenar e processar N^2 valores?
4. A predição terá latência aceitável com a quantidade esperada de exemplos ativos ou vetores de suporte?
5. Há validação para escolher o núcleo e seus hiperparâmetros sem usar o conjunto de teste?

Se N for muito grande, alternativas incluem mapas explícitos de baixa dimensão, aproximações por características aleatórias, métodos de Nyström e modelos lineares ou neurais treinados em lotes. O núcleo é uma ferramenta poderosa, não uma etapa obrigatória.

7.4.20 Exemplo de fluxo experimental

```
# Pseudocódigo: a sintaxe exata depende da biblioteca utilizada.
separar_treino_validacao_teste()

for nome_kernel, parametros in configuracoes:
    for particao_treino, particao_validacao in validacao_cruzada(treino):
        normalizador.ajustar(particao_treino.X)
        Xtr = normalizador.transformar(particao_treino.X)
        Xva = normalizador.transformar(particao_validacao.X)

        modelo = SVM(kernel=nome_kernel, **parametros)
        modelo.ajustar(Xtr, particao_treino.y)
```

```
        registrar(modelo.avaliar(Xva, particao_validacao.y))

escolher_configuracao_por_validacao()
avaliar_uma_vez_no_teste()
```

O normalizador é reajustado em cada partição para evitar vazamento. A avaliação final no teste é realizada uma única vez, depois que todas as decisões de modelagem foram encerradas.

7.4.20.1 Exercícios

1. Identifique onde o núcleo aparece na predição de uma SVM.
2. Explique por que $G + \lambda I$ é numericamente mais estável que G quando alguns autovalores são próximos de zero.
3. Derive a matriz H para $N = 3$ e verifique que $H\mathbf{1} = 0$.
4. Proponha uma função de similaridade para um domínio estruturado e discuta o que faltaria provar para usá-la como núcleo.
5. Em que cenário um mapa explícito pode ser preferível à matriz de Gram?

7.4.21 Síntese do capítulo

Transformar atributos permite converter relações não lineares no espaço original em relações lineares em outro espaço. Mapas polinomiais tornam essa construção concreta, mas sua dimensão cresce combinatoriamente. O artifício do núcleo evita materializar o mapa quando o algoritmo depende somente de produtos escalares. Em troca, desloca o custo para a matriz de Gram e para as avaliações de similaridade. A escolha entre mapa explícito e núcleo deve considerar expressividade, generalização, memória, tempo de treinamento e custo de predição.

7.5 Exercícios de múltipla escolha

As questões integram transformação de atributos, mapas polinomiais e métodos de núcleo. Há somente uma alternativa correta em cada item.

1. Ao aplicar $\phi(x_1, x_2) = (x_1^2, x_2^2)$, a fronteira linear $w_1 z_1 + w_2 z_2 + b = 0$ no espaço transformado corresponde, no espaço original, a uma fronteira:

   a. necessariamente linear.

   b. quadrática.

c. cúbica.

d. independente de x_1 e x_2.

2. Depois de uma transformação não linear ϕ, chamar o modelo de linear significa que ele é linear:

a. nos atributos transformados e em seus parâmetros.

b. em qualquer variável do problema original.

c. apenas no número de exemplos.

d. somente no tempo de execução.

3. Quantos monômios de grau total até p existem em d variáveis, incluindo o termo constante?

a. dp.

b. $d + p$.

c. $\binom{d+p}{p}$.

d. 2^{dp} em todos os casos.

4. Aumentar o grau de um mapa polinomial tende a:

a. reduzir sempre a capacidade e o custo.

b. aumentar expressividade e também o risco de sobreajuste.

c. preservar exatamente a mesma classe de hipóteses.

d. garantir erro de teste zero.

5. O artifício do núcleo é aplicável quando o algoritmo pode ser escrito em termos de:

a. produtos escalares entre exemplos transformados.

b. ordenação alfabética dos atributos.

c. apenas uma coordenada por exemplo.

d. operações bit a bit.

6. Uma função núcleo válida satisfaz $k(x, z) = \langle \phi(x), \phi(z) \rangle$ para algum mapa de atributos. Consequentemente, sua matriz de Gram deve ser:

   a. antissimétrica.

   b. triangular com diagonal nula.

   c. simétrica e semidefinida positiva.

   d. composta somente por números inteiros.

7. Qual expressão define o núcleo RBF?

   a. $x^{\mathsf{T}} z$.

   b. $(x^{\mathsf{T}} z + c)^p$.

   c. $\exp(-\gamma \|x - z\|^2)$.

   d. $\|x + z\|$.

8. No núcleo RBF, um valor muito alto de γ geralmente torna a similaridade:

   a. mais local, decaindo rapidamente com a distância.

   b. constante e igual a um.

   c. independente dos dados.

   d. necessariamente negativa.

9. Uma limitação importante de métodos que armazenam uma matriz de Gram densa para N exemplos é o consumo de memória de ordem:

   a. $O(1)$.

   b. $O(\log N)$.

   c. $O(N)$.

d. $O(N^2)$.

10. A escolha de hiperparâmetros como grau, γ ou regularização deve ser realizada principalmente com:

    a. o conjunto de teste final.

    b. dados de validação ou validação cruzada.

    c. exemplos inventados após conhecer o teste.

    d. somente o erro de treinamento.

Gabarito comentado

1. **b.** Substituir $z_1 = x_1^2$ e $z_2 = x_2^2$ produz uma equação de segundo grau nas coordenadas originais.
2. **a.** A fronteira é linear em $\phi(x)$ e nos pesos, embora possa ser curva em x.
3. **c.** A contagem de combinações com repetição resulta em $\binom{d+p}{p}$.
4. **b.** Mais termos permitem ajustar padrões mais complexos, mas elevam capacidade, custo e sensibilidade à amostra.
5. **a.** O núcleo substitui diretamente $\langle \phi(x), \phi(z) \rangle$ sem materializar ϕ.
6. **c.** Essa é a condição que permite interpretar a função como produto interno em um espaço de atributos.
7. **c.** O RBF decai exponencialmente com a distância quadrática.
8. **a.** Quanto maior γ, menor a escala espacial de influência de cada exemplo.
9. **d.** A matriz contém uma similaridade para cada par de exemplos.
10. **b.** O teste deve permanecer intocado; a validação orienta as decisões de modelagem.

Capítulo 8

Redes Neurais

Nos capítulos anteriores, estudamos modelos lineares e transformações de atributos. Redes neurais combinam essas duas ideias de modo adaptativo: em vez de escolher manualmente todos os atributos, o treinamento ajusta sucessivas transformações a partir dos dados.

Em aprendizado supervisionado, recebemos uma amostra

$$D = \{(x_i, y_i)\}_{i=1}^{N}, \qquad x_i \in X, \quad y_i \in Y,$$

e escolhemos uma hipótese parametrizada $h_\theta : X \to \widehat{Y}$. O vetor θ reúne todos os pesos e vieses da rede. Uma função de perda

$$\ell : \widehat{Y} \times Y \longrightarrow [0, \infty)$$

mede a qualidade de uma predição individual. O erro empírico médio é

$$E_D(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(h_\theta(x_i), y_i).$$

O algoritmo de treinamento procura parâmetros que reduzam esse erro, normalmente com regularização e um método baseado em gradientes. O objetivo real continua sendo generalizar para exemplos não observados; memorizar a amostra não é suficiente.

8.1 Da unidade artificial à rede

Um **neurônio artificial** recebe um vetor $x \in \mathbb{R}^d$, calcula uma combinação afim e aplica uma função de ativação g:

$$z = w^{\mathsf{T}} x + b, \qquad a = g(z). \tag{8.1}$$

Aqui, $w \in \mathbb{R}^d$ contém os pesos, $b \in \mathbb{R}$ é o viés, z é a pré-ativação e a é a ativação produzida. O viés permite deslocar o limiar; sem ele, muitas fronteiras seriam forçadas a passar pela origem.

Uma camada com m neurônios calcula todas essas operações em paralelo:

$$z = Wx + b, \qquad a = g(z),$$

onde $W \in \mathbb{R}^{m \times d}$, $b \in \mathbb{R}^m$ e g é aplicada componente a componente. Cada linha de W corresponde aos pesos de um neurônio.

Uma rede **feedforward**, ou sem realimentação, encadeia camadas sem ciclos. Para duas camadas parametrizadas,

$$a^{(1)} = g_1(W^{(1)} x + b^{(1)}),$$

$$h_\theta(x) = g_2(W^{(2)} a^{(1)} + b^{(2)}).$$

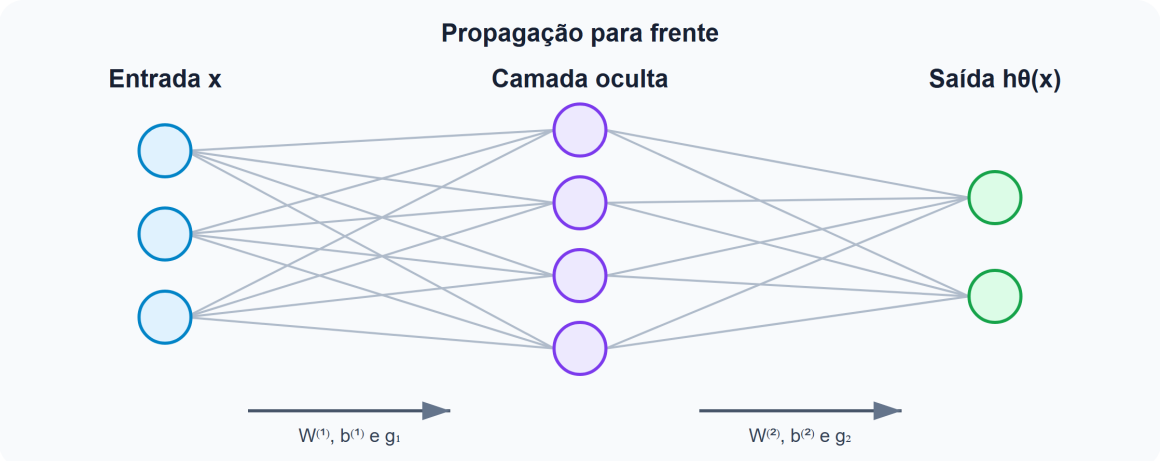


Figura 8.1: Fluxo de uma rede feedforward: a entrada é transformada por uma camada oculta e chega à camada de saída.

Chamamos de **camadas ocultas** as transformações entre a entrada e a saída. A entrada em si não possui parâmetros e, por isso, não costuma ser contada como camada treinável. A expressão "rede de uma camada" pode ser ambígua: alguns autores contam apenas camadas ocultas, enquanto outros contam todas as camadas com pesos. Neste capítulo, diremos explicitamente "uma camada oculta" sempre que essa distinção importar.

8.2 Por que a ativação precisa ser não linear?

Se todas as ativações forem a identidade, duas camadas afins se reduzem a uma única transformação afim:

$$W^{(2)}(W^{(1)}x + b^{(1)}) + b^{(2)} = \underbrace{W^{(2)}W^{(1)}}_{W'}\,x + \underbrace{W^{(2)}b^{(1)} + b^{(2)}}_{b'}.$$

Empilhar transformações lineares, portanto, não aumenta a classe de funções representáveis. Ativações não lineares entre as camadas impedem essa redução e permitem construir fronteiras e aproximações complexas. A camada de saída, contudo, deve ser escolhida de acordo com a tarefa: identidade para regressão irrestrita, sigmoide para uma probabilidade binária e *softmax* para probabilidades entre várias classes são escolhas comuns.

8.3 Arquitetura e aprendizado

A **arquitetura** define o grafo de operações: quantidade de camadas, largura de cada camada, ativações e conexões. Os pesos e vieses são parâmetros aprendidos. Já largura, profundidade, taxa de aprendizado e força de regularização são hiperparâmetros escolhidos por validação.

O treinamento de uma rede feedforward pode ser resumido em quatro etapas repetidas:

1. **propagação para frente:** calcular as ativações e a predição;
2. **perda:** comparar a predição ao alvo;
3. **retropropagação:** calcular derivadas pela regra da cadeia;
4. **atualização:** modificar os parâmetros com um otimizador.

```
# Pseudocódigo de uma camada densa
z = W @ x + b
a = ativacao(z)
```

Embora a expressão seja curta, é essencial acompanhar os formatos. Se `x` tem d componentes e a camada possui m neurônios, `W` tem formato `(m, d)`, enquanto `b`, `z` e `a` têm m componentes. Erros de dimensão estão entre os problemas mais comuns nas primeiras implementações.

8.3.0.1 Exercícios

1. Determine os formatos de W e b para uma camada que recebe 8 atributos e produz 32 ativações.
2. Quantos parâmetros treináveis essa camada possui?
3. Demonstre que a composição de três transformações afins ainda é afim.
4. Diferencie parâmetro, hiperparâmetro, pré-ativação e ativação.
5. Explique por que uma perda baixa no treino não garante boa generalização.

8.4 Funções de Ativação

Uma função de ativação transforma a pré-ativação z produzida por um neurônio. Nas camadas ocultas, sua função principal é introduzir não linearidade. Na saída, ela também determina o domínio da predição e deve ser compatível com a tarefa e com a perda.

Durante a retropropagação, a derivada da ativação participa da regra da cadeia. Assim, saturação, descontinuidade da derivada e escala dos gradientes afetam diretamente a otimização.

