Aprendizado de Máquina

UFAL, Maceió — Inverno 2019

Enno Nagel

Inverno 2019

Curso dado no Instituto Matemático da UFAL no semestre de inverno de 2019. Introduz

- às aplicações,
- às implementações e
- à teoria

dos algoritmos de aprendizado de máquina.

Sumário

Introdução

Para quê estudar o Aprendizado de Máquina? A resposta esmagadora do guru Andrew Ng:

"I hope we can build an AI-powered society that gives everyone affordable healthcare, provides every child a personalized education, makes inexpensive self-driving cars available to all, and provides meaningful work for every man and woman."

Traduzido para português:

"Espero que possamos criar uma sociedade baseada na inteligência artificial, que

- oferece um plano de saúde acessível a todo mundo,
- proporciona uma educação personalizada a cada criança,
- disponibiliza carros auto-conduzidos baratos a todos, e
- um trabalho significativo com sentido a cada homem e mulher."

Amem. Vamos levar o Brasil para frente!

Objetivo do Curso

Este curso inicia o leigo aos métodos básicos do *aprendizado de máquina*. Tradicionalmente, foram introduzidas regras ao computador para ele calcular as consequências. No *aprendizado de máquina*, são introduzidos dados ao computador para ele calcular as regras, detectar as regularidades neles (e, em seguida, calcular as consequências destas regras).

Com o advento de cada vez maiores volumes de dados e maior potência computacional, o aprendizado de máquina torna-se cada vez mais proveitoso. Por exemplo,

- na medicina: a partir de um conjunto de dados de saúde com os diagnósticos por especialistas, aprender a diagnosticar.
- na *mineração de dados*, o uso de aprendizado de máquina para detectar estruturas estatísticas lucrativas (em bancos de dados relacionais). Notavelmente, para antecipar
  - os interesses de um usuário de um portal on-line a partir daqueles nas visitas anteriores;
  - em particular, as futuras compras dos clientes de um comércio on-line a partir das anteriores.

Resumo

O *aprendizado de máquina* faz cada vez mais sucesso na resolução de problemas computacionais até há pouco acreditados irresolúveis: Talvez o mais espetacular seja a derrota fulminante do melhor jogador de Go mundial por AlphaGo, uma inteligência artificial baseada no aprendizado de máquina. Até então, os computadores chegavam nem sequer perto aos mestres. Como exemplo protótipo e primeira aplicação, olhamos a classificação de e-mails em Spam (= lixo) e Ham (= não Spam) pelo aprendizado a partir dum conjunto exemplar de e-mails sorteados. Este algoritmo de aprendizado de máquina, baseando-se unicamente na frequência de palavras-chave lixosas no e-mail, revelou-se mais eficiente do que todos os outros até então criados.

Os métodos básicos por trás do aprendizado de máquina são principalmente geométricos: Dado um conjunto de pontos, buscam-se (por exemplo, nos métodos da *regressão linear/logística*, da *Support vetor Machine* ou do *K-means*) as melhores (em um sentido preciso) retas, hiperplanos (em dimensão maior) ou compartimentos para separá-los.

Além da solução deste problema teórico, geométrico, há os problemas práticos:

- o da dificuldade computacional, e
- o da significância (como modelação da realidade).

Para reduzir a dificuldade computacional há, por exemplo, o método da *análise de componentes principais* (para reduzir o número de parâmetros com perda

informacional mínima). Como problema de significância, surge o do *Sobre-Ajuste*, quer dizer, que o resultado só se aplica aos dados usados, mas em geral não, e o *Método de Regularização* ajuda mitigá-lo.

Cronograma

Estudamos os fundamentos do curso Machine Learning por Andrew Ng disponível no coursera.org pelo livro Abu-Mostafa, Magdon-Ismail, e Lin (2012) disponível gratuitamente online no endereço Learning from Data.

Começamos com a parte teórica:

1. Explicamos e formalizamos o que é um problema de aprendizado (supervisionado).
2. Formalizamos a noção de aprendizagem pelo PAC (= Provavelmente Aproximadamente Correto). O PAC dá em particular uma explicação matemática ao problema recorrente do *Sobre-Ajuste*, quando as conclusões se aplicam só aos dados analisados, mas não revelam nada geral.
3. Mostramos pela dimensão de Vapnik-Chervonenkis que uma grande classe de problemas é apreensível.

Concluímos com a parte prática:

4. Os métodos lineares:
    1. O método do `Perceptron` em uma variável e em várias variáveis. Geometricamente, busca-se um hiperplano que separa dois conjuntos de pontos.
    2. O método da *Regressão Linear*, geometricamente, para dados pontos busca-se uma reta que os aproxime o melhor possível.
    3. O método da Regressão *Logística* que mede a probabilidade de uma propriedade; Por exemplo, se um e-mail é spam ou não.
5. O Método do *Máximo Declive (estocástico)* (`Gradient Descent`) para encontrar extremos de funções (em várias variáveis); por exemplo, minimizar as distâncias nos métodos lineares acima.

6. O método da *Máquina de Vetores de Suporte* (Support vetor Machine), como extensão não-linear (kernelization) do Perceptron: Geometricamente, busca-se um hiperplano que separa pontos em duas classes. Em comparação à Regressão Logística computacionalmente preferível quando há poucos parâmetros, quer dizer, a dimensão é pequena.

7. Redes Neurais como iteração da Regressão Logística em várias variáveis;

    - os seus modelos, e
    - o algoritmo da Retropropagação de Erro, um método de máximo declive que reajusta os pesos dos neurônios conforme a uma regra de correção de erro.

8. O método das *K-médias* (K-means) cuja saída é finita. Geometricamente, busca-se uma divisão de pontos em compartimentos tal que ela minimize as distâncias entre os pontos em cada compartimento.

9. O método da *Análise da Componente Principal* (análise de componentes principais) para projetar um conjunto de pontos a um espaço de dimensão menor sem perda de informação quanto à sua classificação.

Otimizar

Para otimizar, isto é, encontrar um ponto em que uma função (de erro) E tem o seu valor mínimo, usaremos

- uma aproximação iterativa, um método de ensaio e erro direcionado, como no Perceptron,
- a projeção de um vetor a um (hiper)plano (onde o vetor de saída é o vetor do (hiper)plano que é mais próximo do vetor de entrada), e
- a computação de um ponto em que a derivada ∇E de E é zero.

Os métodos numéricos (iterativos) para calcular esta projeção ou o ponto em que a derivada é zero serão só superficialmente explicadas.

Aspectos Matemáticos

Recapitulamos noções básicas

- da Álgebra Linear para tratar problemas multidimensionais como o da regressão linear em várias variáveis,
- da análise multidimensional para encontrar pontos (localmente) máximos de funções em várias variáveis.

Links

Lembremo-nos de que, mesmo se o link `URL` expirou, o conteúdo é ainda disponível pelo `archive.org` sob o endereço https://web.archive.org/web/URL/ . Por exemplo, a animação da regressão linear http://mste.illinois.edu/activity/regression/ da Universidade do Illinois permanecerá disponível sob https://web.archive.org/web/http://mste.illinois.edu/activity/regression/ . Neste caso, a página foi arquivada no dia 5 de maio 2018; o seu estado nesta data continua a estar disponível em https://web.archive.org/web/20180505102200/http://mste.illinois.edu/activity/regression/ .

Notações

Esta secção explica a linguagem matemática básica, conjuntos e funções. Como as notações usada na matemática acadêmica diferem em uns pontos das do ensino escolar, repitamos as mais comuns:

Conjuntos

A noção fundamental da matemática é a de um *conjunto*; segundo os dicionários Aurélio, Houaiss ou Michaelis:

- qualquer coleção de seres matemáticos,
- qualquer reunião de objetos, determinados e diferenciáveis, quer esses objetos pertençam à realidade exterior, quer sejam objetos do pensamento, ou
- qualquer reunião das partes que constituem um todo.

Esta coleção de seres matemáticos é denotada por { seres matemáticos }. Se um ser matemático a pertence ao conjunto A, dizemos que a é em X e escrevemos $a \in \mathrm{A}$ ou $\mathrm{A} \ni a$.

Por exemplo, o conjunto

- dos *números naturais* $\mathbb{N} = \{1, 2, 3, \ldots\}$ é denotado por $\mathbb{N}$.
- dos *números inteiros* $\mathbb{Z} = \{-2, -1, 0, 1, 2, \ldots\}$ é denotado por $\mathbb{Z}$,
- dos *números racionais* $\{x/y$ para x,y em $\mathbb{Z}\}$ é denotado por $\mathbb{Q}$; por exemplo, $2/3$ é nele , e
- dos *números reais* $\{a_{\mathrm{N}}10^{\mathrm{N}} + \cdots + a_1 10 + a_0 + a_{-1}10^{-1} + \cdots\}$ para a_{N}, …, a_1, a_0, a_{-1} em { 0, 1, …, 9 } é denotado por $\mathbb{R}$; por exemplo, $\pi = 3{,}14159\ldots$ é nele.

Podemos comparar dois conjuntos A e B:

- denote $\mathrm{A} \supseteq \mathrm{B}$ que A contém (ou inclui) B, e
- denote $\mathrm{A} \subseteq \mathrm{B}$ que A é contido (ou incluso) B.

Podemos formar um novo conjunto a partir de dois conjuntos A e B:

- denote $\mathrm{A} \cup \mathrm{B}$ a *união* de A e B, o conjunto dos elementos que pertencem a A *ou* B,

- denote $A \cap B$ a *intersecção* de A e B, o conjunto dos elementos que pertencem a A *e* B;
- se $A \supseteq B$, então denote $A - B$ a *diferença* entre A e B, o conjunto dos elementos que pertencem a A mas *não* a B.

Por exemplo, os *números irracionais* são todos os números reais que não são racionais, isto é, que pertencem ao $\mathbb{R} - \mathbb{Q}$.

Denote $A \times B$ o seu *produto* cujos elementos são (a,b) para a em A e b em B, isto é,

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

Se B = A, escrevemos A^2 em vez de $A \times A$, e A^3 em vez de $A \times A \times A$, e assim por diante. Por exemplo, $\mathbb{R}^2$ é o plano (euclidiano), e $\mathbb{R}^3$ é o espaço.

Funções

Fórmula. Uma *função* descreve a dependência entre duas quantidades. No ensino escolar, é usualmente denotada por uma equação

$$f(x) = \text{ expressão em } x$$

onde na expressão figuram

- operações aritméticas $+, -, \cdot, /$ ...
- sobre números reais, constantes, funções algébricas como x^2 e funções especiais como $\exp(x)$, $\log(x)$, $\text{sen}(x)$, ...

Por exemplo:

1. A distância s percorrida por um objeto na queda livre depende do tempo da queda t: vale $s = g \cdot t^2$ onde $g = 9{,}8\text{m/s}^2$ é a constante da aceleração gravitacional da terra.
2. A forca de atração K entre duas massas m' e m'' depende de m', m'' e a distância r entre elas: Pela lei gravitacional de Newton $K = G(m'm'')/r^2$ onde $G = 6{,}67 \cdot 10^{-11}\text{Nm}^2\text{kg}^{-2}$ é a constante de gravidade universal de Newton.

Porém, existem outras dependências, por exemplo:

3. O preço de uma entrada ao teatro de fantoches depende do número da fila do assento: Esta dependência descreve-se por uma tabela de preços, dando o número da fila 1, 2, 3, ... e o preço correspondente de 100 Centavos, 20 Centavos, 5 Centavos, ...

Definição. Uma *função* f é uma regra que associa a cada elemento x de um conjunto X exatamente um elemento y de um conjunto Y.

Esta associação é simbolicamente expressa por $y = f(x)$. Refiramo-nos

- a x como *variável (independente)* ou *argumento*,
- a y como *variável dependente* ou *valor (da função)*,
- a X como *domínio* da função, e
- a Y como *contra-domínio* da função.

A *imagem* $\mathrm{im}f$ de f é o conjunto de todos os valores de f,

$$\mathrm{im}f = \{ \text{ todos os } f(x) \text{ para } x \text{ em X}\}.$$

Por exemplo, imsen = [−1,1] e $\mathrm{im}f = \{c\}$ para a função constante $f \equiv c$. O contra-domínio contém a imagem, mas não necessariamente coincide com ela,

$$\mathrm{im}f \subseteq \mathrm{Y}.$$

Usualmente, vemos funções com X e Y $\subseteq \mathbb{R}$, isto é, *funções reais*; chamamo-nas muitas vezes simplesmente de funções. Usualmente as letras a,b,c,$\ldots$,r,s,t,u,x,y,z denotem números reais; as letras i,j,k,l,n,m denotem números naturais (usados para enumerações). Nos nossos exemplos:

1. Aqui (a magnitude) de t em $\mathbb{R}_{\geq 0}$ e $f(t) = s \in \mathbb{R}_{\geq 0}$. Tem-se $\mathrm{X} = \mathbb{R}_{\geq 0}$ e $\mathrm{im}f = \mathbb{R}_{\geq 0}$.
2. Aqui (a magnitude) de m',m'' em $\mathbb{R}_{\geq 0}$ e K em $\mathbb{R}_{\geq 0}$. Tem-se $\mathrm{X} = (\mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0}) \times \mathbb{R}_{\geq 0}$ e $\mathrm{im}f = \mathbb{R}_{\geq 0}$.
3. Aqui $\mathrm{X} = \mathbb{N} = \{1,2,3\}$ e $\mathrm{im}f = \{100,20,5\}$.

Funções reais aparecem em diversas formas: além de

- uma equação,

obtemos pela substituição da variável por um número real como 0,5, 1, 2, 1000, ...

- uma tabela de valores x e y dado pela avaliação do lado direito da igualdade $f(x) = ...$, e
- um *gráfico*

    1. pela marcação dos pontos (x,y) no plano que são dados pelos valores da tabela, e
    2. pela conexão destes pontos.

Em mais detalhes:

- Na forma de uma equação como nos exemplos acima: $y = f(x)$, por exemplo, $y = x^2$,sen$x$ ... Para facilitar, fazemos o convênio seguinte: Se escrevemos apenas a regra de associação, por exemplo, $f(x) = 1/(x(x-3))$ ou $\sqrt{x}$, então o domínio é o subconjunto máximo de $\mathbb{R}$ para que esta regra seja definida (quer dizer, faça sentido). Por exemplo, nestes exemplos, $X = \mathbb{R} - \{0,3\}$ e $X = \mathbb{R}_{\geq 0}$.
- Na forma de uma tabela de valores, por exemplo: Um experimento mede a tensão U de um resistor em dependência da corrente Y. A tabela tem as entradas $Y = 50; 100; 150, \ldots$ (mA) e $U = 2; 4{,}6; \ldots$ (V). Aqui $X = \{50,100,\ldots\}$ (e $Y = \mathbb{R}_{\geq 0}$). (Gostaríamos de extrapolar esta função, isto é, estender o seu domínio a $\mathbb{R}_{\geq 0}$; a este fim, parece provável que $U = RI$ com $R = 0{,}04\Omega$.)
- A equação funcional $y = f(x)$ associa a cada valor x um único valor y, em símbolos, $x \mapsto y = f(x)$. Um tal par de valores (x_0,y_0) pode ser interpretado como ponto P no plano cartesiano. Para cada par de valores (x_0,y_0) obtemos exatamente um ponto. Dado um ponto P, os números reais x_0 e y_0 são chamados das *coordenadas (cartesianas)*. O conjunto de todos os pontos $(x, y = f(x))$ forma a *curva da função* (ou *gráfico*), que ilustra o percurso da função $y = f(x)$. (Por exemplo, a parábola $y = x^2$ ou uma função afim $y = ax + b$.)