Figura 8.2: Comparação qualitativa entre ReLU, Leaky ReLU, sigmoide e tangente hiperbólica.

8.4.1 Função de Ativação ReLU (Unidade Linear Retificada)

A unidade linear retificada é definida por

$$\mathrm{ReLU}(z) = \max(0, z).$$

Sua derivada, exceto em $z = 0$, é

$$\mathrm{ReLU}'(z) = \begin{cases} 0, & z < 0, \\ 1, & z > 0. \end{cases}$$

No ponto zero a função não é diferenciável; bibliotecas adotam uma subderivada convencional, normalmente zero. A ReLU é barata e não satura no semieixo positivo, por isso é uma escolha frequente em camadas ocultas.

No semieixo negativo, porém, o gradiente é zero. Um neurônio que recebe pré-ativações negativas para todos os exemplos pode deixar de atualizar seus pesos: é o problema da **ReLU morta**. Inicialização, taxa de aprendizado e variantes com inclinação negativa ajudam a reduzi-lo.

8.4.2 Função de Ativação Leaky ReLU

A Leaky ReLU mantém uma pequena inclinação no lado negativo:

$$\mathrm{LeakyReLU}_{\alpha}(z) = \begin{cases} z, & z > 0, \\ \alpha z, & z \le 0, \end{cases} \qquad 0 < \alpha < 1.$$

Sua derivada vale 1 no lado positivo e α no lado negativo, exceto pela convenção escolhida em zero. Dessa forma, um neurônio recebe algum gradiente mesmo quando sua pré-ativação é negativa. Valores como $\alpha = 0{,}01$ são comuns, mas α é um hiperparâmetro e não uma constante universal.

8.4.3 Função de Ativação Sigmoide

A sigmoide logística é

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \qquad 0 < \sigma(z) < 1.$$

Sua derivada pode ser calculada a partir da própria saída:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z)).$$

O valor máximo da derivada é $1/4$, atingido em $z = 0$. Para $|z|$ grande, a saída se aproxima de zero ou um e a derivada se aproxima de zero. Em redes profundas, multiplicar sucessivamente derivadas pequenas contribui para o **desaparecimento do gradiente**.

A sigmoide é especialmente adequada à camada de saída de uma classificação binária, onde pode representar $P(Y = 1 \mid x)$. Em camadas ocultas modernas, ReLU e variantes costumam ser preferidas.

Uma implementação direta de e^{-z} pode transbordar para z muito negativo. Funções numericamente estáveis tratam separadamente os casos $z \geq 0$ e $z < 0$.

8.4.4 Função de Ativação Tangente Hiperbólica (tanh)

A tangente hiperbólica é

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \qquad -1 < \tanh(z) < 1,$$

e sua derivada é

$$\frac{d}{dz}\tanh(z) = 1 - \tanh^2(z).$$

Ao contrário da sigmoide, a tanh é centrada em zero. Isso pode facilitar a otimização quando ativações positivas e negativas são úteis. Ela também satura para entradas de grande magnitude e, portanto, não elimina o desaparecimento do gradiente. É frequente em formulações clássicas de redes recorrentes.

8.4.5 Softmax

Para classificação entre K classes mutuamente exclusivas, a camada de saída produz *logits* $z \in \mathbb{R}^K$. A softmax os converte em probabilidades:

$$\operatorname{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \qquad i = 1, \ldots, K.$$

Cada componente é positiva e a soma é um. Somar a mesma constante a todos os logits não altera o resultado. Usamos essa propriedade para evitar transbordamento:

$$\operatorname{softmax}(z)_i = \frac{e^{z_i - m}}{\sum_{j} e^{z_j - m}}, \qquad m = \max_j z_j.$$

A softmax acopla as saídas: aumentar a probabilidade de uma classe reduz a massa disponível para as outras. Por isso, ela se aplica a classes exclusivas. Em classificação multirrótulo, na qual

várias classes podem ser verdadeiras simultaneamente, usamos uma sigmoide independente por rótulo.

8.4.6 Ativação de saída e tarefa

Tarefa	Saída típica	Ativação	Perda comum
Regressão real	1 ou mais valores	identidade	erro quadrático
Classificação binária	1 logit	sigmoide	entropia cruzada binária
Multiclasse exclusiva	K logits	softmax	entropia cruzada categórica
Multirrótulo	K logits	K sigmoides	entropia cruzada binária

Na prática, bibliotecas frequentemente combinam logits e entropia cruzada em uma única operação estável. Nesse caso, a função de perda espera logits crus e aplica internamente sigmoide ou softmax; aplicar a ativação antes pode duplicar a operação e prejudicar a estabilidade.

```
import numpy as np


def softmax_estavel(z):
    z = np.asarray(z, dtype=float)
    deslocado = z - np.max(z, axis=-1, keepdims=True)
    exp_z = np.exp(deslocado)
    return exp_z / np.sum(exp_z, axis=-1, keepdims=True)
```

8.4.6.1 Exercícios

1. Calcule ReLU, sigmoide e tanh para $z \in \{-2, 0, 2\}$.
2. Demonstre que $\sigma'(z) = \sigma(z)(1 - \sigma(z))$.
3. Prove que a softmax é invariante à soma de uma constante em todos os logits.
4. Explique por que softmax não é apropriada para uma imagem que pode conter simultaneamente “carro” e “pedestre”.
5. O que acontece com os gradientes de uma pilha de sigmoides quando as pré-ativações têm grande magnitude?

8.5 Rede neural de camada única

Antes de estudar redes largas, é útil separar duas arquiteturas que às vezes recebem nomes semelhantes.

- Um **único neurônio** calcula $g(w^\top x + b)$.
- Uma rede com **uma camada oculta** usa vários neurônios intermediários e depois combina suas ativações em uma saída.

Neste capítulo, "rede de camada única" significará uma rede com uma camada oculta, pois essa é a convenção usada nos teoremas de aproximação da seção seguinte.

8.5.1 Um único neurônio

Um neurônio com entrada $x \in \mathbb{R}^d$ produz

$$h(x) = g(w^\top x + b).$$

Escolhas diferentes de ativação e perda recuperam modelos já conhecidos:

Modelo	Ativação de saída	Predição	Perda típica
Regressão linear	identidade	valor real	erro quadrático
Perceptron	sinal	classe em $\{-1, +1\}$	erro de classificação
Regressão logística	sigmoide	probabilidade binária	entropia cruzada
Regressão softmax	softmax sobre K logits	probabilidades de K classes	entropia cruzada categórica

Todos esses modelos produzem fronteiras de decisão lineares no espaço de entrada. A ativação altera a saída e a perda apropriada, mas o nível $w^\top x + b = 0$ continua sendo um hiperplano. Um único neurônio não resolve, por exemplo, o XOR sem uma transformação prévia de atributos.

8.5.2 Uma camada oculta

Considere m neurônios ocultos, uma entrada com d atributos e uma saída escalar. A camada oculta calcula

$$a_j(x) = g(w_j^\top x + b_j), \qquad j = 1, \dots, m.$$

A saída combina essas novas características:

$$h_\theta(x) = g_{\text{out}}\left(c + \sum_{j=1}^{m} v_j a_j(x)\right). \tag{8.2}$$

Em notação matricial,

$$h_\theta(x) = g_{\text{out}}(v^\mathsf{T} g(Wx + b) + c),$$

com

$$W \in \mathbb{R}^{m\times d}, \quad b \in \mathbb{R}^m, \quad v \in \mathbb{R}^m, \quad c \in \mathbb{R}.$$

Cada neurônio oculto aprende uma característica $a_j(x)$. A camada de saída aprende como combiná-las. Diferentemente de uma transformação polinomial fixa, tanto as direções w_j quanto os deslocamentos b_j são ajustados pelos dados.

8.5.3 Contagem de parâmetros

A matriz W contém md pesos, o vetor b contém m vieses, a saída contém m pesos e um viés. Logo, a rede escalar em Equação 8.2 possui

$$md + m + m + 1 = m(d + 2) + 1$$

parâmetros treináveis. Para k saídas, usamos $V \in \mathbb{R}^{k\times m}$ e $c \in \mathbb{R}^k$, totalizando

$$md + m + km + k$$

parâmetros. Essa contagem ajuda a estimar memória, comparar arquiteturas e identificar erros de implementação.

8.5.4 Propagação para frente em lote

Adote a convenção de que cada linha da matriz $X \in \mathbb{R}^{N\times d}$ é um exemplo. Então

$$Z^{(1)} = XW^\mathsf{T} + \mathbf{1}b^\mathsf{T} \in \mathbb{R}^{N\times m},$$

$$A^{(1)} = g(Z^{(1)}),$$

$$Z^{(2)} = A^{(1)}V^{\mathsf{T}} + \mathbf{1}c^{\mathsf{T}} \in \mathbb{R}^{N\times k}.$$

O vetor de uns representa a transmissão do mesmo viés para todos os exemplos; bibliotecas fazem essa operação por *broadcasting*.

```
import numpy as np


def forward_uma_camada(X, W, b, V, c):
    Z1 = X @ W.T + b
    A1 = np.maximum(Z1, 0.0)
    Z2 = A1 @ V.T + c
    return Z1, A1, Z2
```

Retornar valores intermediários é útil no aprendizado: a retropropagação reutiliza $Z^{(1)}$ e $A^{(1)}$ para calcular as derivadas.

8.5.5 Treinamento

Os parâmetros não são ajustados neurônio por neurônio de forma independente. A perda na saída gera gradientes que atravessam a camada de saída e chegam a todos os neurônios ocultos. Como diferentes permutações dos neurônios podem representar a mesma função e a função de perda não é, em geral, convexa nos parâmetros de todas as camadas, a solução não possui a simplicidade da regressão linear por mínimos quadrados.

Mesmo assim, derivadas eficientes permitem treinar redes grandes por descida do gradiente e suas variantes. Inicialização, escala dos dados, taxa de aprendizado, tamanho do lote e regularização influenciam o resultado.

8.5.5.1 Exercícios

1. Conte os parâmetros de uma rede com $d = 10$, $m = 20$ e $k = 3$.
2. Verifique os formatos de todas as matrizes no cálculo em lote.
3. Explique por que trocar a ordem de dois neurônios ocultos e trocar as colunas correspondentes de V não altera a função da rede.

4. Por que aplicar a função sinal dentro da camada oculta dificulta o treinamento por gradientes?
5. Compare os atributos polinomiais fixos com as ativações aprendidas de uma camada oculta.

8.6 Teorema de aproximação universal para redes de camada única

Considere as funções produzidas por redes com uma camada oculta e saída linear:

$$g(x) = c + \sum_{j=1}^{m} v_j \, sigma(w_j^{\mathsf{T}} x + b_j). \tag{8.3}$$

Cada valor de m define uma classe com largura finita. O teorema estuda a união dessas classes quando permitimos que m seja tão grande quanto necessário.

8.6.1 Enunciado

Seja $K \subset \mathbb{R}^d$ compacto e seja $\sigma : \mathbb{R} \to \mathbb{R}$ uma função contínua. As combinações da forma Equação 8.3 são densas em $C(K)$, com a norma do supremo, se e somente se σ não é um polinômio.

Em termos concretos, para toda função contínua $f : K \to \mathbb{R}$ e todo $\varepsilon > 0$, existe uma largura finita m e existem parâmetros w_j, b_j, v_j, c tais que

$$\|f - g\|_\infty = \sup_{x \in K} |f(x) - g(x)| < \varepsilon. \tag{8.4}$$

Para uma saída vetorial $f : K \to \mathbb{R}^k$, podemos aproximar cada componente e reuni-las em uma camada de saída com k unidades. Escolhendo o erro de cada componente suficientemente pequeno, obtemos uma aproximação na norma vetorial desejada.

8.6.2 Como interpretar o teorema

A palavra **densa** significa que podemos chegar arbitrariamente perto de qualquer elemento de $C(K)$, não que uma única rede fixa represente todas as funções. A largura e os parâmetros dependem de f, de K e da tolerância ε.

A hipótese de compacidade também importa. Em $\mathbb{R}^d$, um conjunto compacto é fechado e limitado. A norma do supremo em Equação 8.4 exige que a aproximação seja boa simultaneamente em

todos os pontos desse domínio, uma garantia mais forte do que acertar somente uma amostra finita.

Ativações usuais como ReLU, sigmoide e tanh são não polinomiais e atendem à condição de expressividade. Se σ for um polinômio de grau q, cada termo $\sigma(w^{\mathsf{T}}x + b)$ tem grau no máximo q; somas finitas continuam com grau no máximo q. Essa classe de dimensão finita não pode aproximar uniformemente todas as funções contínuas.

8.6.3 Intuição com ReLU em uma dimensão

Uma combinação de ReLUs em uma variável,

$$g(x) = a + sx + \sum_{j=1}^{m} v_j \,\mathrm{ReLU}(x - t_j),$$

é linear por partes. Antes de t_j, o termo correspondente vale zero; depois de t_j, ele altera a inclinação em v_j. Portanto, colocando pontos de quebra t_j e escolhendo as mudanças de inclinação, podemos construir uma interpolação linear por partes.

Toda função contínua em um intervalo compacto é uniformemente contínua. Ao refinar suficientemente a malha, sua interpolação linear por partes fica uniformemente próxima da função. Isso fornece uma intuição concreta para a universalidade da ReLU em uma dimensão. Em dimensões maiores, a prova geral requer argumentos adicionais.

8.6.4 O que o teorema não garante

O resultado é de **existência**. Ele não afirma:

- qual largura m é necessária para uma função específica;
- que a rede será pequena ou computacionalmente eficiente;
- que descida do gradiente encontrará os parâmetros existentes;
- que uma amostra finita permitirá identificar a função;
- que a rede generalizará bem fora de K;
- que uma camada oculta é sempre a melhor arquitetura.

Uma rede larga pode representar a solução e ainda ser difícil de treinar. Além disso, aproximar qualquer função contínua não implica aproximar com poucos dados. Expressividade, otimização e generalização são questões distintas.

8.6.5 Largura versus profundidade

O teorema permite concentrar toda a complexidade em uma camada oculta, mas não diz que isso seja eficiente. Algumas funções possuem descrição compacta como composição de operações simples. Redes profundas refletem essa estrutura composicional e podem representar certas funções com muito menos unidades que uma rede rasa.

Assim, "uma camada é universal" não torna a profundidade inútil. A universalidade responde **se** uma aproximação existe; estudos de eficiência investigam **quantas** unidades e camadas são necessárias.

8.6.6 Exemplo computacional: função linear por partes

```
import numpy as np


def relu(x):
    return np.maximum(x, 0.0)


def rede_relu_1d(x, intercepto, inclinacao, pontos, saltos):
    """Representa uma função linear por partes usando ReLUs."""
    x = np.asarray(x, dtype=float)
    y = intercepto + inclinacao * x
    for t, delta in zip(pontos, saltos):
        y = y + delta * relu(x - t)
    return y
```

O vetor `saltos` não armazena as inclinações de cada trecho, mas a mudança de inclinação em cada ponto de quebra. Essa parametrização é uma representação explícita de uma rede ReLU com uma entrada e uma camada oculta.

8.6.6.1 Exercícios

1. Explique a diferença entre erro uniforme em K e erro somente nos pontos de treino.
2. Mostre que uma combinação finita de ativações polinomiais de grau q continua sendo um polinômio de grau no máximo q.
3. Represente $f(x) = |x|$ exatamente com duas unidades ReLU.
4. Construa uma função com inclinação 1 antes de zero e 3 depois de zero usando o código anterior.

5. Dê um exemplo de afirmação sobre treinamento que não decorre do teorema de aproximação universal.

8.7 Rede neural de múltiplas camadas

Uma rede profunda encadeia várias transformações aprendidas. Cada camada recebe uma representação, produz outra e a entrega à camada seguinte. Como as conexões apontam somente para a frente, o grafo de computação não possui ciclos.

8.7.1 Notação por camadas

Considere uma rede com L camadas parametrizadas. Definimos $a^{(0)} = x \in \mathbb{R}^{n_0}$ e, para $\ell = 1, \dots, L$,

$$z^{(\ell)} = W^{(\ell)} a^{(\ell-1)} + b^{(\ell)}, \qquad a^{(\ell)} = g_\ell\big(z^{(\ell)}\big). \tag{8.5}$$

Se a camada ℓ possui n_ℓ unidades, então

$$W^{(\ell)} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}, \qquad b^{(\ell)}, z^{(\ell)}, a^{(\ell)} \in \mathbb{R}^{n_\ell}.$$

A hipótese completa é

$$h_\theta = h^{(L)} \circ h^{(L-1)} \circ \cdots \circ h^{(1)},$$

onde $h^{(\ell)}(a) = g_\ell(W^{(\ell)} a + b^{(\ell)})$. A camada $h^{(1)}$ aparece à direita porque é aplicada primeiro. Inverter essa ordem é um erro frequente ao escrever composições.

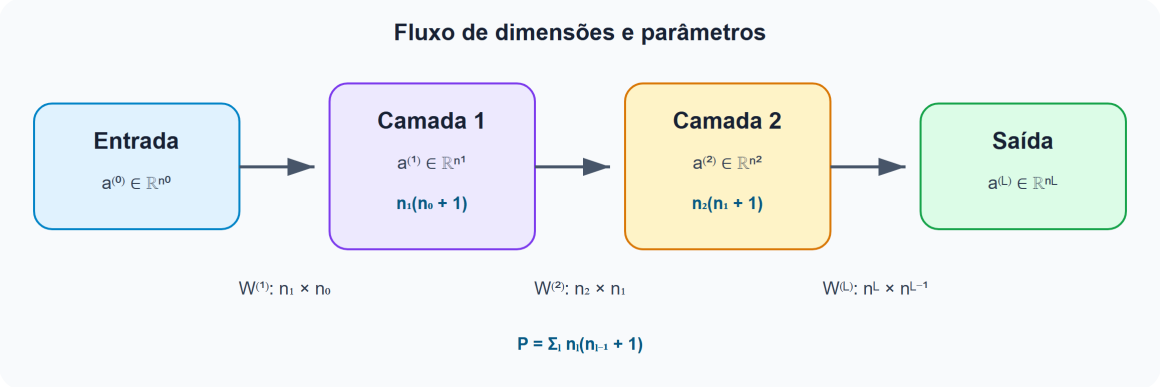


Figura 8.3: As larguras determinam os formatos das matrizes entre camadas consecutivas.

8.7.2 Camadas densas

Em uma camada **densa**, ou totalmente conectada, cada unidade recebe todas as ativações da camada anterior. Isso explica por que $W^{(\ell)}$ possui uma entrada para cada par formado por uma unidade da camada atual e uma unidade da camada anterior.

Nem toda rede usa conexões densas. Camadas convolucionais compartilham pesos e exploram estrutura espacial; mecanismos de atenção combinam representações segundo similaridades aprendidas; redes recorrentes compartilham parâmetros ao longo do tempo. A notação de composição ainda é útil, mas os operadores não precisam ser matrizes densas arbitrárias.

8.7.3 Ativações ocultas e ativação de saída

As ativações ocultas $g_1, \dots, g_{L-1}$ introduzem não linearidade. Elas não precisam ser iguais em todas as camadas, embora arquiteturas homogêneas sejam comuns. A ativação final g_L é escolhida conforme o significado desejado para a saída:

- identidade para regressão sem restrição de faixa;
- sigmoide para probabilidade binária ou rótulos independentes;
- softmax para classes mutuamente exclusivas;
- outra transformação específica quando a saída deve obedecer a uma restrição do problema.

Durante o treinamento, normalmente mantemos os valores antes da ativação final, chamados **logits**, porque perdas combinadas com logits oferecem maior estabilidade numérica.

8.7.4 Contagem de parâmetros

Uma camada densa de largura $n_{\ell-1}$ para n_ℓ possui

$$n_\ell n_{\ell-1} + n_\ell = n_\ell(n_{\ell-1} + 1)$$

parâmetros: uma matriz de pesos e um viés por unidade. Portanto, a rede inteira possui

$$P = \sum_{\ell=1}^{L} n_\ell(n_{\ell-1} + 1) \tag{8.6}$$

parâmetros treináveis. Por exemplo, a arquitetura $4 \to 8 \to 6 \to 3$ contém

$$8(4+1) + 6(8+1) + 3(6+1) = 115$$

parâmetros. A contagem inclui os vieses e não conta a camada de entrada como parametrizada.

8.7.5 Cálculo em lote

Quando cada linha representa um exemplo, $A^{(0)} = X \in \mathbb{R}^{N \times n_0}$. A propagação usa

$$Z^{(\ell)} = A^{(\ell-1)} (W^{(\ell)})^{\mathsf{T}} + b^{(\ell)},$$

em que o viés é transmitido para as N linhas por *broadcasting*. Assim, $Z^{(\ell)}$ e $A^{(\ell)}$ têm formato $N \times n_\ell$.

```
import numpy as np


def relu(z):
    return np.maximum(z, 0.0)


def forward_denso(X, pesos, vieses):
    A = np.asarray(X, dtype=float)
    cache = []

    for W, b in zip(pesos[:-1], vieses[:-1]):
        Z = A @ W.T + b
        A_novo = relu(Z)
        cache.append((A, Z))
        A = A_novo

    Z_saida = A @ pesos[-1].T + vieses[-1]
    cache.append((A, Z_saida))
    return Z_saida, cache
```

O exemplo usa ReLU nas camadas ocultas e deixa a última camada em logits. O `cache` guarda valores necessários à retropropagação. Em uma biblioteca de autodiferenciação, o grafo computacional cumpre esse papel.

8.7.6 Representações hierárquicas

Cada camada pode reutilizar características construídas pela anterior. Em uma tarefa visual, camadas iniciais podem responder a contrastes locais, enquanto camadas posteriores combinam essas respostas em padrões mais abstratos. Essa interpretação é útil, mas não significa que cada neurônio sempre tenha um significado humano simples.

A profundidade também muda a geometria da otimização. A função pode ser simples no espaço de saída e, ainda assim, possuir muitas parametrizações equivalentes. Gradientes que atravessam várias camadas podem desaparecer ou explodir; inicialização, normalização, conexões residuais e escolha da ativação ajudam a controlar esses efeitos.

> ⚠ Verifique os formatos antes dos valores
>
> Muitos erros de redes neurais são incompatibilidades silenciosas de eixos. Documente se exemplos ocupam linhas ou colunas e teste cada produto matricial. *Broadcasting* pode produzir uma matriz válida com um significado diferente do pretendido.

8.7.6.1 Exercícios

1. Liste os formatos de todos os tensores em uma rede $5 \to 10 \to 4 \to 2$ para um lote com 32 exemplos.
2. Calcule o número de parâmetros dessa rede usando
   5.
3. Mostre algebricamente que duas camadas sem ativação não linear podem ser substituídas por uma única camada afim.
4. Explique por que a camada de entrada não acrescenta parâmetros.
5. Modifique o pseudocódigo para aplicar softmax estável na saída apenas durante a inferência.

8.8 Teorema de aproximação universal para redes ReLU limitadas por largura

O teorema anterior permite usar uma única camada oculta, desde que sua largura possa crescer. Há uma possibilidade complementar: manter a largura pequena e permitir que a profundidade cresça. Para redes ReLU, essa troca ainda preserva a capacidade de aproximação universal em certos espaços de funções.

8.8.1 Enunciado em norma L^1

Seja $f : \mathbb{R}^n \to \mathbb{R}$ uma função integrável no sentido de Lebesgue, isto é,

$$\|f\|_1 = \int_{\mathbb{R}^n} |f(x)|\, dx < \infty.$$

Para todo $\varepsilon > 0$, existe uma rede feedforward totalmente conectada com ativação ReLU, largura máxima não superior a $n + 4$ e profundidade finita tal que a função F_θ representada pela rede satisfaz

$$\|f - F_\theta\|_1 = \int_{\mathbb{R}^n} |f(x) - F_\theta(x)|\, dx < \varepsilon. \tag{8.7}$$

A profundidade não é fixada previamente: ela pode aumentar quando a função-alvo se torna mais complexa ou quando ε diminui. Portanto, o resultado não afirma que uma arquitetura única de largura $n + 4$ e profundidade fixa aproxima todas as funções.

8.8.2 O significado da largura

A largura de uma rede é o maior número de unidades em uma de suas camadas ocultas. Se as larguras forem $n_1, \dots, n_{L-1}$, então

$$\text{largura}(\mathcal{N}) = \max_{\ell=1,\dots,L-1} n_\ell.$$

O limite $n + 4$ depende da dimensão de entrada n, e não do número de exemplos da amostra. Ele é um limite suficiente para o resultado acima, não uma afirmação de que toda função exige exatamente essa largura nem de que esse seja o melhor limite para toda variante do problema.

8.8.3 Comparação com a aproximação por uma camada oculta

Os dois resultados distribuem capacidade de formas diferentes:

Aspecto	Rede rasa	Rede de largura limitada
Profundidade	uma camada oculta	pode crescer
Largura	pode crescer	no máximo $n + 4$
Funções no enunciado	contínuas em compacto	integráveis em $\mathbb{R}^n$
Norma	uniforme L^∞	integral L^1
Tipo de garantia	existência	existência

As normas expressam exigências distintas. O erro uniforme controla o maior desvio:

$$\|f - g\|_\infty = \sup_{x \in K} |f(x) - g(x)|.$$

Já o erro L^1 acumula a área — ou o volume — da diferença. Uma aproximação pode ter erro muito grande em uma região de medida pequena e ainda apresentar erro L^1 reduzido. Por isso, os dois teoremas não devem ser comparados apenas pelo número de neurônios.

8.8.4 Intuição da construção

Uma demonstração construtiva pode ser entendida em quatro movimentos.

1. **Truncar as caudas.** Como $f \in L^1(\mathbb{R}^n)$, escolhemos um cubo grande fora do qual a integral de $|f|$ é pequena.
2. **Aproximar por blocos simples.** Dentro do cubo, funções simples, constantes em pequenas regiões, aproximam f em L^1.
3. **Construir um bloco por vez.** Camadas ReLU profundas formam aproximações das regiões e de seus valores, reutilizando poucas coordenadas auxiliares.
4. **Acumular os resultados.** A rede transporta a entrada e uma soma parcial enquanto acrescenta a contribuição do bloco seguinte.