    Para desenhar a curva:

    1. Enche uma tabela de valores para uns argumentos x', x'', ...
    2. Desenha os pontos dados nela, e
    3. Conecta-os para obter o percurso da curva da função.

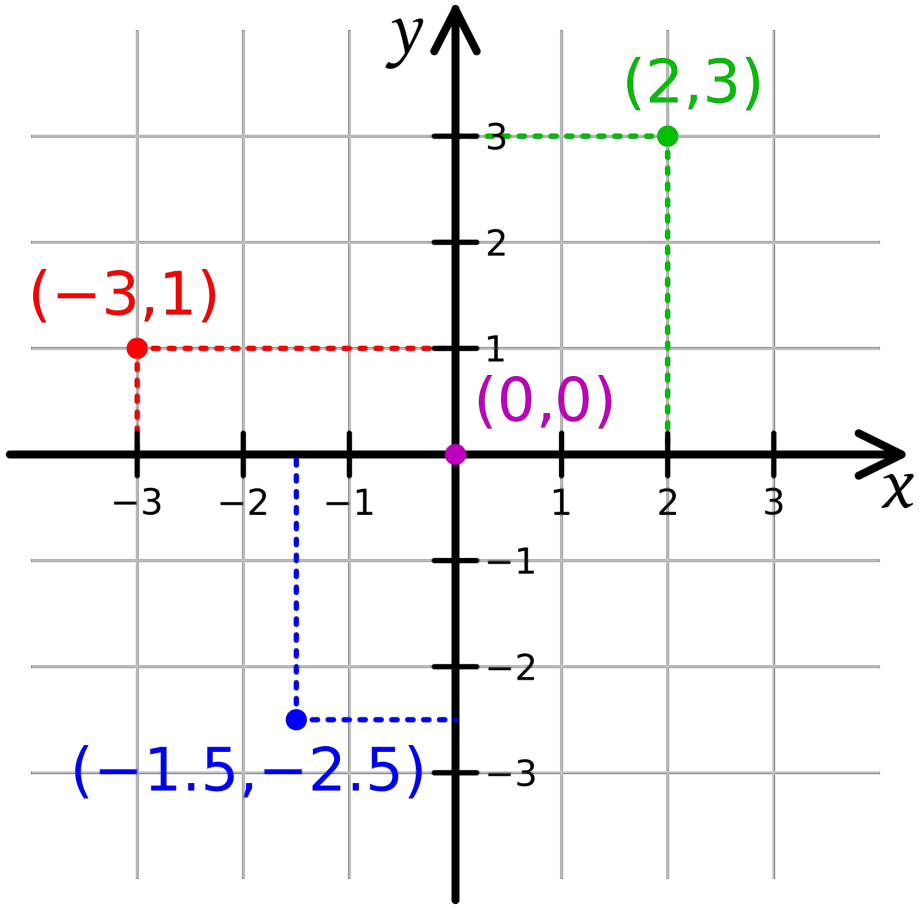


Figura 1: Coordenadas Cartesianas de um ponto nos eixos x e y

De vez em quando, para destacar que um objeto, por exemplo, uma função $h\colon \mathrm{X} \to \mathrm{Y}$, depende de outro objeto, por exemplo, de um conjunto D ou uma ênupla w, escrevemos $h(\mathrm{D})$ ou h_{D} respetivamente $h(w)$ ou h_w em vez de h.

Aplicação. No ensino acadêmico, uma *função* ou *aplicação* é denotada por

$$f\colon \mathrm{X} \to \mathrm{Y}$$

para os dois conjuntos,

- o *domínio* X, e
- o *contra-domínio* Y de f

dizemos que *manda* ou *envia* cada *argumento* x em X a um único *valor* y em Y.

Lê-se que f é uma função de X a Y, ou tem *argumentos* em X e *valores* em Y. No contexto informático, pensamos de f como um algoritmo, e referimo-nos a X e Y como *entrada* e *saída.*

Uma função manda ou envia cada *argumento* x em X a um único *valor* y em Y (ou *associa* a cada x um único y). Escrevemos

$$x \mapsto y$$

e denotemos este valor y por $f(x)$. Frequentemente, $\mathrm{X} = \mathbb{R}$ ou $\mathrm{X} = \mathbb{R} \times \cdots \times \mathbb{R}$ e $\mathrm{Y} = \mathbb{R}$ ou $\mathrm{Y} = \{\pm 1\}$. Por exemplo, a função $f : \mathbb{R} \to \mathbb{R}$ dada por $x \mapsto x^2$ manda 2 em $\mathbb{R}$ a $2^2 = 4$ em $\mathbb{R}$.

Função e *aplicação* são sinónimos. Contudo, a conotação é outra: Se os objetas do domínio são, por exeplo, números inteiros, conjuntos, então *aplicação* é mais comum, enquanto função é principalmente usada quando o domínio consiste de (ênuplas de) números reais.

Formar Novas Funções.

Composição. A partir de duas funções f e g (sobre quaisquer domínios e imagens) pode ser obtida outra por *concatenação*; a *função composta* ou a *composição* $g \circ f$, dado que a imagem de f é contida no domínio de g, isto é, $\mathrm{Y}(f) \subseteq \mathrm{X}(g)$. É definida por

$$g \circ f : x \mapsto g(x) \mapsto f(g(x)),$$

simbolicamente, para obter $g \circ f(x) = g(f(x))$, substituímos x em $g(x)$ por $f(x)$.

Se temos duas funções,

$$f : \mathrm{X} \to \mathrm{Y}, \quad \text{e} \quad g : \mathrm{Y} \to \mathrm{Z}$$

isto é, tais que os valores de f são argumentos de g, a sua *composição* é definida por

$$x \mapsto f(x) \mapsto g(f(x)),$$

isto é, a saída de f é a entrada de g, e denotada por

$$g \circ f : \mathrm{X} \to \mathrm{Z}.$$

Inversão de funções. Por definição, uma função $f : \mathrm{X} \to \mathrm{Y}$ associa a cada argumento $x \in \mathrm{X}$ exatamente um valor $y \in \mathrm{Y}$. Frequentemente surge o problema inverso: Dado um valor y, determina o seu argumento x sob f, isto é, tal que $y = f(x)$. Se uma função f é *injetora*, isto é, $x' \neq x''$ implica $f(x') \neq f(x'')$, isto é, a argumentos diferentes são associados valores diferentes, então a cada valor $y \in \mathrm{im} f$ é associado um único argumento $x \in \mathrm{X}$. A função $g : \mathrm{im} f \to \mathrm{X}$ obtida pela associação inequívoca *inversa* $y \mapsto x$, ou $g(y) = x$ é a *função inversa* ou o *inverso* de f e denotada por $g = f^{-1}$. Ora, y é a variável *independente* e x a variável *dependente*. Em fórmulas, a função inversa g é obtida pela permutação das duas variáveis x e y na equação $y = f(x)$.

Matematicamente, a função f tem o inverso $g = f^{-1}$, se $g(f(x)) = x$ e $f(g(x)) = x$. Por exemplo, sobre $\mathbb{R}_{\geq 0}$ vale $\sqrt{(x^2)} = x = (\sqrt{x})^2$ para $f(x) = x^2, g(x) = \sqrt{x}$, e $(2x+1)/2 - 1/2 = x = 2(x/2 - 1/2) + 1$ para $f(x) = 2x+1$ e $g(x) = x/2 - 1/2$.

Para desenhar a função inversa, poderíamos permutar as designações dos dois eixos. Porém, isto comumente não se faz. Porém, se o domínio e a imagem coincidem, o gráfico da função inversa $g = f^{-1}$ é o espelhamento do gráfico da função invertida f na diagonal dos pontos cuja coordenada x é igual à coordenada y.

Exemplos. Olhemos uns exemplos de funções invertíveis. Observamos em particular que a invertibilidade depende do domínio: Quanto menor o domínio, tanto mais provável que a função seja invertível.

- A parábola $y = x^2$ *não* é invertível sobre $\mathbb{R}$, mas só sobre $\mathbb{R}_{\geq 0}$ e $\mathbb{R}_{\leq 0}$ (pela radiciação quadrática).
- A função $y = 2x+1$ cresce de modo estritamente monótono, em particular, é invertível.
- O seno (que mede o comprimento do cateto oposto no círculo unitário) cresce de modo estritamente monótono no intervalo $[-\pi/2, \pi/2]$; logo tem um inverso, o *arco-seno* neste intervalo.

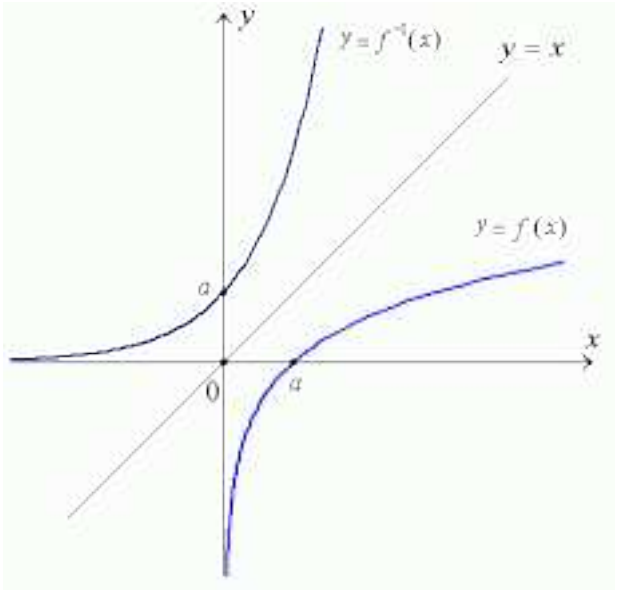


Figura 2: A função inversa

- a exponencial exp é invertível pelo logaritmo log.

Resumimos:

- Toda função estritamente monótona (crescente ou decrescente) é invertível.
- Na inversão o domínio e imagem trocam os papéis.
- Se o domínio e imagem coincidem, obtemos o gráfico da função inversa pelo espelhamento do gráfico da função na diagonal $y = x$.

Algarismos

Um número real no dia-a-dia é escrito em notação decimal

$$a_N \dots a_1 a_0, a_{-1} \dots$$

com $a_N, \dots, a_1, a_0, a_{-1} \dots$, em $\{0,1,\dots,9\}$. Isto é, é como soma em potências de 10 e com algarismos $0, \dots, 9$. Por exemplo,

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

Em vez da *base decimal* $b = 10$, existem outras. As mais comuns na informática são

- a *base binária* $b = 2$ com os algarismos 0 e 1, e
- a *base hexadecimal* $b = 16$ com os algarismos $0, \ldots, 9$ e A, B, C, D, E e F (que correspondem a 10, 11, 12, 13, 14 e 15).

Isto é,

- no conta-quilómetro binário, o último algarismo retorna à posição inicial após dois quilómetros, o penúltimo após $2^2 = 4$ quilómetros, e assim por diante,
- no conta-quilómetro hexadecimal, o último algarismo retorna à posição inicial após 16 quilómetros, o penúltimo após $16^2 = 256$ quilómetros, e assim por diante.

Por exemplo,

$$1101 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$

e

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

1 O que é um problema de aprendizado?

Tradicionalmente, foram introduzidas regras ao computador para ele calcular as consequências.

regras	→ computador →	resultados

No *aprendizado de máquina*, são introduzidos dados ao computador para ele calcular as regras, detectar as regularidades neles (e, em seguida, calcular as consequências destas regras).

dados	→ computador →	regras	(→ computador →	resultados)

Com o advento de cada vez maiores volumes de dados e maior potência computacional, o aprendizado de máquina torna-se cada vez mais proveitoso.

1.1 Categorias de Aprendizado

Primeiro destacamos a diferença entre **aprendizado** e **especificação** para resolver um problema de classificação. Com efeito, o aprendizado leva a especificação:

Na *especificação*, são introduzidas regras ao computador para ele calcular as consequências. No *aprendizado de máquina*, são introduzidos dados ao computador para ele especificar as regras (e, em seguida, calcular as consequências das regras).

Recordemo-nos

dados	→ computador →	regras	→ computador →	resultados

Enquanto

- o computador à esquerda aprende as regras, especifica-as,
- o computador à direita aplica-as para calcular os resultados.

Exemplo: para distinguir entre valores de moedas pelos seus pesos e tamanhos, usamos

- a *especificação* se contatamos a cada da moeda e classificamos as moedas por estas medidas com uma certa margem de erro, e
- o *aprendizado* se derivamos estas medidas com uma certa margem de erro a partir de um grande conjunto de moedas.

Concluímos:

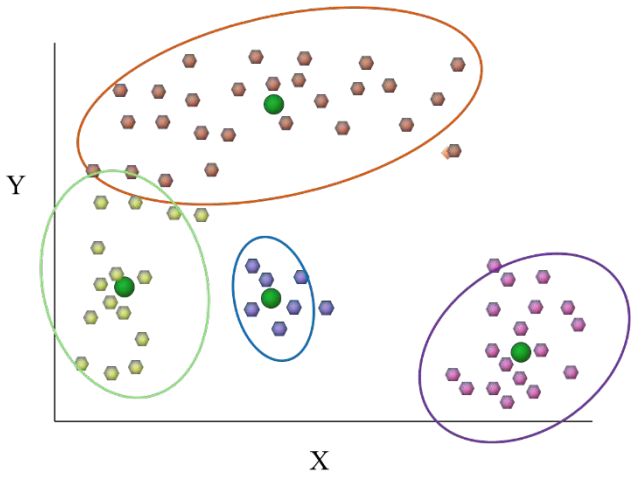


Figura 3: Agrupamento de moedas por peso e tamanho

Se os círculos foram desenhados *antes* dos pontos, trata-se de *especificação*. Se os círculos foram desenhados *depois* dos pontos, trata-se de *aprendizado*.

O termo que usamos daqui para diante

- para os dados a partir de que aprenderemos será *a amostra* ou *os exemplos*.

Pensamos

- de cada exemplo como uma *entrada*, e
- do nosso resultado de aprendizado (tipicamente uma classificação) como *saída*.

Aprendizado com Supervisionamento. No aprendizado com supervisionamento, os exemplos são **classificados**. Isto é, temos pares

$$(\text{entrada}, \text{saída})$$

Por exemplo, duas pastas de e-mails, a primeira do tipo `spam`, a outra do tipo `ham`.

Aprendizado por Reforço. No aprendizado por reforço, os exemplos da amostra são **parcialmente avaliados**. Isto é, temos pares

$$(\text{entrada}, \text{saída e a sua avaliação})$$

Destacamos que, dada uma entrada x, não necessariamente todos os pares (x,y) para todas as possíveis saídas y são contidos na amostra.