A largura permanece limitada porque as mesmas unidades funcionam como um pequeno espaço de trabalho reutilizado ao longo da profundidade. A rede troca processamento “em paralelo”, característico de uma camada muito larga, por processamento “sequencial” em muitas camadas.

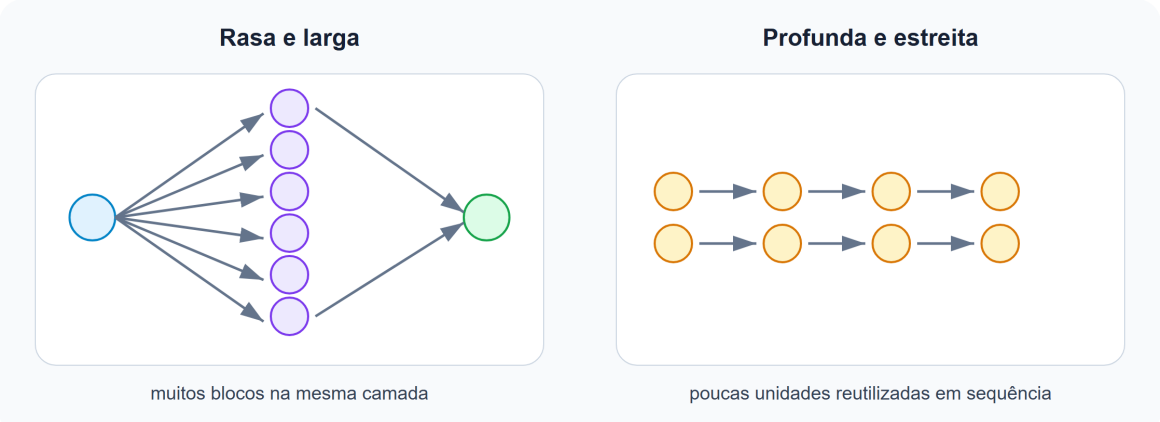


Figura 8.4: Uma rede larga calcula muitos blocos em paralelo; uma rede estreita e profunda reutiliza largura ao longo das camadas.

8.8.5 Por que funções indicadoras podem ser aproximadas?

A ReLU é contínua, portanto uma rede ReLU finita não representa exatamente a descontinuidade de uma função indicadora em todos os pontos. Em norma L^1, isso não é necessário. Podemos substituir a borda abrupta por uma transição estreita e linear por partes. Ao diminuir a espessura da região de transição, sua contribuição ao erro integral também diminui.

Essa observação evidencia a importância da norma. A mesma sequência não converge uniformemente para uma função indicadora em um domínio que contenha sua fronteira, pois o salto permanece descontínuo.

8.8.6 Extensão para várias saídas

Para $f : \mathbb{R}^n \to \mathbb{R}^k$, uma estratégia simples é aproximar cada componente. Contudo, o limite de largura precisa contabilizar as saídas e a arquitetura usada para transportá-las. Não devemos aplicar automaticamente o valor escalar $n + 4$ a qualquer teorema vetorial sem verificar seu enunciado específico.

8.8.7 O que a garantia não resolve

Assim como o teorema de uma camada oculta, Equação 8.7 é um resultado de representação. Ele não fornece:

- uma profundidade pequena;
- uma quantidade pequena de parâmetros;
- um algoritmo para encontrar os pesos;
- estabilidade numérica durante o treinamento;
- garantia de generalização a partir de uma amostra finita;
- boa aproximação ponto a ponto, pois a norma é L^1.

Uma construção existente pode ser profunda demais para a prática. Também pode usar pesos de grande magnitude, o que dificulta a otimização. A escolha de arquitetura continua sendo uma decisão estatística e computacional.

8.8.8 Verificação numérica da norma

Em uma dimensão, se conhecemos uma malha suficientemente fina, podemos aproximar a integral do erro pela regra do trapézio:

```
import numpy as np
```

```
def erro_l1_aproximado(f, g, inicio, fim, pontos=20_001):
    x = np.linspace(inicio, fim, pontos)
    erro = np.abs(f(x) - g(x))
    return np.trapezoid(erro, x)
```

Esse cálculo avalia apenas o intervalo escolhido e produz uma aproximação numérica. Para relacioná-lo à integral em toda a reta, também precisamos limitar ou estimar o erro nas caudas.

8.8.8.1 Exercícios

1. Diferencie largura máxima, profundidade e número total de parâmetros.
2. Explique por que uma função com erro alto em um intervalo muito estreito pode ter erro L^1 pequeno.
3. Mostre que a função indicadora de um intervalo não pode ser aproximada uniformemente por funções contínuas com erro menor que $1/2$ em um domínio que contenha suas extremidades.
4. Para $n = 20$, qual largura suficiente é fornecida pelo enunciado? O resultado determina a profundidade?
5. Compare a ideia de computação paralela de uma rede larga com a reutilização sequencial de unidades em uma rede profunda.

8.9 Largura versus profundidade das redes neurais

Os teoremas anteriores mostram que tanto a largura quanto a profundidade podem fornecer capacidade de aproximação. Eles não dizem, porém, que duas arquiteturas com o mesmo número de parâmetros sejam igualmente eficientes para representar uma função particular.

Largura oferece muitas unidades que operam em paralelo sobre a mesma representação. **Profundidade** permite compor transformações e reutilizar resultados intermediários. Essa diferença é análoga à comparação entre uma expressão expandida e um programa que cria valores auxiliares e os reaproveita.

8.9.1 Composição e regiões lineares

Redes ReLU representam funções lineares por partes. Em uma dimensão, uma camada com várias ReLUs pode introduzir vários pontos de quebra. Ao compor camadas, os pontos de quebra de uma transformação podem ser replicados nas regiões criadas anteriormente, fazendo o número de trechos crescer rapidamente.

Considere a função triangular em $[0, 1]$:

$$T(x) = 2\,\mathrm{ReLU}(x) - 4\,\mathrm{ReLU}\left(x - \frac{1}{2}\right) + 2\,\mathrm{ReLU}(x - 1).$$

Ela sobe de 0 a 1 e depois desce a 0. A composição $T \circ T$ produz mais oscilações; repetir a composição aumenta seu número sem precisar reescrever cada trecho independentemente.

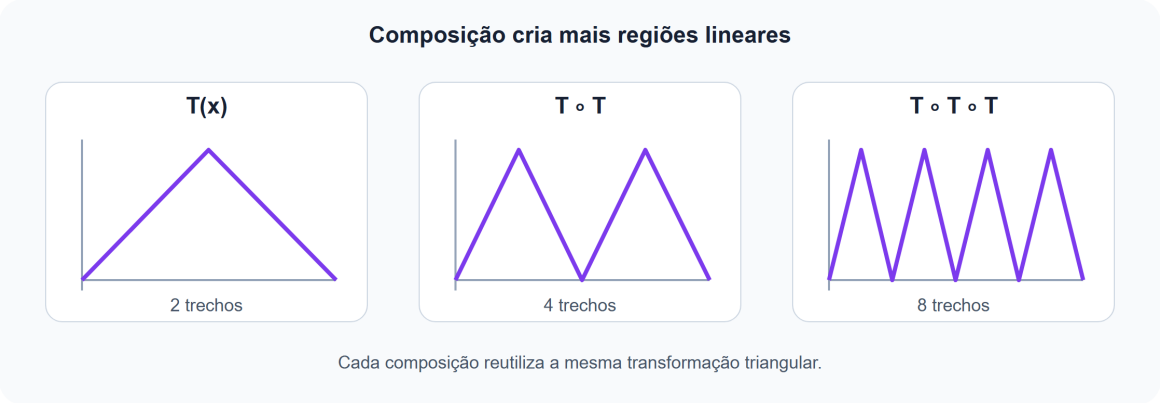


Figura 8.5: A composição de uma função triangular cria progressivamente mais regiões lineares.

Resultados de separação de profundidade formalizam essa intuição: há famílias de funções que redes profundas representam com quantidade moderada de unidades, mas cuja aproximação por redes com profundidade limitada exige largura muito maior, em alguns enunciados de modo exponencial. Toda afirmação desse tipo depende das hipóteses — ativação, norma, erro permitido, domínio e limites impostos aos pesos — e não deve ser convertida em uma regra universal sobre qualquer conjunto de dados.

8.9.2 Quando a largura ajuda

Camadas mais largas podem:

- aprender várias características no mesmo nível de abstração;
- reduzir gargalos que descartariam informação cedo demais;
- aumentar o paralelismo disponível em aceleradores;
- oferecer caminhos curtos entre entrada e saída.

Largura excessiva também aumenta memória, produtos matriciais e risco de redundância. Mais parâmetros não substituem dados, regularização ou uma função objetivo apropriada.

8.9.3 Quando a profundidade ajuda

Profundidade é especialmente natural quando a tarefa possui estrutura composicional: pixels formam padrões locais, padrões formam partes e partes formam objetos; em linguagem, símbolos formam expressões e expressões participam de contextos maiores.

Entretanto, redes profundas possuem caminhos de derivação longos. Isso pode causar gradientes que desaparecem ou explodem, elevar a latência sequencial e tornar a otimização mais sensível. Conexões residuais, normalização e inicialização adequada foram desenvolvidas, em parte, para tornar redes profundas treináveis.

8.9.4 Gargalos e conexões residuais

Uma camada muito estreita pode funcionar como gargalo. Isso é útil se queremos compressão, mas prejudicial quando elimina informação necessária às camadas seguintes. Uma conexão residual calcula

$$a^{(\ell+1)} = a^{(\ell)} + F_\ell(a^{(\ell)}),$$

desde que os formatos sejam compatíveis. A parcela identidade oferece um caminho direto para informação e gradientes, permitindo que o bloco aprenda uma correção em vez de reconstruir toda a representação.

8.9.5 Escolha orientada por validação

Não existe uma proporção universal entre largura e profundidade. Uma comparação justa deve controlar, quando possível:

1. número total de parâmetros ou custo computacional;
2. procedimento de treinamento e orçamento de épocas;
3. regularização e transformação dos dados;
4. métrica em validação, latência e uso de memória;
5. variabilidade entre inicializações aleatórias.

Uma arquitetura menor que atinge a métrica necessária com treinamento estável costuma ser preferível. Teoremas de expressividade ajudam a entender possibilidades e limitações, mas a escolha final combina teoria com evidência experimental.

```
def contar_parametros_densos(larguras):
    return sum(
        saida * (entrada + 1)
```

```
        for entrada, saida in zip(larguras[:-1], larguras[1:])
    )


print(contar_parametros_densos([20, 64, 10]))
print(contar_parametros_densos([20, 24, 24, 10]))
```

Comparar apenas a quantidade de camadas seria enganoso: o código permite verificar também o orçamento de parâmetros das arquiteturas candidatas.

8.9.5.1 Exercícios

1. Esboce $T(x)$ e $T(T(x))$ no intervalo $[0, 1]$.
2. Explique por que composição pode ser mais econômica que enumerar diretamente todos os trechos de uma função.
3. Calcule os parâmetros das duas arquiteturas do exemplo em Python.
4. Dê um exemplo de tarefa com estrutura naturalmente composicional.
5. Por que um resultado de separação teórica não garante que uma rede mais profunda terá melhor acurácia em qualquer base real?

8.10 Descida do Gradiente

Treinar uma rede significa escolher seus parâmetros $\theta = \{W^{(\ell)}, b^{(\ell)}\}_{\ell=1}^{L}$ para reduzir uma função objetivo. Como redes profundas compõem muitas operações não lineares, essa função raramente possui uma solução fechada. Usamos então métodos iterativos baseados em derivadas.

8.10.1 Função objetivo

Para uma amostra $D = \{(x_i, y_i)\}_{i=1}^{N}$, definimos

$$E_D(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(h_\theta(x_i), y_i) + \lambda R(\theta), \tag{8.8}$$

onde ℓ é a perda por exemplo, R é um termo opcional de regularização e $\lambda \geq 0$ controla sua força. Alguns exemplos são:

- erro quadrático em regressão;
- entropia cruzada binária para dois resultados;
- entropia cruzada categórica para classes exclusivas.

A escolha deve combinar o significado da saída, o modelo probabilístico e a estabilidade numérica. A acurácia, apesar de útil para avaliação, é descontínua e não costuma servir diretamente como objetivo de gradiente.

8.10.2 Direção de maior descida

Para uma pequena perturbação $\Delta\theta$, a aproximação de primeira ordem é

$$E_D(\theta + \Delta\theta) \approx E_D(\theta) + \nabla E_D(\theta)^\mathsf{T} \Delta\theta.$$

Entre direções unitárias, o gradiente aponta para o maior crescimento local e seu oposto aponta para o maior decréscimo. A **descida do gradiente** atualiza

$$\theta_{t+1} = \theta_t - \eta_t \nabla E_D(\theta_t), \tag{8.9}$$

com taxa de aprendizado $\eta_t > 0$. O nome "máximo declive" descreve a direção negativa do gradiente, mas "descida do gradiente" é a terminologia mais comum.

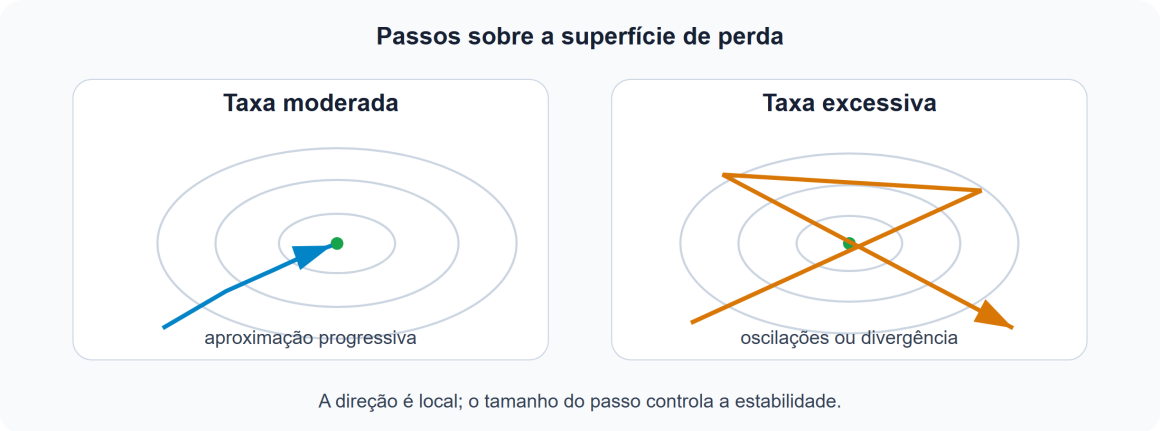


Figura 8.6: A direção negativa do gradiente reduz localmente a função objetivo; taxas diferentes produzem passos de tamanhos diferentes.

Não normalizamos o gradiente na atualização padrão. Sua magnitude contém informação sobre a inclinação local. Métodos com gradiente normalizado existem, mas constituem outro algoritmo e mudam a interpretação de η_t.

8.10.3 Lote completo, estocástico e mini-lote

Calcular Equação 8.8 sobre todos os exemplos fornece o gradiente de **lote completo**. Para grandes bases, dividimos os dados em mini-lotes B_t e usamos

$$g_t = \frac{1}{|B_t|} \sum_{i \in B_t} \nabla_\theta \ell(h_\theta(x_i), y_i) + \lambda \nabla_\theta R(\theta).$$

A atualização torna-se $\theta_{t+1} = \theta_t - \eta_t g_t$. Um lote com um exemplo caracteriza a descida estocástica; lotes intermediários aproveitam vetorização e introduzem ruído que pode ajudar a exploração.

Uma **época** é uma passagem pelos exemplos do treino. Uma **iteração** é uma atualização. Com N exemplos e lotes de tamanho B, uma época contém aproximadamente $\lceil N/B \rceil$ iterações.

8.10.4 Taxa de aprendizado

Se η for muito pequena, o treinamento avança lentamente. Se for muito grande, a perda pode oscilar, divergir ou produzir valores não finitos. Não há um valor como $0{,}1$ que seja adequado para todas as redes.

Podemos usar taxa constante, decaimento programado ou adaptação baseada no histórico dos gradientes. *Momentum* acumula uma média das direções; Adam adapta escalas por parâmetro. Esses métodos facilitam o treinamento, mas ainda exigem monitoramento e não garantem o mínimo global.

8.10.5 Inicialização e quebra de simetria

Inicializar todos os pesos ocultos em zero faz neurônios da mesma camada receberem gradientes iguais. Eles permanecem idênticos e desperdiçam a largura disponível. Usamos valores aleatórios com variância relacionada ao número de entradas, como inicializações de Xavier ou He, dependendo da ativação.

Vieses podem frequentemente começar em zero porque pesos aleatórios já quebram a simetria. A semente aleatória deve ser registrada para tornar experimentos reproduzíveis.

8.10.6 Critérios de parada

Gradiente exatamente zero não é um critério realista e pode indicar mínimo, máximo, ponto de sela ou saturação. Em prática, combinamos:

- limite máximo de épocas ou tempo;
- perda e métricas em treino e validação;
- ausência de melhora na validação por certo número de épocas;
- norma do gradiente ou variação dos parâmetros;
- verificação de valores NaN ou infinitos.

A **parada antecipada** guarda os parâmetros com melhor validação, não necessariamente os da última época. O conjunto de teste permanece reservado até o fim.

8.10.7 Laço mínimo de otimização

```
def treinar(modelo, lotes, otimizador, epocas):
    for epoca in range(epocas):
        modelo.modo_treino()

        for X, y in lotes:
            otimizador.zerar_gradientes()
            logits = modelo(X)
            perda = funcao_perda(logits, y)
            perda.retropropagar()
            otimizador.atualizar()

        avaliar_na_validacao(modelo)
```

Zerar gradientes é explícito porque muitas bibliotecas os acumulam por padrão. A retropropagação calcula derivadas; o otimizador decide como transformá-las em atualizações.

> **i** Minimização não significa gradiente sempre menor
>
> Em mini-lotes, a perda do lote seguinte pode aumentar mesmo quando a tendência global melhora. Além disso, a norma do gradiente pode crescer temporariamente. Curvas suavizadas e métricas de validação são mais informativas que uma única iteração.

8.10.7.1 Exercícios

1. Derive Equação 8.9 a partir da aproximação de primeira ordem e da escolha $\Delta\theta = -\eta\nabla E_D$.
2. Quantas atualizações há por época para $N = 10\,000$ e $B = 128$?
3. Explique por que pesos ocultos todos iguais preservam simetria.
4. Descreva sinais observáveis de uma taxa de aprendizado excessiva.
5. Diferencie perda de treino, perda de validação e métrica de teste.

8.11 Vetorização

Vetorização consiste em expressar várias operações escalares como uma operação sobre vetores ou matrizes. Além de tornar a notação compacta, ela permite que bibliotecas usem rotinas altamente otimizadas e executem muitas multiplicações em paralelo em CPU, GPU ou outros aceleradores.

8.11.1 Convenção de eixos

Adotaremos uma regra única:

- cada **linha** contém um exemplo;
- cada **coluna** contém um atributo ou uma unidade;
- a matriz de pesos armazena um neurônio por linha.

Assim, para uma camada que recebe d valores e produz m ativações,

$$W \in \mathbb{R}^{m \times d}, \qquad b \in \mathbb{R}^m.$$

Para um único vetor-coluna $x \in \mathbb{R}^d$, usamos

$$z = Wx + b \in \mathbb{R}^m.$$

Já no código em lote, os exemplos ocupam linhas, e a transposta de W aparece à direita.

8.11.2 Uma camada sobre um lote

Se $X \in \mathbb{R}^{B \times d}$ representa um mini-lote com B exemplos, então

$$Z = XW^{\mathsf{T}} + \mathbf{1}_B b^{\mathsf{T}} \in \mathbb{R}^{B \times m}, \tag{8.10}$$

e

$$A = g(Z) \in \mathbb{R}^{B \times m}.$$

A entrada Z_{ij} é a pré-ativação da unidade j para o exemplo i. A ativação é aplicada elemento a elemento, salvo funções vetoriais como softmax, que operam ao longo do eixo das classes.

Figura 8.7: Formatos de uma camada densa vetorizada com exemplos organizados nas linhas.

8.11.3 Broadcasting do viés

Não precisamos construir materialmente a matriz $\mathbf{1}_B b^\top$. Ao somar um vetor de formato `(m,)` a uma matriz `(B, m)`, NumPy e bibliotecas semelhantes repetem logicamente o vetor em cada linha. Esse mecanismo é chamado *broadcasting*.

Broadcasting economiza memória, mas pode esconder erros. Um viés com formato `(B, 1)` também pode ser somado a `Z`, porém acrescentaria um valor por exemplo, não um valor por unidade. Os formatos seriam compatíveis e o significado estaria errado.

8.11.4 Várias camadas

Para uma arquitetura $n_0 \to n_1 \to \cdots \to n_L$ e um lote com B exemplos,

$$A^{(0)} = X \in \mathbb{R}^{B \times n_0},$$

$$Z^{(\ell)} = A^{(\ell-1)} (W^{(\ell)})^\top + b^{(\ell)}, \qquad A^{(\ell)} = g_\ell(Z^{(\ell)}).$$

Em cada camada, somente a segunda dimensão muda: de $n_{\ell-1}$ para n_ℓ. A dimensão B do lote é preservada.

```
import numpy as np


def camada_densa(A, W, b, ativacao):
    A = np.asarray(A)
```

```
    W = np.asarray(W)
    b = np.asarray(b)

    if A.ndim != 2 or W.ndim != 2 or b.ndim != 1:
        raise ValueError("A e W devem ser matrizes; b deve ser vetor")
    if A.shape[1] != W.shape[1] or W.shape[0] != b.shape[0]:
        raise ValueError("dimensões incompatíveis na camada densa")

    Z = A @ W.T + b
    return ativacao(Z), Z
```

As verificações tornam a convenção explícita. Em código de produção, testes de unidade com lotes pequenos ajudam a confirmar valores, não apenas formatos.

8.11.5 Por que é mais rápido?

Um laço Python que calcula cada neurônio e cada exemplo separadamente tem grande custo de interpretação. A multiplicação matricial delega o trabalho a implementações compiladas, usa instruções vetoriais, memória em blocos e paralelismo. A quantidade matemática de multiplicações não desaparece, mas a execução utiliza melhor o hardware.

Vetorização também possui limites. Um lote muito grande pode exceder a memória, e armazenar todas as ativações é caro durante o treinamento. Mini-lotes equilibram paralelismo, memória e frequência de atualização dos parâmetros.

8.11.6 Vetorização da perda

Para regressão com k saídas, se $\widehat{Y}, Y \in \mathbb{R}^{B\times k}$, o erro quadrático médio pode ser escrito como

$$E_B = \frac{1}{Bk}\|\widehat{Y} - Y\|_F^2,$$

onde $\|\cdot\|_F$ é a norma de Frobenius. É preciso declarar se a redução usa soma ou média e sobre quais eixos; isso altera a escala do gradiente e, consequentemente, a taxa de aprendizado adequada.

8.11.6.1 Exercícios

1. Determine o formato de Z para $B = 64$, $d = 100$ e $m = 32$.
2. Escreva Equação 8.10 usando exemplos como colunas e compare onde aparece a transposta.
3. Explique o erro semântico de somar um vetor `(B, 1)` a `Z`.

4. Por que lotes maiores não reduzem a quantidade matemática de multiplicações, embora possam reduzir o tempo de execução?
5. Compare o gradiente de uma perda somada com o da mesma perda média.

8.12 Autodiferenciação reversa pela regra da cadeia

Para atualizar milhões de parâmetros, precisamos calcular muitas derivadas de uma única saída escalar: a perda. O **modo reverso da autodiferenciação** é adequado a essa estrutura. Ele executa o grafo para frente, guarda valores intermediários e percorre as operações na ordem inversa, aplicando a regra da cadeia.

Autodiferenciação não é diferenciação simbólica nem aproximação por diferenças finitas. Ela aplica regras exatas de derivação às operações elementares realizadas pelo programa, sujeita apenas ao arredondamento numérico.

8.12.1 Um grafo computacional simples

Considere

$$z = wx + b, \qquad a = g(z), \qquad E = \ell(a, y).$$

A passagem para frente calcula z, a e E. No sentido inverso,

$$\frac{\partial E}{\partial z} = \frac{\partial E}{\partial a}\frac{\partial a}{\partial z},$$

e depois

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial z}\frac{\partial z}{\partial w}, \qquad \frac{\partial E}{\partial b} = \frac{\partial E}{\partial z}\frac{\partial z}{\partial b}.$$

Como $\partial z/\partial w = x$ e $\partial z/\partial b = 1$, obtemos

$$\frac{\partial E}{\partial w} = \delta x, \qquad \frac{\partial E}{\partial b} = \delta, \qquad \delta := \frac{\partial E}{\partial z}.$$

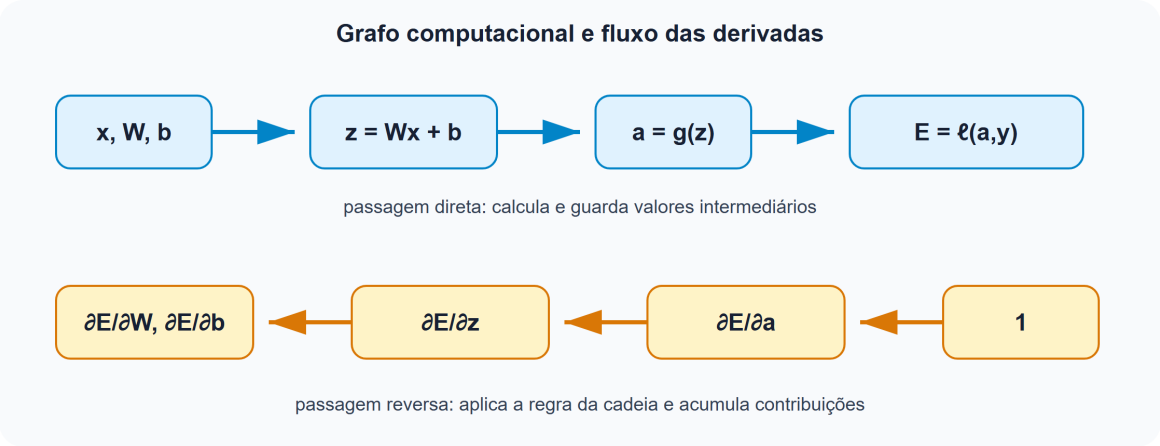


Figura 8.8: Na passagem reversa, a derivada da perda flui pelas operações na ordem contrária e cada nó usa valores guardados na passagem direta.