Exploração. Quer dizer, não sabemos necessariamente quão boas outras saídas poderiam ter sido. Por exemplo, ao aprender um jogo de tabuleiro, dado uma configuração x do tabuleiro, y será o lance e a sua avaliação a probabilidade de ganhar o jogo.

A avaliação. No aprendizado por reforço, a avaliação é uma função na entrada e saída. Por exemplo,

- no jogo Xadrez, a função poderia avaliar a configuração do tabuleiro
  - pelo número de peças de cada um,
  - pela a sua centralidade e agilidade;
- no jogo Go, a função poderia contar
  - as pedras e
  - as casas controladas por elas;
- no jogo Gamão, simplesmente avaliar uma configuração como
  - *neutra* se o partido é indeciso,
  - *positiva* (+1), se o partido é ganhado, e
  - *negativa* (−1), se o partido é perdido.

Aprendizado sem Supervisionamento. No aprendizado sem supervisionamento, os exemplos são **inclassificados**.

Isto é, o otimizador

1. Precisa de criar as possíveis saídas, ou os rótulos, e
2. *depois* classifica.

O exemplo principal é o *clustering*, ou *agrupamento*. Por exemplo, mesmo sem rótulos, as moedas podem ser agrupadas pelos seus pesos e tamanhos, vide Figura 3

1.2 Exemplos de Aprendizado Supervisionado

Por exemplo,

- na medicina: Aprender a diagnosticar a partir de
  - dados sobre à saúde dos pacientes e
  - os seus diagnósticos.
- na identificação de conteúdo: Decidir se um e-mail é `Spam` (= lixo) (ou `Ham` [= não Spam]) a partir de duas pastas de e-mails,
  - uma do tipo `Spam` e
  - a outra do tipo `Ham`.

  O otimizador de aprendizado a ser apresentado, que se baseia unicamente na frequência de palavras-chave lixosas no e-mail, revelou-se mais eficiente do que todos os outros até então criados.
- nas finanças: Decidir, para um suplicante de um crédito, sobre
  - a questão se conseguirá pagar as suas dívidas ou não,
  - a probabilidade que conseguirá pagar as suas dívidas,
  - sobre a quantia que lhe será concedida,

  a partir
  - dos dados financeiros de credores anteriores e
  - dos seus desempenhos no pagamento das dívidas (quer dizer, faliu ou cumpriu?).

- no futebol: decidir se um jogador pertence à seleção nacional a partir das
  - capacidades (físicas) de jogadores da `Série A` e
  - se foram nomeados.

1.3 Formalização

Formalmente, em um problema de aprendizado, temos

- a **entrada** X,
- a **saída** Y,
- os **dados** ou a *amostra* $D = \{x_1, \ldots, x_N\}$ (= um conjunto finito em X sobre o qual a *função alvo* é conhecida, isto é, $y_1 = y(x_1)$, $\ldots$, $y_N = y(x_N)$ são conhecidos, e
- a **função alvo** (ou *função de regressão* ou *classificadora*) $y \colon X \to Y$ (= a função ótima, que sempre acerta, a ser aproximada; chamamo-la também de função *divina* para exprimir que existe e é aproximável com certa probabilidade, mas inatingível)

e

- as **hipóteses** $H = \{h \colon X \to Y\}$ (= um conjunto de funções que aproximam y), e
- uma medida de erro $E \colon X \times X \to [0, \infty]$ que mede a diferença entre os valores da hipótese e os da função alvo, e
- um **algoritmo** (ou **otimizador**) A que escolhe uma função h em H que aproxima" y "o melhor", em particular, minimiza E sobre D.

Enfatizamos que

- a entrada, a saída, a amostra e a função alvo são *dadas*,
- as hipóteses e o otimizador são *escolhidos* pelo programador.

As hipóteses e o otimizador são chamados o *Método de Aprendizado.*

Observação. Mais adiante, a função alvo, em vez de ser uma função *determinista* $y \colon X \to Y$, será uma *variável aleatória*, isto é, intuitivamente, para cada argumento x, em vez de devolver

- um *único* valor y_0,

devolve

- ou para *cada* valor y_0 uma probabilidade $\mathrm{P}(y = y_0)$ (quando Y é discreto, por exemplo, binário, finito ou $\mathbb{N}$),
- ou para *cada* intervalo de valores $[y', y'']$ uma probabilidade $\mathrm{P}(y \text{ em } [y', y''])$ (quando Y é contínuo, por exemplo, um intervalo $[a, b]$, a reta $\mathbb{R}$, ou o plano $\mathbb{R} \times \mathbb{R}$).

(Em linguagem matemática, cada possível valor $y = y_0$ constitui um *evento* cuja probabilidade $\mathrm{P}(y = y_0)$ é medida pela *distribuição*, ou *medida de probabilidade*, P.)

Porém, os valores $y_1, \ldots, y_N$ para os dados $x_1, \ldots., x_N$ continuam ser deterministas. O otimizador tem de deduzir as probabilidades dos valores pelas suas frequências na amostra. Uma tal função divina aleatória é chamada de uma função com *ruído*.

1.4 Formalização dos Exemplos

Nos exemplo

- do diagnóstico, temos
  - X = ênuplas de atributos médicos (= os de um paciente)
  - Y = doenças
  - D = diagnósticos históricos
- da classificação de e-mails em `Spam` ou `Ham`, temos
  - X = ênuplas do número de ocorrências de palavras (= os de um e-mail)
  - Y = Sim ou Não
  - D = e-mails recebidos e suas classificações como `Spam` ou `Ham`
- da questão se um devedor poderá pagar as suas dívidas ou não, temos
  - X = ênuplas de atributos financeiros (= os de um e-mail)
  - Y = Sim ou Não
  - D = históricos de créditos concedidos
- da composição da seleção nacional, temos

  - X = ênuplas de capacidades (= as de um jogador)
  - Y = Sim ou Não
  - D = seleções anteriores.

Observamos que os últimos três exemplos, a classificação de e-mails, de credores e de jogadores, são exemplos de uma **classificação binária**, isto é, a saída é binária.

Enquanto as ênuplas na classificação de e-mails têm muitos entradas (já que existem muitas palavras), a dimensão dos outros dois exemplos é menor e poderemos dar uma interpretação geométrica. Em mais detalhes, olhamos, por exemplo

- a avaliação da solvência de um possível devedor por um banco. A entrada X aqui são os dados financeiros de um cliente, por exemplo
  - idade
  - salário
  - haveres
  - dívidas
- a avaliação da aptidão de um jogador para a seleção nacional. A entrada X aqui são os dados físicos do jogador, por exemplo a velocidade
  - do chute, e
  - do jogador

1.5 Pontuação

Nestes dois exemplos, da discriminação entre credores e jogadores aptos,

- a entrada pode ser interpretada como ênupla de números reais, e
- a saída é binária: Sim e Não.

Neste caso, podemos dar uma *pontuação* ao possível devedor ou ao jogador relativo aos seus atributos financeiros ou físicos. Mais explicitamente, avaliar as suas *importâncias* (ou *pesos*) e *somar* os atributos (*features*) pesados. Se esta soma é suficientemente alto, quer dizer, em cima de um certo *limiar*, aceitamos o devedor ou jogador.

Por exemplo,

- para certo Neymar, a velocidade
  - do chute é 100 km por hora, e
  - do jogador é 40 km por hora;
- para certo Roberto Carlos, a velocidade do
  - do chute é 200 km por hora, e
  - do jogador é 20 km por hora;
- para certo Frederico, a velocidade do
  - do chute é 100 km por hora, e
  - do jogador é 20 km por hora

Por exemplo, poderíamos dar,

- *peso* 10 à *velocidade do jogador*, e
- *peso* 1 à *velocidade do chute.*

Obtemos,

- para Neymar a pontuação $1 \cdot 100 + 10 \cdot 40 = 500$
- para Roberto Carlos a pontuação $1 \cdot 200 + 10 \cdot 20 = 400$
- para Frederico a pontuação $\cdot 100 + 10 \cdot 20 = 300$

Se Neymar e Roberto Carlo estão na seleção nacional, mas Frederico não, então, por exemplo, um limiar de 320 modela isto.

Geometricamente, - os dois atributos de um jogador correspondem a um *ponto* no plano cartesiano, e - os pesos e o limiar a uma *reta* que separa entre os jogadores na seleção nacional e o resto.

Aplicaremos em Secção 4.1 o método geométrico do otimizador `Perceptron`, ou, um nome mais expressivo, *Discriminador Linear* que busca (ou *aprendre*) esta reta.

Observamos que estes pesos separam perfeitamente para a nossa amostra, mas provavelmente, estes mesmos pesos falham com outra. Por exemplo, os pesos obtidos pela seleção pelo Filipão (em 2006) são bem diferentes dos pelo Tite (em 2018). Isto toca a questão do aprendizado discutida em Secção 2: Quando podemos dizer que a amostra é típica e nos levou a uma boa escolha dos pesos?

2 O que é aprendizado?

Recordemo-nos de que formalmente, em um problema de aprendizado (com supervisionamento), temos os *dados*, isto é,

- a **entrada** X
- a **saída** Y
- a *amostra* $\mathrm{D} = \{x_1, \ldots, x_\mathrm{N}\}$ (= um conjunto finito em X sobre o qual a *função alvo* é conhecida, isto é, $y_1 = y(x_1), \ldots, y_\mathrm{N} = y(x_\mathrm{N})$ são conhecidos, e
- a **função alvo** (ou *divina*) $y\colon \mathrm{X} \to \mathrm{Y}$ (= a função ótima, que sempre acerta, a ser aproximada)

e o *método*, isto é,

- as **hipóteses** $\mathrm{H} = \{h\colon \mathrm{X} \to \mathrm{Y}\}$ (= um conjunto de funções que aproximam y), e
- um **algoritmo** A que escolhe uma função h em H que aproxima" y (e em particular D) o mais possível.

2.1 Erro

A escolha da proximidade, ou melhor, da distância mais apropriada do *erro*, isto é, entre os valores da hipótese h e da função divina y, depende, entre outros, da saída Y.

O que significa a mais próxima de y?. Esta escolha faz parte da definição do algoritmo A: Por exemplo,

- quando $\mathrm{Y} = \{\pm 1\}$ é binária, isto é, existem os dois erros $h(x) = 1$, mas $y(x) = -1$ e vice versa $h(x) = -1$, mas $y(x) = 1$,
  - podemos atribuir a cada erro a mesma importância, isto é, definir a distância pelo número de erros

$$\mathrm{E}_\mathrm{D}(h) = \#\{\text{erros}\}, \quad \text{ou}$$

– podemos distinguir entre as gravidades w e v dos erros (por exemplo, quando pensamos na deteção de uma doença grave e contagiosa como HIV, é menos grave detetar à toa, um falso positivo w, do que deixá-la passar impercebida, v), isto é, definir o erro por

$$\mathrm{E_D}(h) = w \cdot \#\{\text{erros do primeiro tipo}\} + v \cdot \#\{\text{erros do segundo tipo}\}.$$

- quando $\mathrm{Y} = \mathbb{R}$, gostaríamos de tomar em conta o tamanho da diferença $h(x) - y(x)$ em cada ponto x; usualmente, usa se a diferença quadrática a este fim, isto é

$$\mathrm{E_D}(h) = [h(x_1) - y(x_1)]^2 + \cdots + [h(x_\mathrm{N}) - y(x_\mathrm{N})]^2.$$

O programador determina

- as hipóteses $\mathrm{H} = \{h \colon \mathrm{X} \to \mathrm{Y}\}$, e
- a medida do erro na amostra $\mathrm{E_D}$, e

em seguida

- o algoritmo A que busca a hipótese h que minimize $\mathrm{E_D}(h)$.

Porém, isto não quer dizer que h minimize o erro fora da amostra, isto é, se aproxime da função divina em todo lugar. Esta proximidade depende da escolha justa de H e $\mathrm{E_D}$.

Formalismo. Será a *função de custo* que capta todas as possíveis medidas de erro neste manuscrito. Quando a saída é binária, e as importâncias dos dois erros

- $f(x) = 1$, mas $y(x) = -1$ e, vice versa,
- $f(x) = -1$, mas $y(x) = 1$,

são iguais, basta contar os pontos em que f erra (isto é, difere de y)

$$\mathrm{E_D}(f) := \mathrm{P}(\{x \in \mathrm{D} \text{ tal que } f(x) \neq y(x)\})$$

isto é,

$$\mathrm{E_D}(f) = \#\{\text{ os pontos } x \text{ em D em que } f(x) \neq y(x)\}/\mathrm{N},$$

Porém, em geral precisamos de uma abordagem mais geral à avaliação da proximidade de f à função divina y:

- para uma saída binária $\mathrm{Y} = \{\pm 1\}$, isto é, os dois erros $f(x) = 1$, mas $y(x) = -1$ e, vice versa, $f(x) = -1$, mas $y(x) = 1$, queremos distinguir entre as gravidades w e v dos erros, ou
- para $\mathrm{Y} = \mathbb{R}$, queremos tomar em conta o tamanho da diferença $f(x) - y(x)$ em cada ponto x

A este fim, introduzimos a **função de custo** $e\colon \mathrm{Y} \times \mathrm{Y} \to [0,\infty[$ tal que

$$\mathrm{E}_{\mathrm{D}}(f) = e(f(x_1), y(x_1)) + \cdots + e(f(x_{\mathrm{N}}), y(x_{\mathrm{N}})).$$

A função de custo faz parte do algoritmo; isto é, a sua apropriada escolha é sob a responsabilidade do programador do algoritmo.

Por exemplo,

- Quando $\mathrm{Y} = \{\pm 1\}$, sim ou não, e queremos tomar em conta que o erro
  - que $f(x) = 1$ mas $y(x) = -1$ é muito mais grave
  - que $f(x) = -1$, mas $y(x) = 1$.

  Por exemplo, é muito mais grave um leitor de dedo num caixa eletrônico
  - aceitar uma impressão digital erroneamente como a dono do cartão de crédito

  do que
  - rejeitar erroneamente a do dono.

  Por exemplo, se o primeiro erro é 1000 vezes mais grave, isto é,

$f(x) \setminus y(x)$	1	−1
1	0	1000
−1	1	0

  obtemos

$$\begin{aligned} &\mathrm{E}_{\mathrm{D}}(h) \\ =&1000 \cdot \mathrm{P}_{\mathrm{D}}(f(x) = 1 \text{ e } y(x) = -1) + 1 \cdot \mathrm{P}_{\mathrm{D}}(f(x) = -1 \text{ e } y(x) = 1). \end{aligned}$$

Isto é, a função de custo

$$e(f(x),y(x))\colon \{(-1,-1),(1,1),(-1,1),(1,-1)\} \to \mathbb{R}$$

é dada por $(1,-1) \mapsto 1000$, $(-1,1) \mapsto 1$ (e 0 alhures).

- Quando $\mathrm{Y} = \mathbb{R}$, e queremos tomar em conta a distância entre $f(x)$ e $y(x)$, a escolha mais comum é o valor médio da *diferença quadrática*,

$$\begin{aligned}\mathrm{E_D}(f) &= \mathbf{E}_\mathrm{D}[(f-y)^2] \\ &= [(f(x_1)-y(x_1))^2 + \cdots + (f(x_1)-y(x_1))^2]/\mathrm{N}\end{aligned}$$

  isto é, $e(f(x),y(x)) := [f(x)-y(x)]^2$. Usa-se, por exemplo, na regressão linear. Observamos

  – que o quadrado $\cdot^2$ (em vez do valor absoluto $|\cdot|$) penaliza poucos pontos muito distantes mais do que muitos pontos pouco distantes; por exemplo, 1 erro de uma distância de 3 mais do que 3 erros de uma distância de 1;
  – que se $\mathrm{Y} = \{\pm 1\}$, então $[f(x)-y(x)]^2 = 1$ se e tão-somente se $f(x) \neq y(x)$. Isto é, a função de custo para funções reais generaliza a função de custo de contagem (isto é, que conta os erros) de para funções cujos valores são binárias.

- Quando $\mathrm{Y} = [0,1]$, isto é, y é uma (media de) probabilidade, a melhor escolha para a hipótese $\mathrm{P} = f$ (parametrizada por certos parâmetros como a média e a variância da probabilidade) a partir da amostra D e os resultados $y(\mathrm{D})$ é, segundo o princípio da *máxima verossimilhança*, aquela escolha que maximiza a probabilidade $\mathrm{P}(y(\mathrm{D}) \mid \mathrm{D})$ da observação dos valores $y(\mathrm{D})$ dada a amostra D. Isto é,

$$\mathrm{E_D}(f) = \mathrm{P}(y(\mathrm{D}) \mid \mathrm{D})$$

  Para (x_1,y_1), …, $(x_\mathrm{N},y_\mathrm{N})$ independente e identicamente distribuídos, após aplicação do logaritmo (ao negativo), queremos minimizar o erro logarítmico (a verossimilhança logarítmica, ou `log-likelihood` em inglês)

$$\mathrm{E_D}(f) = -[\log \mathrm{P}(y_1|x_1) + \cdots + \log \mathrm{P}(y_\mathrm{N}|x_\mathrm{N})]/\mathrm{N}$$

  Isto é, a função de custo para cada exemplo é

$$e(f,y)(x) = -\log(\mathrm{P}y(x)|x).$$

- Como estamos mais interessados em minimizar o erro do que no seu valor exato, podemos aplicar uma função monótona m ao estimador do erro e, por exemplo, como acima, o logaritmo log (para diminuir valores muito grandes). Isto é, como m é monótona, a composição $m \circ e$ é mínima se e tão-somente se e é mínimo.

2.2 Provavelmente Aproximadamente Correto

O que significa **aprendizado**?. Isto é, qual é o nosso alvo no aprendizado de máquina?

Segundo o princípio PAC (Provavelmente Aproximadamente Correto) é, encontrar um algoritmo que consegue a partir de *quase qualquer* amostra (isto é, *Provavelmente*) encontrar uma função que *quase sempre* acerta (isto é, que é *Aproximadamente* Correta):

Uma hipótese mostra bom aprendizado se tira as conclusões certas a partir de uma amostra; isto é, mais importante do que acertar na amostra D é acertar na entrada inteira X:

Seja, por exemplo, a saída $Y = \{\pm 1\}$ binária, isto é, um problema de classificação binária. Dada a amostra $D = \{x_1, \ldots, x_N\}$ e uma função f, podemos calcular o erro **específico** por

$$\begin{aligned} E_D(f) &:= P(\{x \in D \text{ tal que } f(x) \neq y(x)\}) \\ &= \#\{ \text{ os pontos } x \text{ em D em que } f(x) \neq y(x)\}/N, \end{aligned}$$

isto é, a probabilidade que f erre sobre D. (Em Abu-Mostafa, Magdon-Ismail, e Lin (2012), denotado por E_{in} para significar *dentro* da amostra.) Por exemplo, o algoritmo `Perceptron`, se termina, consegue $E_D(f(D)) = 0$.

Mas, como y é desconhecida, *não* sabemos o erro **geral**,

$$E_X(f) := P(\{x \in X \text{ tal que } f(x) \neq y(x)\}),$$

isto é, a probabilidade que f erre sobre X. (Em Abu-Mostafa, Magdon-Ismail, e Lin (2012), denotado por E_{out} para significar *fora* da amostra.)

Veremos quando podemos minimizar esta probabilidade de erro. Porém, depende sempre da amostra, que pode ser atípica e levar mesmo um bom algoritmo à escolha de uma hipótese que tira conclusões erradas. Isto é, o algoritmo pode

só acertar com certa probabilidade na escolha de uma hipótese próxima (a função divina); isto é, é *provavelmente aproximadamente correto.*

Formalismo. Dada a amostra $D = \{x_1, ..., x_N\}$ e uma função f, podemos calcular o erro *específico*,

$$E_D(f)$$

isto é, a probabilidade que f erre sobre D. (Em Abu-Mostafa, Magdon-Ismail, e Lin (2012), denotado por E_{in} para significar *dentro* da amostra.) Por exemplo, o algoritmo `Perceptron`, se termina, consegue $E_D(f(D)) = 0$.

Mas, como y é desconhecida, *não* sabemos o erro *geral* (ou *erro de generalização*),

$$E_X(f)$$

isto é, a probabilidade que f erre sobre X. (Em Abu-Mostafa, Magdon-Ismail, e Lin (2012), denotado por E_{out} para significar *fora* da amostra.)

Se, por exemplo,

- $X = \{1, 2, 3, ..., 8\}$ e $Y = \{\pm 1\}$ (classificação binária), e
- D tem 5 pontos,

qualquer função que coincide em D com y é uma possível solução; sobram 3 valores a definir. Qual destas $\#Y^3 = 2^3$ extensões dos 5 aos 8 pontos é a mais próxima da função alvo? Não há resposta definitiva, mas provavelmente aproximadamente correta:

O problema é **apreensível por PAC** (= Provavelmente Aproximadamente Correto) se existe um algoritmo A que calcula para qualquer amostra $D = \{x_1, ..., x_N\}$ uma hipótese $h = h(D)$ em H que seja provavelmente aproximadamente correta, isto é:

Para todo (erro) $\epsilon > 0$ e (toda difidência) $\delta > 0$ e para toda distribuição (= medida de probabilidade) P, existe um N tal que para toda amostra D com pelo menos N exemplos independente e identicamente distribuídos

$$P(E_X(h(D)) < \epsilon) > 1 - \delta.$$

Para o algoritmo A ser eficientemente computável, é mais exatamente requerido que $N \geq P(1/\epsilon, 1/\delta)$ para um polinómio P, chamado de *complexidade de amostra* do algoritmo A.

Recordemo-nos de que

- a integral $E_X(h(D)) = \int_X e(h(D), y)dP$ mede o erro geral entre $h(D)$ e y ($< \epsilon$ = Aproximadamente Correto), e
- a probabilidade $P(E_X(h(D)) < \epsilon)$ mede o conjunto (em $X \times \cdots \times X$) em que a ênupla $D = (x_1, \ldots, x_N)$ implica uma escolha $h(D)$ pelo algoritmo A que é próxima de y, isto é, tal que o erro $E_X(h(D)) < \epsilon$; como é $> 1 - \delta$, é, em palavras, (altamente) provável.

Isto é, **provavelmente** a amostra leva a uma hipótese **aproximadamente correta**. Quer dizer, se a amostra é suficientemente típica (o que vale com uma probabilidade $> 1 - \delta$, isto é, **provavelmente**), então a probabilidade do erro é pequena, $< \epsilon$, isto é, a função é **aproximadamente** correto.

Em Secção 3, dados ϵ e $\delta > 0$, calcularemos para uns problemas de aprendizado o tamanho N da amostra D que garante que

$$P(E_X(h(D)) < \epsilon) > 1 - \delta.$$

2.3 Treinar e Testar

Como saber se um método aprende bem, isto é, se a hipótese h escolhida pelo algoritmo A a partir de D tem um erro geral $E_X h$ pequeno?

Holdout. Na prática, uma abordagem é o *holdout*, isto é, não usar toda a amostra D para a computação da melhor escolha h por A, mas divide aleatoriamente D em dois subconjuntos D′ e D″, por exemplo, 2/3 dos exemplos em D′ e 1/3 dos exemplos em D″ para

- usar o conjunto D′ como amostra para a computação da melhor hipótese h, o *conjunto de treinamento*, e
- usar o conjunto D″ para verificar a hipótese h, o *conjunto de teste.*

No treinamento, minimizamos $E_{D'}(h)$ (sobre todas as hipóteses h). No teste, verificamos que $E_{D''}(h)$ é pequeno.

Se a saída é finita, $Y = \{1, 2, \ldots, n\}$, então o erro de teste $E_{D''}(h)$ é visualizado por uma *matriz de confusão*, por exemplo, para $n = 3$ com 10 exemplos de cada tipo 1, 2 e 3,

$h(x) \setminus y(x)$	1	2	3
1	7	2	1
2	1	8	1
3	1	0	9

onde

- as casas na diagonal indica o número de exemplos que foram acertados e
- as outras o número dos que foram confundidos.

Random Subsampling. Se o alvo do programador, em vez de encontrar pelo algoritmo *a melhor hipótese* a partir de uns exemplos (e em seguida medir o erro da hipótese para outros exemplos), é *medir a média* dos erros das hipóteses encontradas pelo algoritmo (isto é, em vez de medir a acuraria da hipótese, medir a acuraria do algoritmo), então se aplica, em vez do holdout, o *random subsampling* que varia o conjunto de teste: Divide D em, no mínimo, três subconjuntos D′, D″ e D‴ e

- usa D′ como conjunto de teste (e D″ ∪ D‴ como conjunto de treinamento),
- usa D″ como conjunto de teste (e D′ ∪ D‴ como conjunto de treinamento), e
- usa D‴ como conjunto de teste (e D′ ∪ D″ como conjunto de treinamento).

O nosso método. Neste manuscrito, porém, só treinamos, e *não* testamos: Confiaremos em que o método, isto é, as escolhas

- de H,
- da medida de erro E, e
- do algoritmo A

são suficientemente apropriadas para o problema de aprendizado aprenda bem, isto é, que a minimização do erro da amostra $\mathrm{E}_{\mathrm{D}}h$ (entre todas as h em H) garante um erro geral $\mathrm{E}_{\mathrm{X}}h$ suficientemente pequeno (sem avaliar $\mathrm{E}_{\mathrm{T}}h$ para um conjunto de teste T diferente de D).

3 Quão bem aprendemos?

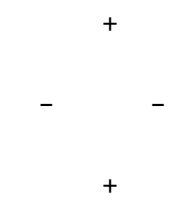
O que é aprendizado? Afinal, queremos encontrar a partir de uma *amostra*, uns exemplos (por exemplo, clientes antigos de um banco) uma função simples, a *hipótese*, que prediz os valores em que nos interessamos (por exemplo, a quantia de um crédito) para outras entradas (isto é, futuros clientes).

Para o algoritmo acertar, quer dizer, fazer provavelmente um erro pequeno, precisamos muitos exemplos, clientes antigos. Nesta Seção descobrimos

- para quais problemas o algoritmo consegue provavelmente acertar, e
- caso consiga, quantos exemplos precisa para isto.

Para isto, usamos uma estimativa geral, a desigualdade de Hoeffding (estatístico finlandês) que inicialmente só se aplicou para um número finito de hipóteses. Infelizmente, na prática o número de hipóteses é infinito. Por exemplo, no `Perceptron` no plano, as hipóteses são todas as retas.

Por isso, restringimos as hipóteses da entrada inteira infinita X à amostra finita D. Por exemplo, no `Perceptron`, restringimos aos valores das retas sobre N pontos no plano. Vimos que já para $N = 4$, tem valores de pontos que nenhuma reta consegue reproduzir, quer dizer, não existe reta que os separe, por exemplo:



Quando um tal número como $N = 4$ existe em que as hipóteses são insuficientes para reproduzir uma classificação da amostra, diremos que a *dimensão (de Vapnik-Chervonenkis) d* das hipóteses é finita. Neste caso, podemos, para qualquer erro ϵ e para qualquer confiança δ, por exemplo, $\epsilon = 0{,}1$ e $\delta = 0{,}1$, garantir que a hipótese escolhida pelo algoritmo a partir de uma amostra com N exemplos faça um erro $< \epsilon = 0{,}1$ com uma probabilidade de $1 - \delta = 90\%$.

O resultado teórico é muito folgado, pois muito geral: Na teoria, $N = 10.000d$, na prática $N = 10d$. Por exemplo, para o `Perceptron` no plano, $d = 3$.

3.1 Defeito

Dada a amostra $\mathrm{D} = \{x_1, \ldots, x_\mathrm{N}\}$ com os seus valores $\{y_1, \ldots, y_\mathrm{N}\}$ e uma hipótese h, podemos calcular o seu erro **específico**,

$$\mathrm{E}_\mathrm{D}(h);$$

contudo, como y é desconhecida, *não* o seu erro **geral**,

$$\mathrm{E}_\mathrm{X}(h),$$

cuja minimização é o alvo final. Para aproximá-lo, introduzamos o **defeito**

$$\Delta_\mathrm{D} h := \mathrm{E}_\mathrm{X}(h) - \mathrm{E}_\mathrm{D}(h)$$

que quantifica quão bem o algoritmo aprende, tira conclusões gerais certas a partir da amostra D: Para limitar $\mathrm{E}_\mathrm{X}(h)$, precisamos, pela desigualdade triangular

$$|\mathrm{E}_\mathrm{X}(h)| \leq |\Delta_\mathrm{D} h| + |\mathrm{E}_\mathrm{D}(h)|,$$

de limitar

$$|\Delta_\mathrm{D} h| \quad \text{e} \quad |\mathrm{E}_\mathrm{D}(h)|.$$

Isto é, a viabilidade de aprender corresponde à viabilidade de minimizar

- o defeito $\Delta_\mathrm{D} h$, e
- o erro específico $\mathrm{E}_\mathrm{D}(h)$.

Para simplificar a exposição, seja daqui em diante $\mathrm{Y} = \{\pm 1\}$, problema de classificação binária, e o erro

$$\mathrm{E}_\mathrm{D}(h) = \mathrm{P}(h \neq y) = \#\{x \text{ em D tais que } h(x) \neq y(x)\}/\#\mathrm{D}$$

dado pela probabilidade de errar.