8.12.2 Produtos vetor–Jacobiano

Uma camada pode produzir um vetor. Se $u = f(v)$ e a perda escalar é $E(u)$, a regra da cadeia pode ser escrita como

$$\nabla_v E = J_f(v)^{\mathsf{T}} \nabla_u E,$$

onde J_f é o Jacobiano de f. O modo reverso não precisa armazenar todo esse Jacobiano: calcula diretamente o produto $J_f(v)^{\mathsf{T}} \nabla_u E$, chamado produto vetor–Jacobiano.

Isso é crucial em redes. Formar explicitamente cada Jacobiano seria caro, enquanto propagar um vetor de derivadas possui custo da mesma ordem da passagem para frente.

8.12.3 Derivação para uma camada densa

Para um exemplo, a camada ℓ calcula

$$z^{(\ell)} = W^{(\ell)} a^{(\ell-1)} + b^{(\ell)}, \qquad a^{(\ell)} = g_\ell(z^{(\ell)}).$$

Defina o sinal retropropagado

$$\delta^{(\ell)} := \frac{\partial E}{\partial z^{(\ell)}}. \tag{8.11}$$

Usando produtos externos, os gradientes dos parâmetros são

$$\frac{\partial E}{\partial W^{(\ell)}} = \delta^{(\ell)} (a^{(\ell-1)})^{\mathsf{T}}, \tag{8.12}$$

$$\frac{\partial E}{\partial b^{(\ell)}} = \delta^{(\ell)}. \tag{8.13}$$

Os formatos confirmam a expressão:

$$(n_\ell \times 1)(1 \times n_{\ell-1}) = n_\ell \times n_{\ell-1}.$$

8.12.4 Última camada

A forma de $\delta^{(L)}$ depende da ativação e da perda. Em geral,

$$\delta^{(L)} = \nabla_{a^{(L)}} E \odot g_L'(z^{(L)}),$$

onde $\odot$ denota produto elemento a elemento.

Para erro quadrático $E = \frac{1}{2}\|a^{(L)} - y\|^2$,

$$\delta^{(L)} = (a^{(L)} - y) \odot g_L'(z^{(L)}).$$

Para softmax combinada com entropia cruzada categórica, a derivação simplifica para

$$\delta^{(L)} = p - y,$$

em que $p = \text{softmax}(z^{(L)})$ e y é o vetor alvo. Essa fórmula pressupõe a combinação específica de ativação e perda; não deve ser reutilizada indiscriminadamente.

8.12.5 Camadas ocultas

A pré-ativação da camada seguinte depende de $a^{(\ell)}$:

$$z^{(\ell+1)} = W^{(\ell+1)} a^{(\ell)} + b^{(\ell+1)}.$$

Aplicando a regra da cadeia,

$$\delta^{(\ell)} = \big((W^{(\ell+1)})^{\mathsf{T}}\delta^{(\ell+1)}\big) \odot g'_{\ell}(z^{(\ell)}). \tag{8.14}$$

A matriz transposta redistribui para cada unidade da camada atual as contribuições de todas as unidades da camada seguinte. Em seguida, o produto por g'_{ℓ} atravessa a ativação local.

A recorrência começa em $\delta^{(L)}$ e segue até a primeira camada. Depois de obter cada $\delta^{(\ell)}$, usamos Equação 8.12 e Equação 8.13.

8.12.6 Forma vetorizada para mini-lotes

Com exemplos nas linhas, sejam $A^{(\ell-1)} \in \mathbb{R}^{B\times n_{\ell-1}}$ e $\Delta^{(\ell)} \in \mathbb{R}^{B\times n_{\ell}}$. Se a perda é a média do lote,

$$\frac{\partial E}{\partial W^{(\ell)}} = \frac{1}{B}(\Delta^{(\ell)})^{\mathsf{T}} A^{(\ell-1)},$$

$$\frac{\partial E}{\partial b^{(\ell)}} = \frac{1}{B}\sum_{i=1}^{B} \Delta^{(\ell)}_{i,:},$$

e a propagação para a camada anterior é

$$\Delta^{(\ell-1)} = \big(\Delta^{(\ell)} W^{(\ell)}\big) \odot g'_{\ell-1}(Z^{(\ell-1)}).$$

O eixo do somatório do viés é o dos exemplos. Somar sobre as unidades produziria um vetor com formato e significado errados.

8.12.7 Ramificações no grafo

Quando um valor alimenta dois caminhos, a perda depende dele por ambos. A derivada total é a soma das contribuições. Se u influencia E através de $r_1(u)$ e $r_2(u)$, então

$$\frac{\partial E}{\partial u} = \frac{\partial E}{\partial u}\bigg|_{r_1} + \frac{\partial E}{\partial u}\bigg|_{r_2}.$$

Isso explica por que bibliotecas acumulam gradientes e por que conexões residuais somam caminhos na passagem reversa.

8.12.8 Verificação por diferenças finitas

Diferenças centrais oferecem uma checagem de implementação:

$$\frac{\partial E}{\partial \theta_j} \approx \frac{E(\theta_j + h) - E(\theta_j - h)}{2h}.$$

```
def derivada_numerica(perda, theta, indice, h=1e-5):
    original = theta[indice]

    theta[indice] = original + h
    mais = perda()

    theta[indice] = original - h
    menos = perda()

    theta[indice] = original
    return (mais - menos) / (2 * h)
```

Esse método requer duas avaliações por parâmetro, sofre com escolha de h e arredondamento, e não é adequado para treinamento. Ele serve como teste em redes pequenas, usando precisão dupla e evitando pontos não diferenciáveis da ReLU.

> ⚠ Gradiente acumulado não é atualização
>
> A retropropagação calcula derivadas. Ela não altera parâmetros por si só. O otimizador usa essas derivadas posteriormente. Misturar os dois papéis dificulta testar e reutilizar a implementação.

8.12.8.1 Exercícios

1. Derive os gradientes de $z = wx + b$ sem omitir as etapas da regra da cadeia.
2. Verifique os formatos em Equação 8.14 para uma camada $n_{\ell-1} \to n_\ell \to n_{\ell+1}$.
3. Explique por que gradientes de dois caminhos devem ser somados.
4. Derive $\delta^{(L)}$ para saída identidade e erro quadrático.
5. Compare custo e finalidade da autodiferenciação reversa e das diferenças finitas.

8.13 Algoritmo Propagação para Trás

Retropropagação, ou *backpropagation*, é a aplicação especializada do modo reverso da autodiferenciação a uma rede neural. Seu resultado são os gradientes da perda em relação a todos os parâmetros. Descida do gradiente, Adam ou outro otimizador usa esses gradientes em uma etapa posterior.

8.13.1 Visão geral

Para um mini-lote, uma iteração de treinamento possui quatro fases:

1. **Propagação direta:** calcular $Z^{(\ell)}$ e $A^{(\ell)}$ da primeira à última camada e guardar os valores necessários.
2. **Perda:** calcular E a partir da saída e dos alvos.
3. **Propagação reversa:** iniciar $\Delta^{(L)}$ na saída e percorrer as camadas de L até 1, calculando gradientes.
4. **Atualização:** aplicar o otimizador aos parâmetros.

Para descida do gradiente simples, a atualização correta é

$$W_{t+1}^{(\ell)} = W_t^{(\ell)} - \eta_t \frac{\partial E}{\partial W_t^{(\ell)}},$$

$$b_{t+1}^{(\ell)} = b_t^{(\ell)} - \eta_t \frac{\partial E}{\partial b_t^{(\ell)}}.$$

O parâmetro antigo, no instante t, aparece no lado direito. Usar o índice $t+1$ nos dois lados tornaria a regra circular.

8.13.2 Algoritmo vetorizado

Com exemplos nas linhas e uma perda média, a propagação reversa executa, para $\ell = L, L-1, \dots, 1$,

$$dW^{(\ell)} = \frac{1}{B}(\Delta^{(\ell)})^{\mathsf{T}} A^{(\ell-1)},$$

$$db^{(\ell)} = \frac{1}{B}\sum_{i=1}^{B} \Delta_{i,:}^{(\ell)},$$

e, se $\ell > 1$,

$$\Delta^{(\ell-1)} = \big(\Delta^{(\ell)} W^{(\ell)}\big) \odot g'_{\ell-1}(Z^{(\ell-1)}).$$

O fator $1/B$ deve aparecer uma única vez. Podemos incorporá-lo no delta inicial ou nos gradientes dos parâmetros, mas duplicá-lo reduz o gradiente por B^2 e omiti-lo faz sua escala crescer com o lote.

8.13.3 Exemplo completo com duas camadas

Considere ReLU na camada oculta, saída identidade e

$$E = \frac{1}{2B} \sum_{i=1}^{B} \|\hat{y}_i - y_i\|^2.$$

```
import numpy as np


def forward(X, W1, b1, W2, b2):
    Z1 = X @ W1.T + b1
    A1 = np.maximum(Z1, 0.0)
    Y_hat = A1 @ W2.T + b2
    cache = (X, Z1, A1)
    return Y_hat, cache


def backward(Y_hat, Y, cache, W2):
    X, Z1, A1 = cache
    B = X.shape[0]

    # Delta da saída; a média 1/B entra aqui.
    D2 = (Y_hat - Y) / B
    dW2 = D2.T @ A1
    db2 = D2.sum(axis=0)

    D1 = (D2 @ W2) * (Z1 > 0)
    dW1 = D1.T @ X
    db1 = D1.sum(axis=0)
    return dW1, db1, dW2, db2
```

```
def atualizar(parametros, gradientes, taxa):
    for nome, gradiente in zip(parametros, gradientes):
        nome -= taxa * gradiente
```

Se `Y_hat` e `Y` tiverem formato `(B, k)`, então `D2` também terá `(B, k)`. A função pressupõe que todas as matrizes são de ponto flutuante e que a atualização no lugar é desejada.

8.13.4 Cache e memória

Para calcular $g'_\ell(Z^{(\ell)})$ e os produtos com $A^{(\ell-1)}$, guardamos ativações da passagem direta. Em redes muito profundas, esse armazenamento domina a memória de treinamento.

Uma alternativa é o *checkpointing*: guardar apenas algumas ativações e recalcular outras durante a passagem reversa. Isso troca tempo de computação por memória. Na inferência, não precisamos do grafo reverso e podemos liberar intermediários mais cedo.

8.13.5 Ordem segura da atualização

Todos os gradientes devem ser calculados com os mesmos valores dos parâmetros da passagem direta. Atualizar $W^{(L)}$ antes de calcular $\Delta^{(L-1)}$ faria a recorrência usar um peso novo, inconsistente com o grafo que produziu a perda. Primeiro calculamos todos os gradientes; somente depois o otimizador altera os parâmetros.

8.13.6 Testes recomendados

Uma implementação própria deve ser verificada progressivamente:

1. testar cada camada direta com valores pequenos conhecidos;
2. conferir formatos de ativações e gradientes;
3. comparar gradientes a diferenças finitas em uma rede minúscula;
4. confirmar que um passo suficientemente pequeno reduz a perda do mesmo lote;
5. tentar sobreajustar deliberadamente um conjunto com poucos exemplos;
6. verificar valores não finitos e normas de gradiente durante o treino.

Sobreajustar um lote pequeno não comprova generalização, mas é um teste útil do encadeamento entre modelo, perda, retropropagação e atualização.

> **!** Retropropagação reutiliza cálculos
>
> Calcular cada derivada parcial do zero repetiria grande parte do grafo. A eficiência do algoritmo vem de armazenar derivadas intermediárias e reutilizá-las para todos os parâmetros anteriores.

8.13.6.1 Exercícios

1. Verifique os formatos de `D1`, `dW1`, `db1`, `D2`, `dW2` e `db2`.
2. Derive o delta da saída usado no exemplo a partir do erro quadrático.
3. Explique por que os pesos só devem ser atualizados após todos os gradientes terem sido calculados.
4. O que muda nas fórmulas se a perda somar os exemplos em vez de usar a média?
5. Descreva como o *checkpointing* troca memória por computação.

8.13.7 Síntese do capítulo

Redes neurais compõem transformações afins e ativações não lineares para aprender representações. Largura e profundidade oferecem formas complementares de expressividade, mas não garantem otimização ou generalização. O treinamento combina operações vetorizadas, uma perda diferenciável, autodiferenciação reversa e um otimizador. Acompanhar formatos, escalas e separação entre essas etapas torna a implementação mais correta e mais fácil de depurar.

8.14 Exercícios de múltipla escolha

Em cada questão, assinale a única alternativa correta. Use os formatos das matrizes para conferir as respostas que envolvem vetorização.

1. Se todas as ativações de uma rede densa forem funções lineares, a composição de várias camadas será equivalente a:

   a. uma única transformação afim.

   b. uma árvore de decisão.

   c. um núcleo RBF.

   d. uma função necessariamente periódica.

2. A função ReLU é definida por:

a. $\max(0, z)$.

b. $1/(1 + e^{-z})$.

c. $\exp(z)/\sum_j \exp(z_j)$.

d. z^2.