3.2 Desigualdade de Hoeffding

Teorema. Dada uma função $h\colon \mathrm{X} \to \mathrm{Y}$, um limiar de erro $\epsilon > 0$ e um número de exemplos N, a **desigualdade de Hoeffding** estima a probabilidade do defeito ser $> \epsilon$ por

$$\mathrm{P}(|\Delta_\mathrm{D} h| > \epsilon) \leq 2\mathrm{e}^{-2\epsilon^2 \mathrm{N}}.$$

Isto é, dada uma hipótese $h\colon \mathrm{X} \to \mathrm{Y}$, sabemos limitar a *probabilidade* do evento

$$\{\mathrm{D} \text{ em } \mathrm{X} \times \cdots \times \mathrm{X} \text{ tal que } |\Delta_{\mathrm{D}} h| < \epsilon\}.$$

de uma amostra $\mathrm{D} = (x_1, \ldots, x_{\mathrm{N}})$ *atípica*, isto é, em que o erro de h difira do erro geral de h por mais de ϵ, por um grande número de exemplos N.

Num problema de aprendizado, dada uma função $h\colon \mathrm{X} \to \mathrm{Y}$, um limiar de erro $\epsilon > 0$ e um número de exemplos N, queremos limitar o risco

$$\mathrm{P}(\mathrm{E}_{\mathrm{X}} h(\mathrm{D}) > \epsilon)$$

que uma amostra D leva o algoritmo a uma hipótese $h(\mathrm{D})$ errada, isto é, com $\mathrm{E}_{\mathrm{X}} h(\mathrm{D}) > \epsilon$. Como

$$\mathrm{P}(\mathrm{E}_{\mathrm{X}} h(\mathrm{D}) > \epsilon) = \mathrm{P}(\mathrm{E}_{\mathrm{D}} h(\mathrm{D}) + \Delta_{\mathrm{D}} h(\mathrm{D}) > \epsilon);$$

e como $\mathrm{E}_{\mathrm{D}} h(\mathrm{D})$ é conhecido, queremos limitar $\mathrm{P}(|\Delta_{\mathrm{D}} h(\mathrm{D})| > \epsilon)$, isto é, a probabilidade do evento

$$\{\mathrm{D} \text{ em } \mathrm{X} \times \cdots \times \mathrm{X} \text{ tal que } |\Delta_{\mathrm{D}} h(\mathrm{D})| > \epsilon\} \quad (\dagger)$$

de uma amostra $\mathrm{D} = (x_1, \ldots, x_{\mathrm{N}})$ *atípica*, isto é, em que o erro de $h(\mathrm{D})$ difira do erro geral de $h(\mathrm{D})$ por mais de ϵ.

Para estimar $\mathrm{P}(|\Delta_{\mathrm{D}} h(\mathrm{D})| > \epsilon)$ pela aplicação da desigualdade de Hoeffding, observamos que

- no evento da desigualdade de Hoeffding, a função é fixa, independente de D, enquanto
- no evento (†), a função varia com D!

Em geral, sem conhecimento do algoritmo que escolhe $h(\mathrm{D})$ a partir de D, só podemos limitar a probabilidade de $|\Delta_{\mathrm{D}} h(\mathrm{D})| > \epsilon$ para $h(\mathrm{D})$ pela probabilidade que $|\Delta_{\mathrm{D}} h| > \epsilon$ para uma h em H. Isto é, que exista h em H com $|\Delta_{\mathrm{D}} h| > \epsilon$; em fórmulas

$$\begin{aligned} &\{\mathrm{D} \text{ em } \mathrm{X} \times \cdots \times \mathrm{X} \text{ tal que } |\Delta_{\mathrm{D}} h(\mathrm{D})| > \epsilon\} \\ \subseteq &\bigcup_{h \text{ em } \mathrm{H}} \{\mathrm{D} \text{ em } \mathrm{X} \times \cdots \times \mathrm{X} \text{ tal que } |\Delta_{\mathrm{D}} h(\mathrm{D})| > \epsilon\}. \end{aligned}$$

Se $\mathrm{H} = \{h_1, \ldots, h_{\mathrm{M}}\}$ é finito, então

$$\mathrm{P}(|\Delta_{\mathrm{D}}(h(\mathrm{D}))| > \epsilon) \leq \mathrm{P}(|\Delta_{\mathrm{D}}(h_1)| > \epsilon) + \cdots + \mathrm{P}(|\Delta_{\mathrm{D}}(h_{\mathrm{M}})| > \epsilon) \leq \mathrm{M} \cdot 2\mathrm{e}^{-2\epsilon^2 \mathrm{N}},$$

onde, na última desigualdade, como $h_1, \ldots, h_M$ não variam com D, aplicamos a cada um dos M parcelas a mesma desigualdade de Hoeffding, obtendo na soma o fator M.

Observação. Quanto maior #H = M, tanto mais facilmente o algoritmo consegue diminuir

$$P(|E_D(h(D))| > \epsilon),$$

mas, também, tanto maior o limite da desigualdade

$$P(|\Delta_D(h(D))| > \epsilon) \leq M \cdot 2e^{-2\epsilon^2 N}.$$

Não há saída deste dilema entre o erro geral e o erro específico, só podemos procurar um equilíbrio entre os dois.

3.3 A dimensão de Vapnik-Chervonenkis

Recordemo-nos de que para uma hipótese $h\colon X \to Y$ e uma amostra D, é a diferença, o *defeito*

$$\Delta_D h = E_D(h) - E_X(h),$$

a diferença entre

- o *erro específico* $E_D(h)$ (sobre a amostra D) e
- o *erro geral* $E_X(h)$ (sobre X).

quantifica quão bem o algoritmo aprende, acerta as conclusões gerais a partir de exemplos. Logo, num problema de aprendizado, dada $\epsilon > 0$, queremos limitar a probabilidade

$$P(|\Delta_D h(D)| > \epsilon)$$

para a hipótese $h(D)$ em H que é *escolhida a partir de* D pelo algoritmo A; em particular, que vária com D. Como A é desconhecido, logo $h(D)$, precisamos de limitar o supremo de todas as hipóteses

$$P(|\Delta_D h(D)| > \epsilon) \leq P(\sup\{|\Delta_D(h)| : h \text{ em } H\} > \epsilon).$$

Observamos

$$\{D : \sup\{|\Delta_D h| : h \text{ em } H\} > \epsilon\} = \bigcup_{h \text{ em } H} \{D : |\Delta_D h| > \epsilon\}.$$

Por isso, se $\mathrm{H} = \{h_1, ..., h_\mathrm{M}\}$ é finito, esta probabilidade pode ser estimada, da maneira mais grossa, por

$$\mathrm{P}(|\Delta_\mathrm{D} h(\mathrm{D})| > \epsilon) \leq \mathrm{P}(\bigcup_{h \text{ em } \mathrm{H}} \{\mathrm{D} : |\Delta_\mathrm{D} h| > \epsilon\}) \leq \mathrm{M} \cdot 2\mathrm{e}^{-2\epsilon^2 \#\mathrm{D}}.$$

Porém, para H infinito, precisamos de uma maneira mais fina para estimar $\mathrm{P}(|\Delta_\mathrm{D} h(\mathrm{D})| > \epsilon)$ que toma em conta as grandes interseções dos conjuntos $\{\mathrm{D} : |\Delta_\mathrm{D} h| > \epsilon\}$ para as h em H que diferem pouco.

Vamos agora conhecer a "dimensão" do conjunto de hipóteses H, que permite limitar o defeito para H infinito.

O número efetivo de hipóteses sobre uma amostra. Para limitar a probabilidade de um alto defeito,

- em vez do número total #H *infinito* de hipóteses, isto é, como funções sobre X inteiro,
- estudamos o crescimento o número finito #H(D) das suas restrições a amostras D (conjuntos finitos em X) crescentes.

Definição: O conjunto das restrições a D das hipóteses h em H, as *dicotomias geradas por* H, é denotado por

$$\mathrm{H}(\mathrm{D}) := \{h_{|\mathrm{D}} : h \in \mathrm{H}\}$$

A *função de crescimento* m_H para H (sobre $\mathbb{N}$) é definida por

$$m_\mathrm{H}(\mathrm{N}) := \max\{\#\mathrm{H}(\mathrm{D}) : \mathrm{D} \text{ uma amostra em X com } \#\mathrm{D} = \mathrm{N}\}.$$

Se #D = N, então

$$\#\mathrm{H}(\mathrm{D}) \leq \#\mathrm{Y} \cdots \#\mathrm{Y} = 2^\mathrm{N}$$

onde os fatores #Y, ..., #Y correspondem aos dois possíveis valores $\{\pm 1\}$ de $x_1, \ldots, x_\mathrm{N}$; logo,

$$m_\mathrm{H}(\mathrm{N}) \leq 2^\mathrm{N}.$$

Observamos nos exemplos seguintes de m_H para H diferentes que

- o seu valor máximo $m_\mathrm{H}(\mathrm{N}) = 2^\mathrm{N}$ é frequentemente só alcançado para argumentos pequenos N = 1,2,3,..., e

- quanto maior o número dos parâmetros que definem H, tanto mais argumentos (pequenos sucessivos) alcançam este máximo:

Exemplos:

- Seja $X = \mathbb{R} \times \mathbb{R}$ o plano e H o conjunto das hipóteses do `Perceptron`. Observamos que $m_H(3) = 8$ é máximo, mas para $N = 4$ há para qualquer amostra D uma dicotomia $D \to \{\pm 1\}$ que não pode ser gerada por H. Isto é, sempre podemos rotular quatro pontos no plano tal que eles não sejam linearmente separáveis. (Isto é, o `Perceptron` não termina.)

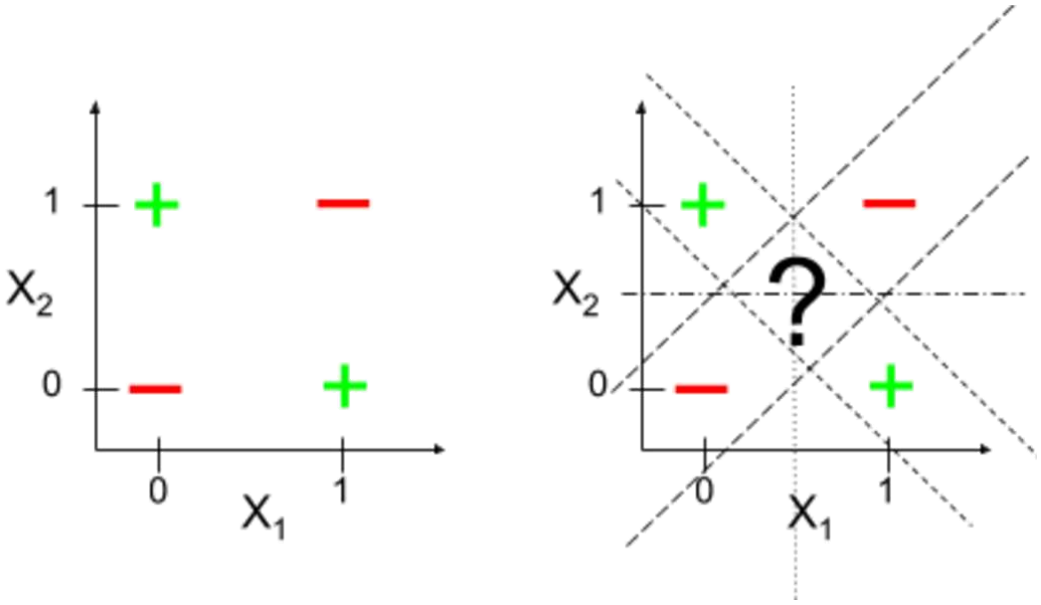


Figura 4: Quatro pontos linearmente inseparáveis

  Com efeito, $m_H(4) = 14$.

- Para $X = \mathbb{R}$ e $H := \{h = \text{signal}(\cdot - a) : a \in \mathbb{R}\}$, temos $m_H(N) = N + 1$: como N pontos cindem a reta em $N+1$ segmentos, há $N+1$ escolhas para a relativo a estes segmentos.
- Para $X = \mathbb{R}$ e $H := \{h = I[a,b] : a,b \in \mathbb{R}\}$ com $I[a,b](x) = 1$ se x está no intervalo $[a,b]$ (e = −1 caso contrário), temos $m_H(N) = N(N+1)/2 + 1$: Como N pontos cindem a reta em $N+1$ regiões, há
  - a escolha que o intervalo está inteiramente incluso em um destes segmentos, e

– mais $N + 1$ escolhas para a primeira extremidade e N escolhas sobrantes para a segunda extremidade do intervalo, sem importar a ordem cronológica da escolha das extremidades; isto é, $N(N+1)/2$ escolhas.

- Seja $X = \mathbb{R} \times \mathbb{R}$ o plano e H a família dada pelos conjuntos *convexos* (isto é, para cada dois pontos no conjunto, também o segmento entre eles fica no conjunto) em X, isto é, $h(x) = 1$ se x está no conjunto convexo que corresponde a h. Para mostrar que $m_H(N)$ é máximo, olhamos a amostra D dada por N pontos na borda de um círculo. Para qualquer dicotomia, conecta todos os pontos do tipo +1 por um polígono convexo (assim que todos os pontos do tipo -1 estão fora dele).

Se $m_H(N) = 2^N$, isto é, se existe uma amostra D tal que $\#H(D) = 2^N$, dizemos que H *estilhaça* D. Caso contrário, se $m_H(k) < 2^k$, isto é, se nenhuma amostra de cardinalidade k pode ser estilhaçada por H, o número natural k é um *ponto de interrupção* para H.

Exemplos: Nos exemplos acima, o *menor* ponto de interrupção é $k_0 = 4, 2, 3$ e ∞. Para obter k_0, usa-se uma rotulagem (= dicotomia) sobre os pontos que é inestilhaçável, isto é, não pode provir de uma hipótese.

Definição: A *dimensão de Vapnik-Chervonenkis* é

$$d_{VC}(H) := \max\{N \text{ tal que } m_H(N) = 2^N\}$$

e $d_{VC}(H) := \infty$ caso $m_H(N) = 2^N$ para todos os N. Isto é, $d_{VC}(H) = k_0 - 1$ para k_0 o *menor* ponto de interrupção para H.

Para calcular $d_{VC}(H)$, limitamos $d_{VC}(H)$ abaixo e acima: Por definição,

$$d_{VC}(H) \geq N \iff \text{uma amostra D com } \#D = N \text{ é estilhaçada por H,}$$

ou, equivalentemente, em contra-posição,

$$d_{VC}(H) < N \iff \text{nenhuma amostra D com } \#D = N \text{ é estilhaçada por H.}$$

Exemplo: Para o `Perceptron` sobre $X = \{1\} \times \mathbb{R}^d$, temos $d_{VC}(H) = d + 1$. Por exemplo, para $d = 2$, temos $d_{VC}(H) = 3$. Em geral, como heurística, $d_{VC}(H)$ é proporcional ao número dos parâmetros que definem as funções em H.