3. Para classificação multiclasse com classes mutuamente exclusivas, uma ativação de saída usual é:

a. ReLU aplicada a um único número.

b. softmax.

c. valor absoluto.

d. identidade seguida de arredondamento obrigatório.

4. Uma camada densa com 5 entradas e 3 neurônios possui quantos parâmetros, incluindo um viés por neurônio?

a. 8.
b. 15.
c. 18.
d. 24.

5. O teorema de aproximação universal afirma, sob suas condições, que uma rede:

a. pode aproximar funções de uma classe-alvo com precisão arbitrária.

b. sempre encontra automaticamente os melhores pesos.

c. generaliza perfeitamente com um exemplo.

d. requer exatamente duas unidades ocultas.

6. Em uma rede ReLU, aumentar a profundidade pode ser especialmente útil para:

a. representar composições hierárquicas.

b. eliminar todos os hiperparâmetros.

c. dispensar ativações não lineares.

d. garantir convexidade da perda nos pesos.

7. Na descida do gradiente, a atualização padrão é:

a. $\theta \leftarrow \theta + \eta \nabla J(\theta)$.

b. $\theta \leftarrow \theta - \eta \nabla J(\theta)$.

c. $\theta \leftarrow \nabla J(\theta)/0$.

d. $\theta \leftarrow \theta^2$.

8. Para um lote $X \in \mathbb{R}^{N \times d}$ e pesos $W \in \mathbb{R}^{d \times m}$, a saída pré-ativação $Z = XW + b$ possui formato:

a. $d \times d$.

b. $m \times N$ obrigatoriamente.

c. $N \times m$.

d. $N \times d$ em todos os casos.

9. Na autodiferenciação reversa, produtos vetor–Jacobiano permitem:

a. propagar derivadas da saída para os nós anteriores do grafo.

b. apagar o grafo antes de calcular a perda.

c. substituir todos os dados por escalares.

d. evitar a regra da cadeia.

10. Durante a retropropagação, os pesos devem ser atualizados:

a. assim que cada derivada intermediária surgir, mesmo que pesos antigos ainda sejam necessários.

b. somente depois de calcular os gradientes necessários para a iteração.

c. antes da propagação para frente.

d. apenas quando a perda for exatamente zero.

Gabarito comentado

1. **a.** A composição de transformações afins continua sendo afim; sem não linearidade, profundidade não amplia essa classe funcional.
2. **a.** A ReLU preserva entradas positivas e zera entradas negativas.
3. **b.** A softmax produz probabilidades não negativas cuja soma é um.
4. **c.** Há $5 \times 3 = 15$ pesos e 3 vieses, totalizando 18 parâmetros.
5. **a.** O resultado trata de existência e aproximação, não garante treinamento, eficiência ou generalização.
6. **a.** Camadas sucessivas podem reutilizar e compor características de níveis anteriores.
7. **b.** O sinal negativo aponta localmente na direção de redução da função objetivo.
8. **c.** O produto $(N \times d)(d \times m)$ resulta em $N \times m$, e o viés é difundido por linha.
9. **a.** O modo reverso acumula sensibilidades por meio da regra da cadeia sem formar Jacobianas completas desnecessárias.
10. **b.** Atualizar antes do fim pode misturar valores novos e antigos e produzir gradientes inconsistentes.

Capítulo 9

Redes Recorrentes

Dados sequenciais possuem ordem e dependências entre posições. Em uma frase, o significado de uma palavra depende das anteriores; em uma série temporal, uma medida recente ajuda a interpretar a seguinte. Uma rede feedforward comum recebe um vetor de tamanho fixo e não mantém, por si só, um estado entre passos.

Uma **rede neural recorrente** (RNN) introduz um estado oculto que é atualizado ao longo da sequência. A mesma transformação é reutilizada em todos os instantes, permitindo processar sequências de diferentes comprimentos.

9.1 Rede recorrente de Elman

Seja $x_t \in \mathbb{R}^{d_x}$ a entrada no instante t e $h_t \in \mathbb{R}^{d_h}$ o estado oculto. Uma RNN simples de Elman calcula

$$h_t = \phi(W_{xh}x_t + W_{hh}h_{t-1} + b_h), \tag{9.1}$$

$$o_t = W_{hy}h_t + b_y, \qquad \hat{y}_t = \psi(o_t). \tag{9.2}$$

Os parâmetros possuem formatos

$$W_{xh} \in \mathbb{R}^{d_h \times d_x}, \quad W_{hh} \in \mathbb{R}^{d_h \times d_h}, \quad W_{hy} \in \mathbb{R}^{d_y \times d_h}.$$

A ativação ϕ costuma ser tanh em uma RNN simples. A função ψ depende da tarefa: identidade para regressão, sigmoide para saída binária ou softmax para classes exclusivas.

O estado inicial h_0 pode ser zero, um parâmetro aprendido ou um vetor fornecido por outro modelo. Essa escolha faz parte da especificação da arquitetura.

9.1.1 Memória como estado

A equação Equação 9.1 combina a observação atual com um resumo do passado. O estado não armazena literalmente todos os itens anteriores; ele é uma representação de dimensão fixa, atualizada recursivamente:

$$h_t = F_\theta(x_t, h_{t-1}).$$

Duas sequências com o mesmo último elemento podem produzir estados diferentes porque percorreram históricos distintos. Ao mesmo tempo, um estado pequeno pode perder detalhes relevantes de sequências longas.

9.1.2 Desdobramento no tempo

Embora o diagrama recorrente contenha um ciclo, para uma sequência finita podemos **desdobrá-lo** em uma cadeia. Cada cópia visual representa a mesma célula em um instante diferente; os parâmetros são compartilhados.

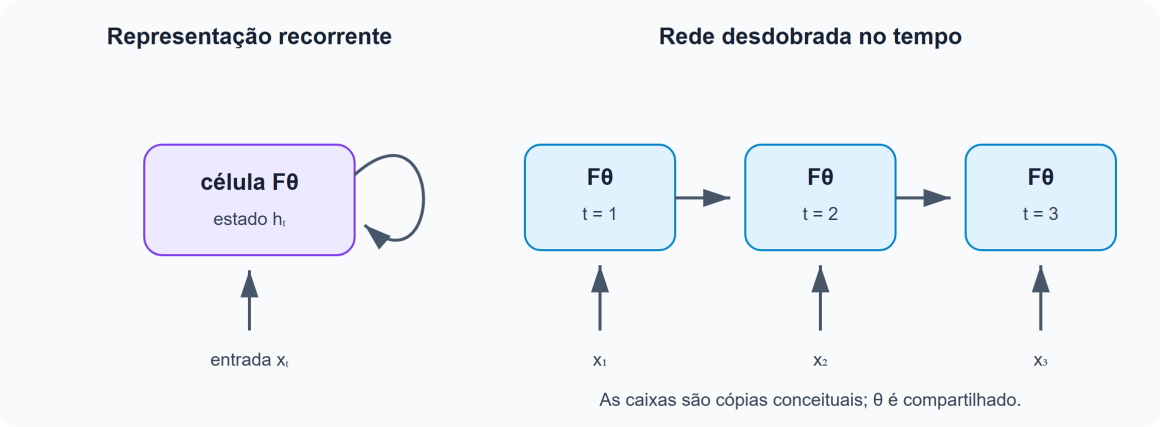


Figura 9.1: Uma RNN recorrente e seu desdobramento: os mesmos parâmetros são reutilizados em todos os passos.

O compartilhamento traz duas consequências:

- o número de parâmetros não cresce com o comprimento da sequência;
- gradientes de todos os instantes contribuem para as mesmas matrizes.

Uma RNN com $T = 100$ passos não possui cem matrizes W_{hh}; possui uma matriz usada cem vezes.

9.1.3 Formatos de entrada e saída

Redes recorrentes podem ser organizadas de acordo com a tarefa:

- **sequência para sequência:** um rótulo por passo, como etiquetagem de palavras;
- **sequência para vetor:** uma saída após ler toda a sequência, como classificação de texto;
- **vetor para sequência:** uma condição inicial gera vários passos;
- **sequência para sequência com comprimentos diferentes:** um codificador produz uma representação usada por um decodificador.

Em lote, uma entrada costuma ter formato $B \times T \times d_x$: exemplos, tempo e atributos. Sequências de comprimentos diferentes podem ser preenchidas até um tamanho comum, mas uma máscara deve impedir que posições artificiais contribuam para a perda ou para métricas.

Preenchimento não cria informação. Além disso, processar uma sequência muito além de seu comprimento real pode alterar estados se a implementação não respeitar a máscara.

9.1.4 Propagação para frente

```
import numpy as np


def rnn_elman(X, W_xh, W_hh, b_h, W_hy, b_y, h0=None):
    """X tem formato (lote, tempo, atributos)."""
    B, T, _ = X.shape
    d_h = W_hh.shape[0]
    h = np.zeros((B, d_h)) if h0 is None else h0.copy()
    estados, saidas = [], []

    for t in range(T):
        h = np.tanh(X[:, t, :] @ W_xh.T + h @ W_hh.T + b_h)
        o = h @ W_hy.T + b_y
        estados.append(h.copy())
        saidas.append(o)

    H = np.stack(estados, axis=1)
```

```
    O = np.stack(saidas, axis=1)
    return O, H
```

O laço temporal é sequencial: h_t depende de h_{t-1}. Dentro de cada passo, todos os exemplos e todas as unidades são vetorizados.

9.1.5 Retropropagação através do tempo

A **retropropagação através do tempo** (BPTT) aplica backpropagation ao grafo desdobrado. Se a perda total é

$$E = \sum_{t=1}^{T} \ell_t(\hat{y}_t, y_t),$$

o estado h_t pode influenciar a perda no próprio passo e todas as perdas futuras. Sua derivada acumula esses caminhos. Em uma forma simplificada,

$$\frac{\partial E}{\partial h_t} = \frac{\partial \ell_t}{\partial h_t} + \frac{\partial E}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial h_t}.$$

Para a recorrência de Elman,

$$\frac{\partial h_{t+1}}{\partial h_t} = \operatorname{diag}(\phi'(z_{t+1}))\, W_{hh}.$$

Produtos repetidos desses Jacobianos explicam dois problemas:

- **gradientes que desaparecem:** normas menores que um são multiplicadas repetidamente, enfraquecendo dependências longas;
- **gradientes que explodem:** normas maiores que um podem crescer rapidamente e desestabilizar o treinamento.

Clipping da norma do gradiente ajuda no segundo caso. Células LSTM e GRU criam caminhos de estado controlados por portas e foram projetadas para facilitar o aprendizado de dependências mais longas; elas reduzem, mas não eliminam, todas as dificuldades.

9.1.6 BPTT truncada

Guardar o grafo de uma sequência muito longa consome memória e aumenta o caminho das derivadas. Na BPTT truncada, processamos blocos de K passos, transportamos o valor do estado para o bloco seguinte, mas interrompemos o grafo de derivação na fronteira.

Isso reduz custo e memória, porém impede que o gradiente atribua crédito diretamente a eventos anteriores à janela. O comprimento K é, portanto, um hiperparâmetro computacional e estatístico.

9.1.7 Relação com redes feedforward

Uma RNN desdobrada se parece com uma rede profunda, mas possui duas diferenças essenciais:

1. a profundidade efetiva depende do comprimento da sequência;
2. os parâmetros das cópias temporais são compartilhados.

Redes feedforward também podem receber uma janela fixa de observações, mas a janela precisa ser escolhida previamente e cada posição costuma ter parâmetros distintos. RNNs oferecem uma regra recorrente aplicável a comprimentos variados.

Hoje, mecanismos de atenção e Transformers são alternativas importantes para muitos problemas sequenciais porque permitem caminhos mais curtos entre posições e maior paralelismo. Ainda assim, RNNs permanecem úteis em fluxos contínuos, modelos compactos e cenários com processamento passo a passo.

> ⚠ Ordem temporal não é ordem do lote
>
> Embaralhar a ordem dos exemplos independentes entre épocas costuma ser adequado. Embaralhar os passos dentro de cada sequência destrói a estrutura temporal que a recorrência deve aprender.

9.1.7.1 Exercícios

1. Verifique os formatos em Equação 9.1 e Equação 9.2.
2. Quantos parâmetros possui uma RNN com $d_x = 10$, $d_h = 32$ e $d_y = 5$?
3. Explique por que o número de parâmetros não depende de T.
4. Diferencie preenchimento, máscara e BPTT truncada.
5. Mostre como produtos repetidos de um escalar $|\lambda| < 1$ ajudam a entender o desaparecimento do gradiente.
6. Quando uma saída por passo é necessária e quando apenas h_T pode ser suficiente?

9.2 Mapas Auto-Organizáveis

Um **mapa auto-organizável** (*Self-Organizing Map*, SOM) é um método de aprendizado não supervisionado que representa dados de alta dimensão por uma grade, geralmente bidimensional, de protótipos. Pontos próximos no espaço dos dados tendem a ativar unidades próximas na grade.

O SOM não descobre rótulos verdadeiros nem aproxima uma “função divina”. Ele produz uma representação exploratória cuja interpretação depende dos dados, da escala dos atributos e dos hiperparâmetros.

9.2.1 Três objetos distintos

Considere dados $x \in \mathbb{R}^d$ e M unidades. Cada unidade i possui:

- uma posição fixa $r_i \in \mathbb{R}^q$ na grade, normalmente $q = 2$;
- um protótipo treinável $w_i \in \mathbb{R}^d$ no espaço dos dados.