Cota do número efetivo de hipóteses sobre uma amostra. Dada uma amostra D com N elementos, $\epsilon > 0$, recordemo-nos da desigualdade de Hoeffding, para um conjunto de hipóteses finito de cardinalidade #H = M,

$$\mathrm{P}(|\Delta_{\mathrm{D}}(h(\mathrm{D}))| \leq \epsilon) \geq 1 - 2\mathrm{M} \cdot 2^{-\mathrm{N}2\epsilon^2}.$$

Logo, com $\delta = 2\mathrm{M} \cdot 2^{-\mathrm{N}2\epsilon^2}$, resolvendo para ϵ, obtemos

$$\mathrm{P}(|\Delta_{\mathrm{D}}(h(\mathrm{D}))| \leq [(1/2\mathrm{N}) \cdot \log(2\mathrm{M}/\delta)]^{1/2}) \geq 1 - \delta.$$

Para H infinito, queremos substituir M pelo número $m_{\mathrm{H}}(\mathrm{N})$ (que depende de N) e limitá-lo. Revela-se

- que a fórmula de Hoeffding precisa de uma leve modificação para esta substituição valer,
- que, se há um ponto de interrupção, então $m_{\mathrm{H}}(\mathrm{N})$ pode ser limitado por um polinômio em N.

Teorema(Lema de Sauer): Se há k em $\mathbb{N}$ tal que $m_{\mathrm{H}}(k) < 2^k$, então

$$m_{\mathrm{H}}(\mathrm{N}) \leq 1 + \mathrm{N} + \mathrm{N}(\mathrm{N}-1)/2 + \cdots + \mathrm{N}(\mathrm{N}-1)\cdots(\mathrm{N}-k+2)/(k-1)!$$

para todos os N.

Em particular, para $d = \mathrm{d}_{\mathrm{VC}}(\mathrm{H})$ a dimensão de Vapnik-Chervonenkis,

$$m_{\mathrm{H}}(\mathrm{N}) \leq 1 + \mathrm{N} + \mathrm{N}(\mathrm{N}-1)/2 + \cdots + \mathrm{N}(\mathrm{N}-1)\cdots(\mathrm{N}-d+1)/d!$$

para todos os N. Isto é, $m_{\mathrm{H}}(\mathrm{N})$ é limitado por um polinômio de grau d.

Corolário. Se $d = \mathrm{d}_{\mathrm{VC}}(\mathrm{H}) < \infty$, então

$$m_{\mathrm{H}}(\mathrm{N}) \leq \mathrm{N}^d + 1$$

Demonstração: Por indução em d.

Cota de erro. **Teorema**(Cota de Generalização de VC = o mais importante resultado matemático na teoria de aprendizado): Para $\delta > 0$,

$$P(|\Delta_D(h)| \leq [(8/N)\log(4m_H(2N)/\delta)]^{1/2}) \geq 1 - \delta.$$

Corolário: Se $d = d_{VC}(H) < \infty$, então para qualquer ϵ e $\delta > 0$ existe N tal que, para toda amostra D com $\#D \geq N$,

$$P(|\Delta_D(h)| \leq \epsilon) > 1 - \delta.$$

Mais exatamente, para todo N com

$$N \geq 8/\epsilon^2 \log([4(2N)^d + 1]/\delta).$$

Demonstração: Se $d_{VC}(H) < \infty$, então pelo Lema de Sauer,

$$m_H(N) \leq N^d + 1$$

Como $\log(x)/x \to 0$, logo $\log(p(x))/x \to 0$ para qualquer polinômio $p(x)$, obtemos

$$[(8/N)\log(4m_H(2N)/\delta)]^{1/2} \to 0 \quad \text{para } N \to \infty.$$

Mais exatamente, podemos resolver

$$[(8/N)\log(4m_H(2N)/\delta)]^{1/2} \leq \epsilon$$

para N, obtendo a desigualdade para N acima.

Vemos que

$$[(8/N)\log(4m_H(2N)/\delta)]^{1/2} \to 0 \quad \text{para } N \to \infty.$$

exatamente porque $m_H(2N)$ é polinomial, e não exponencial, 2^N. Neste caso, valeria $\log(2^N) = c \cdot N$; em particular, não converge a 0.

Observação. Porém, a desigualdade

$$N \geq 8/\epsilon^2 \log([4(2N)^d + 1]/\delta)$$

é implícita, isto é, N aparece ao lado direto. Por isso, N precisa de ser aproximado iterativamente: Se $N_0 < f(N_0)$, onde $f(N_0)$ denote o lado direito, então

o crescimento monótono da função f permite aproximar o mínimo N tal que $N \geq f(N)$ pelos valores $f(N_0)$, $f(f(N_0))$, ...

Por exemplo, para $d = 3$, uma confidência $\delta = 0{,}1$ e erro $\epsilon = 0{,}1$, uma escolha $N = 10000$ leva a um valor no lado direto de $N = 21193$. Esta iteração converge rápido a um valor de $N \approx 30000$. Semelhantemente, para $d = 4$ e $d = 5$, obtém-se $N \approx 40000$ e $N = 50000$.

Isto é, o número de exemplos requerido é proporcional à dimensão de Vapnik-Chervonenkis com um fator de 10000. Na prática, um fator de 10 revelou-se suficiente (vide `Deep Learning` para um método de aprendizado que funciona bem na prática mas desafia o quadro teórico de Vapnik-Chervonenkis).

Contudo, apesar da cota demasiada ampla, as estimativas obtidas por Vapnik-Chervonenkis são

- as únicas que funcionam em geral, e outras abordagens gerais não levaram a melhores resultados, e
- já expõem fenômenos gerais como a proporcionalidade entre
  - o número de exemplos necessários para obter uma hipótese provavelmente aproximadamente correta, e
  - o número de parâmetros que definem as hipóteses, isto é, d_{VC}.

4 Hipóteses Lineares

Muitos problemas de aprendizado podem ser resolvidos por hipóteses que são funções lineares. Para três saídas diferentes,

- a saída binária, $\mathrm{Y} = \{\pm 1\}$
- a saída ilimitada, $\mathrm{Y} = \mathbb{R}$, e
- a saída probabilista, $\mathrm{Y} = [0,1]$

vamos explicar três algoritmos que escolhem entre tais hipóteses:

- o `Perceptron` ou Discriminador Linear,
- a Regressão Linear, e
- a Regressão Logística.

Por exemplo, um banco pergunta-se sobre um cliente:

- se ele deve receber um crédito (sim ou não)?
- qual é a quantia do crédito (um número real)?
- quão provável é que ele quite as contas (uma probabilidade) ?

Recordemo-nos de que em um problema de aprendizado, temos um algoritmo A que escolhe em função da amostra D a melhor hipótese h em H. A *melhor* hipótese quer dizer a hipótese que é mais próxima da função divina y. A *proximidade* de h mede-se pela função de erro *geral* $\mathrm{E}_{\mathrm{X}}(h)$, o erro sobre X. Como y é só conhecida sobre D (mas não sobre X), podemos só aproximá-lo pelo erro *específico* $\mathrm{E}_{\mathrm{D}}(h)$ sobre a amostra D.

Em todos os algoritmos lineares que serão apresentados,

- cada hipótese h corresponde a um vetor de pesos w em $\mathbb{R} \times \cdots \times \mathbb{R}$, mais exatamente
$$h = h_0(w^{\mathrm{T}} \cdot)$$
para uma função $h_0 \colon \mathbb{R} \to \mathbb{R}$ que
  - é $h_0 = \mathrm{sinal}$, o sinal, no `Perceptron`,
  - é $h_0 = \mathrm{id}$, a identidade, na Regressão Linear, e
  - é $h_0 = \theta = \exp/(1+\exp)$, a função logística, na Regressão Logística;
- o erro especifico $\mathrm{E}_{\mathrm{D}}(w)$ é uma soma finita nos valores $w^{\mathrm{T}}x_1, \ldots, w^{\mathrm{T}}x_{\mathrm{N}}$ sobre a amostra $\mathrm{D} = \{x_1, \ldots, x_{\mathrm{N}}\}$.

Em todos os algoritmos lineares que serão apresentados, contentamo-nos com um algoritmo A que para cada D escolhe o w que minimize esta soma finita $\mathrm{E}_{\mathrm{D}}(w)$ nos valores $w^{\mathrm{T}}x_1, \ldots, w^{\mathrm{T}}x_{\mathrm{N}}$.

Frequentemente, acrescentamos uma coordenada constante a X, isto é, $\mathrm{X} = \{1\} \times \mathbb{R} \times \cdots \times \mathbb{R}$ como artifício para aumentar a dimensão de H e preservar a dimensão de X. Isto permite dispensar de um parâmetro constante nos algoritmos, isto é, em vez de

$$(x_1, \ldots, x_d) \mapsto w_0 + w_1 x_1 + \cdots + w_d x_d,$$

basta-nos considerar

$$(x_1, \ldots, x_d) \mapsto w_1 x_1 + \cdots + w_d x_d,$$

pelo acréscimo de uma entrada $x_0 = 1$.

4.1 `Perceptron` (ou Discriminador Linear)

Formalizemos a questão de como decidir se um cliente consegue quitar um crédito a partir dos seus atributos (*features*) por comparação com as quitações históricas.

A ideia é pontuar o cliente, e conceder o crédito caso a pontuação foi suficientemente alta. Isto é, dar pontos aos clientes de um banco em relação aos seus atributos financeiros, por exemplo, o seu salário e os seus bens. Se a pontuação é suficientemente alta, o crédito lhe é concedia.

A questão surge como ponderar os atributos dos clientes, quais pesos dar aos seus atributos, e onde localizar o limiar a partir de que será aceito?

O algoritmo que busca estes pesos chama-se Perceptron: Mais abstratamente, se o problema de aprendizado é uma *classificação binária*, isto é

- $\mathrm{X} = \mathbb{R} \times \cdots \times \mathbb{R}$, e
- $\mathrm{Y} = \{\pm 1\}$,

podemos aplicar o método de aprendizado do `Perceptron`, em que

- $\mathrm{H} = \{h(x) = \mathrm{sinal}(w_1 x_1 + \cdots + w_d x_d - b) : w_1, \ldots, w_d, b \text{ em } \mathbb{R}\}$

com sinal(x) = 1 se $x > 0$ e -1 se $x < 0$. Isto é, uma função h em H é dada pelos

- seus **pesos** $w_1, \ldots, w_d$ e
- o seu **limiar** b (= a nossa pontuação mínima),

e, para x em X, vale

$$h(x) = \pm 1 \qquad \text{se e tão-somente se} \qquad w_1 x_1 + \cdots + w_d x_d \leqslant b.$$

Geometricamente, os pontos $x = (x_1, \ldots, x_d)$ que satisfazem a igualdade $w_1 x_1 + \cdots + w_d x_d = b$ formam um hiperplano que divide o espaço X nos dois semiespaços dos pontos x cuja soma é $\leqslant b$). Por exemplo, se X é o plano, a igualdade define uma reta e cada lado dela um semiplano. Se as coordenadas sejam denotadas por x e y, os pesos por w respetivamente $v \neq 0$, então a equação da reta como função no argumento x com valor y é dada por

$$wx + vy = b \text{ se, e tão-somente se, } y = \mathrm{W}x + \mathrm{B} \text{ com } \mathrm{W} = \frac{-w}{v} \text{ e } \mathrm{B} = \frac{b}{v}.$$

Por exemplo, os parâmetros $w = 2$, $v = 1$ e $b = 1$ correspondem à reta abaixo:

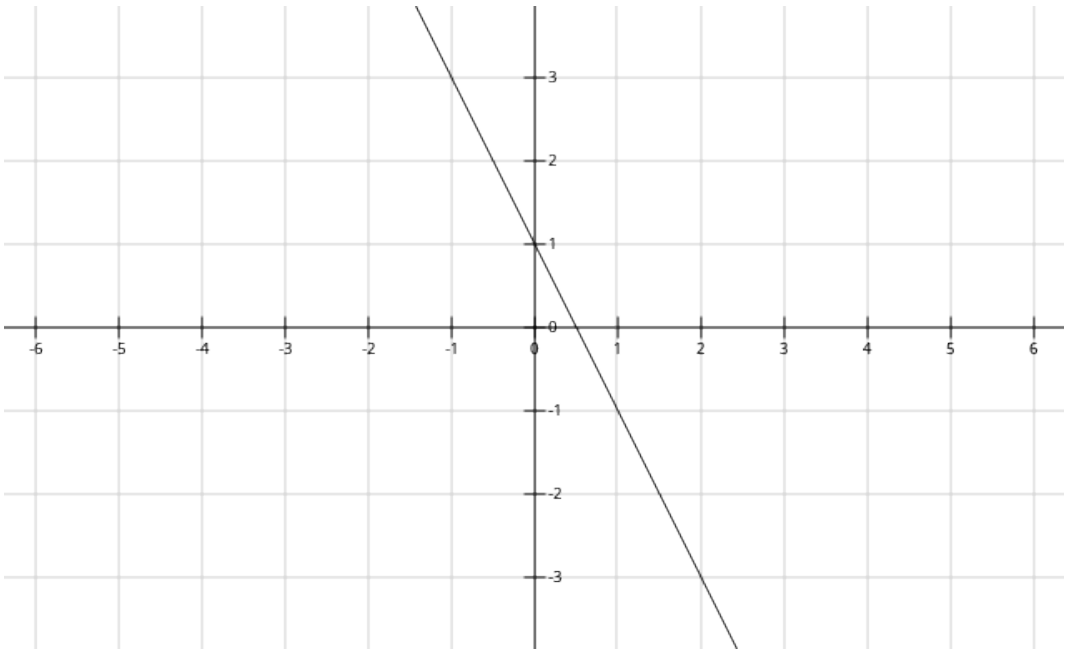


Figura 5: A reta $y = 1 - 2x$ divisa o plano nos dois semiplanos $y + 2x \leqslant 1$

No caso

- da concessão de um crédito, dependendo dos dados financeiros de um cliente e da importância de cada critério: Entre os pesos, revelam-se
  - uns *positivos*, por exemplo, os haveres e o salário,
  - outros *negativos*, por exemplo, as dívidas;
- da seleção de jogadores para a equipa nacional: Em Secção 1.5, concordamos que a velocidade do jogador importa bem mais do que a do seu chute; em números,
  - o peso da velocidade do jogador foi 10, e
  - o da velocidade do chute foi 1;
- da detecção de um e-mail como `Spam` dependendo de quais palavras o e-mail contém e as suas frequências em (e-mails de) `Spam` e em (e-mails de) `Ham`. Por exemplo, são pesos
  - *positivos* palavras de propaganda e de marcas, como `promoção`, `aumento` ou `Viagra`,
  - *negativos* palavras como conjunções hesitantes, por exemplo, `entretanto`, ou nomes raros de conhecidos, como `Elivélton`.

O algoritmo `Perceptron` aproxima os pesos e o limiar por iteração sobre todos os pontos no conjunto de dados exemplares D a uma solução h_0, quer dizer, tal que

$$h_0(x) = y(x) \qquad \text{para todo } x \text{ em D.}$$

Hipóteses. Facilitemos a formulação do problema do aprendizado: Para abreviar a notação da soma dos atributos pesados $w_1x_1 + \cdots + w_dx_d$, usamos o **produto escalar** entre o vetor $w = (w_1, ..., w_d)$ e $x = (v_1, ..., v_d)$ dado por

$$w^{\mathrm{T}}x := w_1x_1 + \cdots + w_dx_d$$

(A notação $\cdot^{\mathrm{T}}$ significa vetor *transposto*. Isto é, o vetor w^{T} é visto como vetor de *coluna* e x como vetor de *linha*. O produto escalar é o produto da *matriz* de uma coluna w^{T} com a *matriz* de uma linha x.) Pondo $x_0 = 1$ e $w_0 = -b$, vale

$$w_1x_1 + \cdots + w_dx_d - b = w_0x_0 + w_1x_1 + \cdots + w_dx_d$$

isto é, para $x = (1, x_1, ..., x_d)$ e $w = (-b, w_1, ..., w_d)$,

$$w_1x_1 + \cdots + w_dx_d - b = w^{\mathrm{T}}x.$$

Concluímos que

- $\mathrm{X} = \{1\} \times \mathbb{R} \times \cdots \times \mathbb{R}$, e
- $\mathrm{Y} = \{\pm 1\}$,

e

- $\mathrm{H} = \{h(x) = \text{sinal}(w^{\mathrm{T}}x) : w_0, w_1, \ldots, w_d \text{ em } \mathbb{R}\}$

com $\text{sinal}(x) = 1$ se $x > 0$ e -1 se $x < 0$. (Geometricamente, para $d = 2$,

- olhamos o plano com as coordenadas x e y como o hiperplano P dado pela equação $x_0 = z = 1$ no espaço de três coordenadas x,y e z,
- o hiperplano H no espaço dado pelos vetores ortogonais a w, e
- a interseção de H e P é a reta que separa os pontos.) Recordemo-nos de que

$$h(x) = \pm 1 \quad \text{se e tão-somente se} \quad w^{\mathrm{T}}x \lesseqgtr 0$$

e

$$w^{\mathrm{T}}x \lesseqgtr 0 \quad \text{se e tão-somente se} \quad w_1x_1 + \cdots + w_dx_d \lesseqgtr b.$$

Erro. O erro especifico $\mathrm{E_D}(h)$ que o algoritmo visa minimizar é o número dos pontos em que h erra,

$$\mathrm{E_D}(h) = \#\{x \text{ em D com } h(x) \neq y(x)\} = \sum_{x \text{ em D}} (h(x) - y(x))^2.$$

Na imagem correspondem:

- o valor de y à cor do ponto, e
- o valor de h ao lado da reta em que se encontra o ponto.

(Equivalentemente, o algoritmo visa minimizar $\mathrm{P_D}(h) := \mathrm{E_D}(h)/\mathrm{N}$, a probabilidade que h erre sobre D.) Com efeito, o algoritmo `Perceptron`

- ou consegue $\mathrm{E_D}(h) = 0$,
- ou oscila sem convergir.

Por isso, apresentaremos em seguida o algoritmo `Pocket` que remedia esta oscilação por iteração sobre o `Perceptron`. Assim, converge mesmo quando $\mathrm{E_D}(h) = 0$ é impossível.

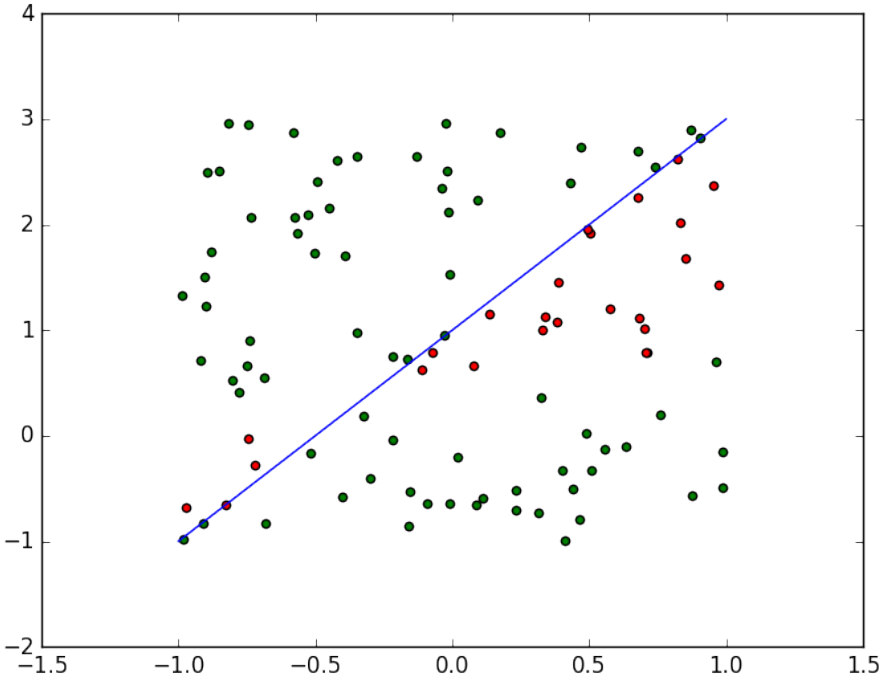


Figura 6: Separação de pontos pelas suas cores e por uma reta

Algoritmo. O algoritmo `Perceptron` determina o limiar e os pesos $w = (b, w_1, \ldots, w_d)$ iterativamente a partir da amostra D como segue:

Seja $t = 0, 1, 2, \ldots$ a iteração e $w(t)$ o valor de w na iteração t. Seja $x(t)$ em D um ponto erroneamente classificado e $y(t) := y(x(t))$ o seu valor correto. Formalmente, *erroneamente classificado* quer dizer

$$\text{sinal}(w(t)^{\mathrm{T}} x(t)) \neq y(t).$$

ou, equivalentemente,

$$y(t)(w(t)^{\mathrm{T}} x(t)) < 0.$$

O `Perceptron` põe

$$w(t+1) := w(t) + y(t)x(t)$$

Geometricamente, para obter a nova reta (= o novo limiar e os novos pesos) $w(t+1)$, o `Perceptron` desloca a reta na direção correta (isto é, a dada por $y(t)$) do ponto erroneamente classificado. Formalmente, há duas possibilidades:

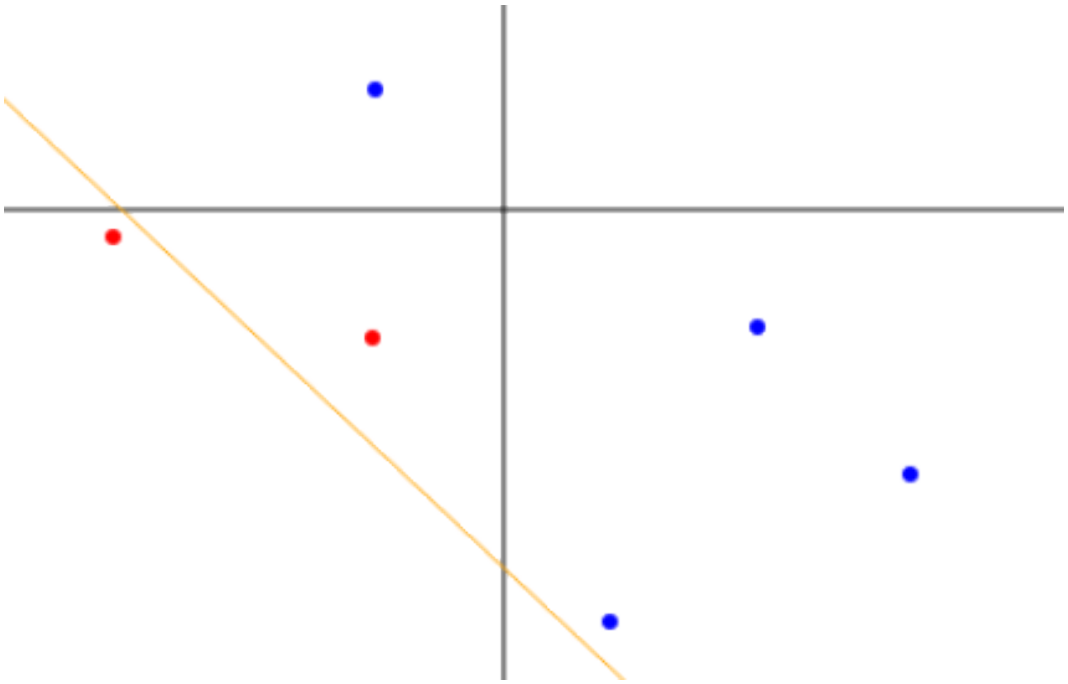


Figura 7: ponto erroneamente classificado

- ou $w(t)^{\mathrm{T}}x(t) > 0$ e $y(t) = -1$,
- ou $w(t)^{\mathrm{T}}x(t) < 0$ e $y(t) = +1$.

Como

$$w(t+1)^{\mathrm{T}}x(t) = w(t)^{\mathrm{T}}x(t) + y(t)x(t)^{\mathrm{T}}x(t)$$

e $v^{\mathrm{T}}v > 0$ para qualquer vetor v, vale

- se $w(t)^{\mathrm{T}}x(t) > 0$ e $y(t) = -1$, então $w(t+1)^{\mathrm{T}}x(t) < w(t)^{\mathrm{T}}x(t)$,
- se $w(t)^{\mathrm{T}}x(t) < 0$ e $y(t) = +1$, então $w(t+1)^{\mathrm{T}}x(t) > w(t)^{\mathrm{T}}x(t)$

Isto é, o valor da (nova) função hipotecada $h(t+1) := w(t+1)^{\mathrm{T}}\cdot$ é *pelo menos no ponto por enquanto erroneamente classificado $x(t)$ mais próximo do valor correto* (do que o da antiga função hipotecada $h(t)$). (Isto não quer dizer que a cada iteração o erro em outros pontos pode aumentar! Diante disto é estupefaciente que a função finalmente obtida é totalmente correta — dado que uma tal função existe.)

O `Perceptron` itera a correção do limiar e dos pesos $w(t) \to w(t+1)$ até que não haja mais pontos erroneamente classificados. Geometricamente, desloca a

reta até que separe as duas classes de pontos.

Código. Se denotamos a amostra por $x_1, \ldots, x_N$ e os seus valores sob a função divina y por $y_1, \ldots, y_N$, então, em pseudo-código, o algoritmo `Perceptron` é dado por

> Põe $w(0) := (0, \ldots, 0)$ e $t := 0$
> Repete
> Para $i = 1, \ldots, N$
> Se $y_i w(t)^T x_i < 0$, então põe $w(t+1) := w(t) + y_i x_i$
> Aumenta t por 1
> até $y_i w(t)^T x_i > 0$ para todo $i = 1, \ldots, N$
> Devolve $w(t)$

Em `Python`, para calcular os pesos e limiar w (inspirado pela postagem Perceptron in a few lines of `Python` code),

```
import numpy as np

X = np.array([
    [-2, 4],
    [4, 1],
    [1, 6],
    [2, 4],
    [6, 2]
])

Y = np.array([-1,-1,1,1,1])

def perceptron(X, Y):
    w = np.zeros(len(X[0]))
    epochs = 100

    for t in range(epochs):
        errors = 0
        for i in enumerate(X):
            if (np.dot(X[i], w)*Y[i]) <= 0:
                w = w + X[i]*Y[i]
```

```
                errors += 1
        if errors == 0:
            break
    return w
```

Para desenhar os pontos classificados pela reta separadora,

```
from matplotlib import pyplot as plt

for d, sample in enumerate(X):
    # Plot the negative samples
    if d < 2:
        plt.scatter(sample[0], sample[1], s = 120, marker = '_', linewidths = 2)
    # Plot the positive samples
    else:
        plt.scatter(sample[0], sample[1], s = 120, marker = ' + ', linewidths = 2)

# Print the line computed by perceptron ()
x2 = [w[0],w[1],-w[1],w[0]]
x3 = [w[0],w[1],w[1],-w[0]]

x2x3 =np.array([x2,x3])
X,Y,U,V = zip(*x2x3)
ax = plt.gca()
ax.quiver(X,Y,U,V,scale = 1, color = 'blue')
```

Exemplo em dois passos. Seja $d = 2$, e

- consista D nos dois pontos $p = (0,2)$ e $q = (1,1)$ com $y(p) = 1$ e $y(q) = -1$, e
- seja $w(0) = (0,0,2)$.

Observamos que $x = (x_0, x_1, x_2) = (1, x, y)$ satisfaz

$$w_0 x_0 + w_1 x_1 + w_2 x_2 = 0$$

se e tão-somente se

$$y = ax + b \quad \text{com } a = -w_1/w_2 \text{ e } b = -w_0/w_2.$$

Isto é, descrevemos a projeção dos vetores ortonormais ao vetor w aos seus últimos dois coordenados pela função (traslada) linear $y = ax + b$.

1. Como $w(0)^{\mathrm{T}} q = 2 > 0$ mas $y(q) < 0$, constatamos que q é erroneamente classificado e determinamos

$$w(1) = w(0) + y(q)q = (0,0,2) - (1,1,1) = (-1,-1,1).$$

   Isto é, $a = 1$ e $b = 1$.
2. Como $w(1)q = -1 > 0$ e $y(q) < 0$, constatamos que q é bem classificado (e p também) e o algoritmo termina.

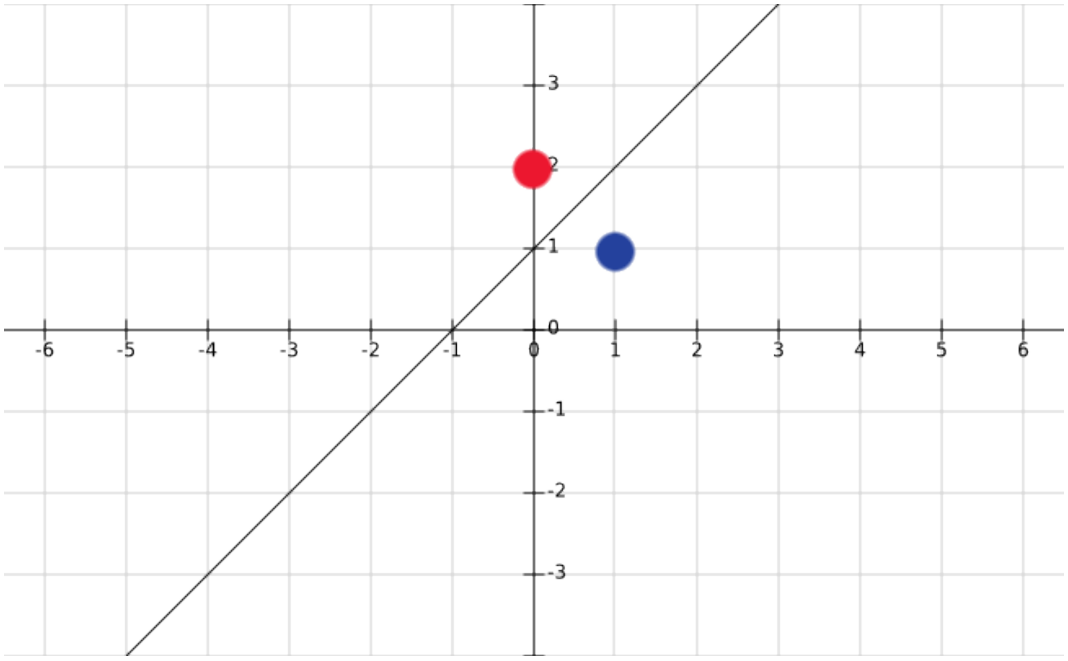


Figura 8: O exemplo dado pelos pontos p e q

Tempo de Execução.