As distâncias $\|x - w_i\|$ comparam um exemplo aos protótipos. As distâncias $\|r_i - r_j\|$ comparam posições na grade. Misturar esses dois espaços torna o algoritmo incorreto.

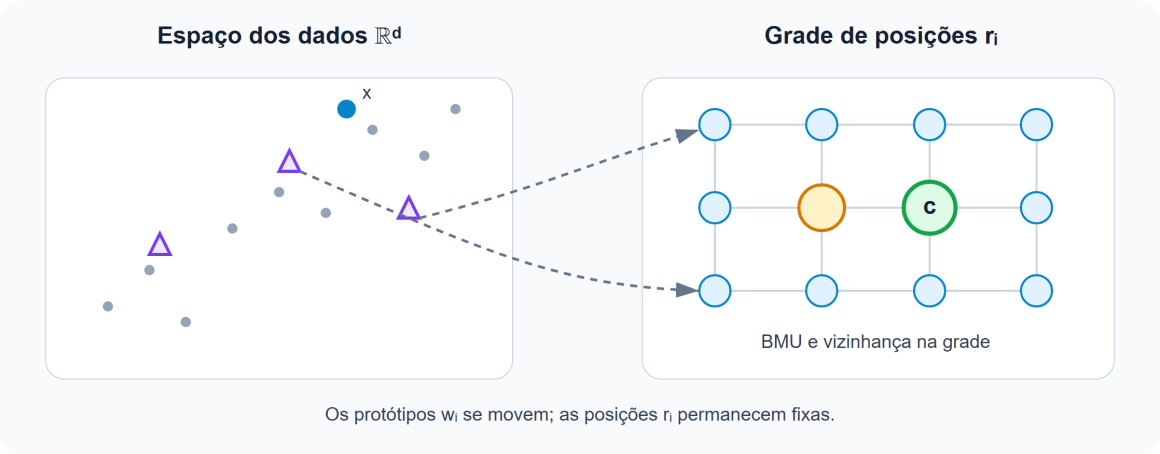


Figura 9.2: Cada unidade ocupa uma posição fixa na grade e possui um protótipo no espaço dos dados; a vencedora e suas vizinhas movem-se em direção ao exemplo.

9.2.2 Unidade de melhor correspondência

Para um exemplo x_t, a unidade vencedora, ou **BMU** (*best matching unit*), é

$$c_t = \arg\min_{i=1,\dots,M} \|x_t - w_i(t)\|^2. \tag{9.3}$$

Usar a distância ou seu quadrado produz a mesma vencedora e evita a raiz quadrada no cálculo.

9.2.3 Função de vizinhança

A influência da vencedora sobre a unidade i pode ser gaussiana:

$$H_{c_t i}(t) = \exp\left(-\frac{\|r_i - r_{c_t}\|^2}{2\sigma(t)^2}\right). \tag{9.4}$$

Essa função não é uma métrica: ela é máxima em unidades próximas e diminui com a distância. O parâmetro $\sigma(t) > 0$ é o raio de vizinhança. O quadrado em $\sigma(t)^2$ é necessário na forma gaussiana usual.

9.2.4 Regra de atualização

Todos os protótipos são atualizados por

$$w_i(t+1) = w_i(t) + \eta(t) H_{c_t i}(t)(x_t - w_i(t)), \tag{9.5}$$

onde $0 < \eta(t) < 1$ é a taxa de aprendizado. A BMU se move mais; suas vizinhas também se aproximam do exemplo, preservando ordem local na grade.

No início, raio e taxa maiores promovem organização global. Depois, ambos diminuem para permitir ajustes locais. Um cronograma exponencial possível para $t = 0, \ldots, T-1$ é

$$\eta(t) = \eta_0 \left(\frac{\eta_f}{\eta_0}\right)^{t/(T-1)}, \qquad \sigma(t) = \sigma_0 \left(\frac{\sigma_f}{\sigma_0}\right)^{t/(T-1)}.$$

Os valores finais devem ser positivos. Reduzir o raio a zero cedo demais transforma o treinamento em competição independente e prejudica a organização topológica.

9.2.5 Algoritmo

1. Padronizar os atributos quando suas escalas não forem comparáveis.
2. Criar as posições r_i de uma grade e inicializar os protótipos w_i.
3. Para cada iteração:
    1. escolher um exemplo, normalmente com ordem embaralhada;
    2. encontrar a BMU por Equação 9.3;
    3. calcular a influência na grade por Equação 9.4;
    4. atualizar os protótipos por Equação 9.5;
    5. reduzir $\eta(t)$ e $\sigma(t)$.
4. Associar cada exemplo à posição de sua BMU para visualizar o mapa.

Inicialização aleatória é simples; amostrar exemplos ou inicializar na direção das primeiras componentes principais pode acelerar a organização. Diferentes sementes podem produzir

orientações distintas da grade sem que uma delas seja necessariamente errada.

9.2.6 Exemplo de uma atualização

Considere dois protótipos

$$w_1 = (0{,}5, 0{,}6, 0{,}8), \qquad w_2 = (0{,}4, 0{,}2, 0{,}5)$$

e o exemplo $x = (0{,}8, 0{,}7, 0{,}4)$. As distâncias quadráticas são

$$\|x - w_1\|^2 = 0{,}3^2 + 0{,}1^2 + (-0{,}4)^2 = 0{,}26,$$

$$\|x - w_2\|^2 = 0{,}4^2 + 0{,}5^2 + (-0{,}1)^2 = 0{,}42.$$

Logo, a BMU é a unidade 1. Com $\eta = 0{,}5$, influência $H_{11} = 1$ e, para simplificar, influência zero sobre a outra unidade,

$$\begin{aligned} w_1' &= w_1 + 0{,}5(x - w_1) \\ &= (0{,}5, 0{,}6, 0{,}8) + 0{,}5(0{,}3, 0{,}1, -0{,}4) \\ &= (0{,}65, 0{,}65, 0{,}60). \end{aligned}$$

O protótipo se deslocou até o ponto médio entre seu valor anterior e o exemplo. Como somente a vencedora foi atualizada, este é o caso limite de **aprendizado competitivo**, semelhante ao k-means on-line. O comportamento distintivo do SOM aparece quando unidades vizinhas também recebem influência positiva.

Por exemplo, se $H_{12} = 0{,}4$, então

$$w_2' = w_2 + 0{,}5 \cdot 0{,}4(x - w_2) = (0{,}48, 0{,}30, 0{,}48).$$

A unidade 2 também se move, mas por uma fração menor.

9.2.7 Implementação NumPy

```
import numpy as np
```

```
def passo_som(x, prototipos, posicoes, taxa, sigma):
    x = np.asarray(x, dtype=float)
    W = np.asarray(prototipos, dtype=float)
    R = np.asarray(posicoes, dtype=float)

    dist_dados = np.sum((W - x) ** 2, axis=1)
    bmu = np.argmin(dist_dados)

    dist_grade = np.sum((R - R[bmu]) ** 2, axis=1)
    influencia = np.exp(-dist_grade / (2 * sigma**2))

    W += taxa * influencia[:, None] * (x - W)
    return bmu, W
```

O índice `[:, None]` transforma a influência de formato M em uma coluna $M \times 1$, permitindo multiplicá-la pelas diferenças $M \times d$ por broadcasting.

9.2.8 Avaliação e visualização

Como não há rótulo obrigatório, usamos critérios internos e inspeção:

- **erro de quantização:** média da distância de cada exemplo ao protótipo de sua BMU;
- **erro topográfico:** frequência com que a primeira e a segunda BMUs não são vizinhas na grade;
- **matriz U:** visualização das distâncias entre protótipos vizinhos;
- distribuição de exemplos por unidade, para detectar unidades vazias ou concentração excessiva.

Baixo erro de quantização não garante boa preservação topológica. Aumentar o número de unidades tende a melhorar quantização, mas também pode criar um mapa esparso e mais difícil de interpretar.

9.2.9 Limitações e uso responsável

SOMs são sensíveis à escala dos atributos, inicialização, ordem dos exemplos, tamanho da grade e cronogramas. A proximidade na grade é uma aproximação da estrutura dos dados, não prova de que dois grupos sejam categorias naturais.

Quando rótulos externos existem, eles podem ser projetados sobre o mapa após o treinamento

para interpretação, mas não devem ser apresentados como rótulos descobertos sem incerteza. PCA, t-SNE, UMAP, k-means e autoencoders respondem a objetivos diferentes e constituem comparações úteis conforme a aplicação.

9.2.9.1 Exercícios

1. Diferencie r_i e w_i, incluindo seus espaços e papéis.
2. Calcule w_2' do exemplo e confirme cada coordenada.
3. O que acontece com Equação 9.5 quando $\sigma(t) \to 0$?
4. Explique por que padronização pode mudar completamente a BMU.
5. Compare erro de quantização e erro topográfico.
6. Implemente uma grade 3×3, gere suas posições e execute uma época do algoritmo.

9.2.10 Síntese do capítulo

RNNs processam sequências mantendo um estado e compartilhando parâmetros ao longo do tempo; seu treinamento desdobra a recorrência e aplica BPTT. SOMs pertencem a outra família: aprendizado competitivo não supervisionado com protótipos organizados em uma grade. Em ambos os casos, a estrutura do modelo — tempo na RNN, vizinhança no SOM — incorpora uma hipótese sobre relações importantes nos dados.

9.3 Exercícios de múltipla escolha

As cinco primeiras questões tratam de redes recorrentes; as demais tratam de mapas auto-organizáveis. Cada item possui uma única resposta correta.

1. Em uma rede recorrente de Elman, o estado oculto h_t depende tipicamente:

   a. apenas de x_t.

   b. de x_t e do estado anterior h_{t-1}.

   c. somente do rótulo final.

   d. de pesos diferentes e independentes em cada instante.

2. “Desdobrar” uma RNN no tempo significa:

   a. representar as aplicações sucessivas da célula como um grafo ao longo dos instantes.

b. duplicar fisicamente o conjunto de dados.

c. remover todas as conexões recorrentes.

d. ordenar os neurônios pelo valor do viés.

3. O compartilhamento de parâmetros ao longo do tempo permite que a RNN:

a. aplique a mesma regra de transição em diferentes posições da sequência.

b. tenha um modelo totalmente distinto para cada comprimento.

c. dispense qualquer estado.

d. produza apenas sequências de tamanho um.

4. Gradientes que desaparecem em sequências longas estão associados a:

a. produtos repetidos de derivadas e matrizes com fatores contrativos.

b. ausência completa da regra da cadeia.

c. uso obrigatório de números inteiros.

d. uma matriz de Gram muito grande.

5. A BPTT truncada reduz custo computacional ao:

a. limitar o número de passos temporais pelos quais o gradiente é propagado.

b. remover a propagação para frente.

c. treinar somente o viés da saída.

d. tornar toda sequência independente do passado.

6. Um mapa auto-organizável (SOM) é um método de aprendizado:

a. supervisionado que exige um rótulo por exemplo.

b. competitivo e não supervisionado.

c. exclusivo para séries temporais rotuladas.

d. baseado em retropropagação de entropia cruzada.

7. A unidade de melhor correspondência (BMU) para uma entrada x é, em geral, o protótipo que:

a. está mais distante de x.

b. possui menor distância a x.

c. ocupa sempre o canto superior esquerdo da grade.

d. recebeu o maior rótulo de classe.

8. Na atualização de um SOM, além da BMU, também são movidos:

a. protótipos vizinhos na grade, com intensidade definida pela função de vizinhança.

b. apenas exemplos do conjunto de teste.

c. todos os protótipos pela mesma quantidade, obrigatoriamente.

d. somente protótipos com rótulo correto.

9. Ao longo do treinamento de um SOM, é comum reduzir gradualmente:

a. a taxa de aprendizado e o raio de vizinhança.

b. a dimensão de cada vetor de entrada.

c. o número de classes verdadeiras.

d. a quantidade de coordenadas da grade em cada época.

10. O erro de quantização de um SOM mede principalmente:

a. a distância média entre cada exemplo e sua BMU.

b. a acurácia de rótulos usados no treinamento supervisionado.

c. o número de gradientes explosivos.

d. a probabilidade PAC de falha.

Gabarito comentado

1. **b.** O estado combina a entrada corrente com uma memória resumida do passado.
2. **a.** O desdobramento explicita as dependências temporais e permite aplicar a regra da cadeia.
3. **a.** Os mesmos pesos são reutilizados, o que expressa invariância da regra de processamento à posição temporal.
4. **a.** Multiplicações sucessivas por fatores menores que um reduzem a magnitude do sinal de gradiente.
5. **a.** A truncagem estabelece uma janela finita de retropropagação, trocando memória temporal longa por eficiência.
6. **b.** O SOM aprende protótipos por competição e cooperação de vizinhança sem usar alvos durante o ajuste.
7. **b.** A BMU minimiza a distância entre a entrada e os vetores protótipos.
8. **a.** A cooperação local organiza protótipos próximos na grade para representar regiões próximas dos dados.
9. **a.** Começar com vizinhança ampla favorece organização global; reduzi-la permite refinamento local.
10. **a.** Essa distância resume a fidelidade com que os protótipos representam os dados, mas não mede sozinha a preservação topológica.