- Se D não é linearmente separável, isto é, não há uma solução em H, este algoritmo não converge a uma solução aproximativa (quer dizer, que minimiza as classificações errôneas).
- Se D é linearmente separável, quanto maior a margem entre os dois conjuntos de pontos, tanto mais fácil a sua separação, isto é, tanto mais

rápido o Perceptron converge. Mais exatamente, este algoritmo termina em $\leq R^2/\gamma^2$ passos onde

- R é o *raio* de D, e
- γ é a *margem*, a máxima distância de um hiperplano (= uma reta quando X é o plano) entre os dois conjuntos dos pontos (onde a distância da reta entre os dois conjuntos de pontos é a mínima distância entre ela e os pontos).

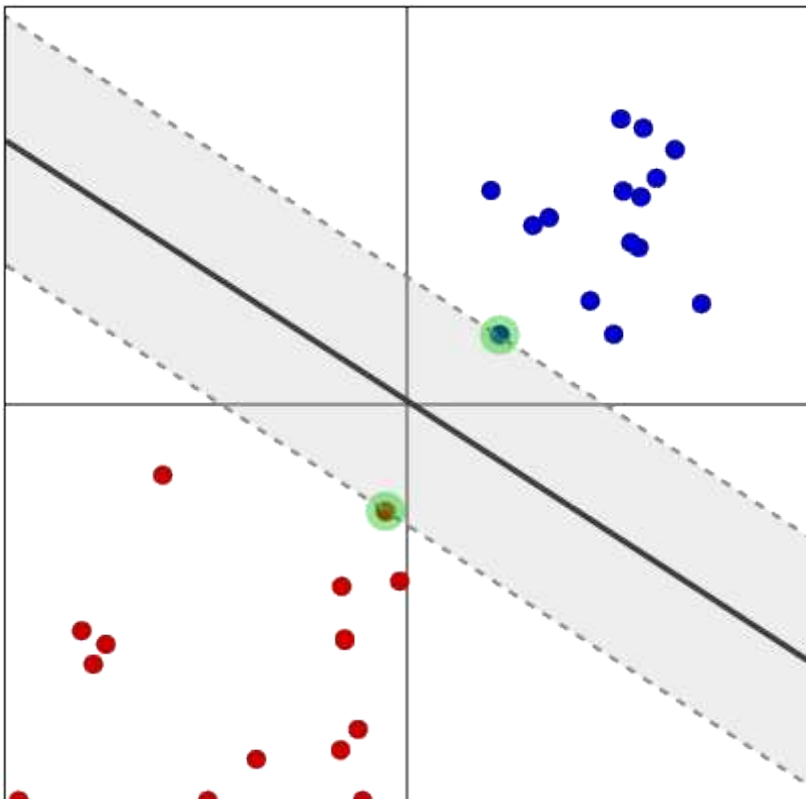


Figura 9: A margem de um hiperplano separando dois conjuntos de pontos

Em fórmulas, o *raio* e a *margem* são definidos como segue:

- o **raio** R de um conjunto de pontos D é definido por

$$R := \max\{\|x\| \text{ em } D\},$$

- para um hiperplano H (dado por $H = \{x \text{ em X ortogonais a } h\}$ onde *ortogonal* significa $x^T h = 0$) e dados D, a **margem** entre H e D é

$$\gamma := \max\{\text{dist}(d, H) : d \text{ em } D\}$$

onde, para um hiperplano $\mathrm{H} = \{x \in \mathrm{X} : h^{\mathrm{T}}x = 0\}$, a *distância* entre ele e um ponto x é definida por

$$\mathrm{dist}(x, \mathrm{H}) := |h^{\mathrm{T}}x|/\|h\|^2$$

(Recordemo-nos de que, para dois vetores x e y no plano,

$$xy = \eta\|x\|\|y\|$$

onde η é o angulo entre eles, e $\|x\|$ e $\|y\|$ são os seus comprimentos.)

Animação. Para ganharmos um intuito como o Perceptron se aproxima da reta separadora, há uma animação online (em `JavaScript`) da Universidade do Texas em Austin.

O Algoritmo `Pocket` (= bolso). Recordemo-nos de que se o problema de aprendizado é um problema de classificação binária, isto é, $\mathrm{X} = \{1\} \times \mathbb{R} \times \cdots \times \mathbb{R}$ (com $1 + d$ fatores) e $\mathrm{Y} = \{\pm 1\}$, então no algoritmo `Perceptron` as hipóteses H têm a forma

$$h \colon x \mapsto \mathrm{sign}(w^{\mathrm{T}}x)$$

para vetores w em $\mathbb{R}^{d+1}$.

Recordemo-nos de que na formalização `PAC` (= Provavelmente Aproximadamente Correto) do aprendizado, buscamos h tal

- que $\mathrm{E}_{\mathrm{X}}(h)$ seja próximo de $\mathrm{E}_{\mathrm{D}}(h)$, e
- que $\mathrm{E}_{\mathrm{D}}(h)$ seja próximo de 0.

O Algoritmo `Perceptron`. Se a amostra D é linearmente separável, isto é, há h em H tal que

$$\mathrm{E}_{\mathrm{D}}(h) = 0,$$

então o algoritmo `Perceptron` encontrará tal h. Recordemo-nos de que, para cada passo $t \geq 0$, o `Perceptron` seleciona um ponto erroneamente classificado $x(t)$ com valor $y(t) = y(x(t))$ e atualiza $w(t)$ por

$$w(t+1) := w(t) + y(t)x(t).$$

Mas, se D não é linearmente separável, isto é, não há uma solução em H, este algoritmo não converge a uma solução aproximativa (quer dizer, que minimiza $\mathrm{E}_{\mathrm{D}}(h)$, as classificações errôneas).

O Algoritmo `Pocket`. Se D não é linearmente separável, isto é, é impossível achar h tal que $\mathrm{E}_{\mathrm{D}}(h) = 0$, queremos pelo menos minimizá-lo, isto é, encontrar o (melhor) vetor de pesos w (em $\mathbb{R}^{d+1}$) que minimiza

$$\mathrm{E}_{\mathrm{D}}(h) = \#\{x_n \text{ em D tal que } \mathrm{sign}(w^{\mathrm{T}} x_n) \neq y_n\}/\#\mathrm{D}$$

O algoritmo `Pocket` (= bolso) aplica o `Perceptron` repetidamente (por exemplo, em T repetições) e guarda no seu bolso o melhor vetor de pesos w até então encontrado. Isto é,

```
Põe w := w(0)
Para t = 0, ..., T:
    Roda um passo do Perceptron para obter w(t + 1)
    Calcula E(w(t + 1))
    Se E(w(t + 1)) < E(w), então põe w = w(t + 1)
Devolve w
```

Isto é, em comparação ao `Perceptron`, o `Pocket` avalia após a atualização $w(t) \mapsto w(t+1)$ se o novo vetor é melhor (quanto o erro sobre a amostra) do que o melhor até então encontrado w, e, caso sim, substitui w por $w(t+1)$. Pela computação de $\mathrm{E}_{\mathrm{D}}(w(t+1))$, que percorre todos os pontos da amostra, o algoritmo `Pocket` é mais lento do que o `Perceptron`.

Sabemos

- nem qual é o melhor vetor de pesos,
- nem quando se encontrará.

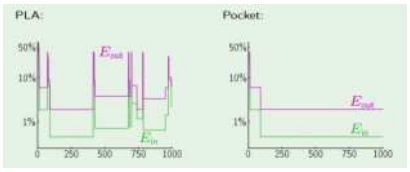


Figura 10: A evolução do erro do `Perceptron` contro o do `Pocket`

Recapitulação. Questões para quem quiser testar a assimilação do conteúdo desta seção sobre o Perceptron:

1. Qual é a medida de erro que o algoritmo Perceptron minimiza?
2. Entre quais funções o Perceptron escolhe a melhor, isto é, que minimiza o erro?
3. Quais parâmetros o Perceptron otimiza, isto é, quais números altera até ter minimizado o erro?
4. O Perceptron consegue separar (por uma reta) qualquer rotulagem (= escolha de cruzes e bolas) de quatro pontos no plano?
5. O número de erros sempre diminui após uma iteração adicional?

4.2 A Regressão Linear

O banco não somente quer decidir se o cliente recebe um crédito ou não, mas mais exatamente estimar o seu valor.

Tem-se

- $\mathrm{X} = \mathbb{R} \times \cdots \times \mathbb{R}$ (com d fatores) = os possíveis dados dos clientes (tais como bens, dívidas, salário, idade),
- $\mathrm{Y} = \mathbb{R}$ = os possíveis valores dos créditos concedidos

e

- $\mathrm{D} = \{(x_1, y_1), \ldots, (x_\mathrm{N}, y_\mathrm{N})\}$ = os dados históricos de clientes e os valores dos créditos que lhes foram concedidos.

Hipóteses. Geometricamente, para $\mathrm{X} = \mathbb{R}$, a regressão linear busca a reta h que minimize a distância (quadrática) aos pontos da amostra D.

Cada hipótese $h\colon \mathrm{X} \to \mathrm{Y}$ corresponde a um vetor $w = (w_1, \ldots, w_d)$ em $\mathbb{R}\times\cdots\times\mathbb{R}$; ela é dada por

$$h\colon x \mapsto xw^\mathrm{T} := x_1 w_1 + \cdots + x_d w_d.$$

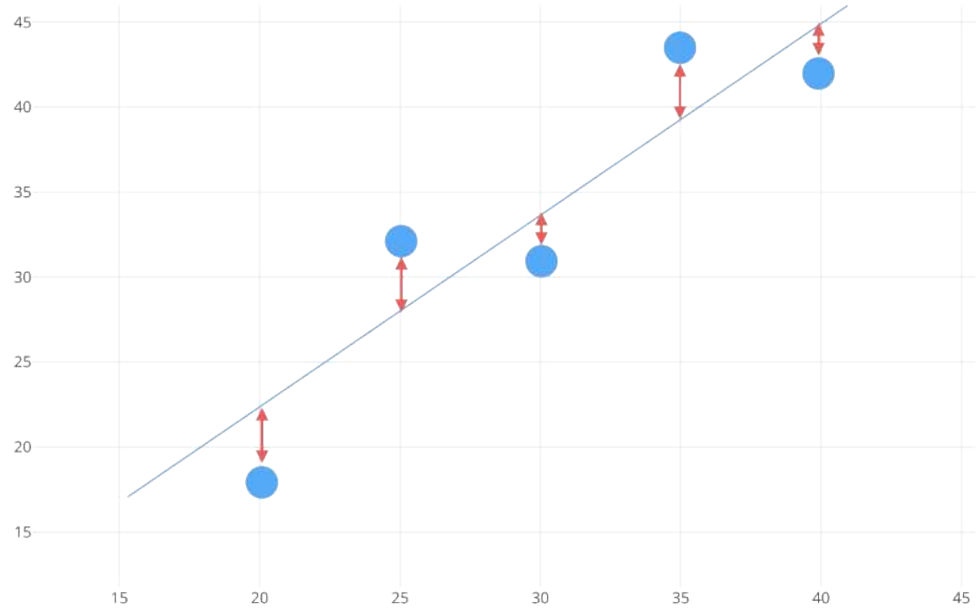


Figura 11: As distâncias entre a reta h e os valores $y_1, \ldots, y_N$ dos pontos da amostra

Erro. O erro que queremos minimizar é a distância quadrática entre h e y, isto é, o *valor médio* da diferença $h - y$ ao quadrado sobre a amostra finita $D = \{x_1, \ldots, x_N\}$, onde a integral do valor médio se torna a soma finita

$$E_D(h) = \int_D (h(x) - y(x))^2 \, dx = [(h(x_1) - y_1)^2 + \cdots + (h(x_N) - y_N)^2]/N$$

Cada hipótese h corresponde ao vetor dos seus pesos w. Por isso, denotamos o erro especifico daqui em diante por $E_D(w)$ (em vez de $E_D(h)$), evidenciando como função em $\mathbb{R} \times \cdots \times \mathbb{R}$.

Se denotamos

- por X a matriz em $\mathbb{R}^d \times \cdots \times \mathbb{R}^d$ com as linhas $x_1, \ldots, x_N$ em D, e
- por y o vetor coluna em $\mathbb{R}^N$ com as entradas $y_1, \ldots, y_N$,

então

$$E_D(w) = [(x_1 w^T - y_1)^2 + \cdots + (x_N w^T - y_N)^2]/N = \|Xw^T - y\|^2/N \qquad (1)$$

onde

- w^{T} denote o vetor *coluna* cujas entradas são as do vetor *linha* w, e
- $\|\cdot\|$ denote a *norma euclidiana*, dada por $\|v\| = (v_1^2 + \cdots + v_d^2)^{1/2}$ para um vetor real $v = (v_1, ..., v_d)$, isto é, $\|v\|^2 = v^{\mathrm{T}} \cdot v$;

e onde

- a primeira igualdade vale por definição de $\mathrm{E_D}(w)$, e
- a segunda igualdade vale porque os coeficientes do vetor $\mathrm{X}w - y$ são $w^{\mathrm{T}}x_1 - y_1, \ldots, w^{\mathrm{T}}x_{\mathrm{N}} - y_{\mathrm{N}}$.

Algoritmo. A **regressão linear** calcula o vetor w em $\mathbb{R}\times\cdots\times\mathbb{R}$ que minimiza $\mathrm{E_D}(w)$.

Se $\mathrm{X} = \{1\} \times \mathbb{R}$, podemos visualizar

- a amostra $(1, x_1), \ldots, (1, x_{\mathrm{N}})$ e os seus valores $y_1, \ldots, y_{\mathrm{N}}$ sob a função divina y pelos pontos $(x_1, y_1), \ldots, (x_{\mathrm{N}}, y_{\mathrm{N}})$ no plano, e
- a hipótese $h \leftrightarrow w = (w_0, w_1)$ pela reta no plano que
  - se origina na altura w_0, e
  - tem inclinação w_1.

  (Isto é, é a função $h\colon x \mapsto w_0 + w_1 x$ cujo gráfico é a reta dada por todos os pontos $(x, w_0 + w_1 x)$ onde x percorre $\mathbb{R}$.)

Com efeito, na Regressão Linear trata-se mais de uma fórmula do que de um algoritmo. Daremos duas vias para demonstrá-la, obtendo o vetor de pesos w que minimiza esta distância

1. como projeção de outro vetor a um espaço menor, e
2. como zero da derivada do erro como função em w.

Antes de começar, recordemo-nos de que, para uma matriz A, a sua matriz *transposta* A^{T} é a matriz cujas colunas são as linhas de A, ou, equivalentemente, é a matriz obtida por espelhamento de A ao longo da diagonal. Em particular, para $v =$ A uma matriz com uma linha ou coluna, um vetor linha respetivamente vetor coluna, v^{T} é o vetor coluna respetivamente linha cujas entradas são as de v.

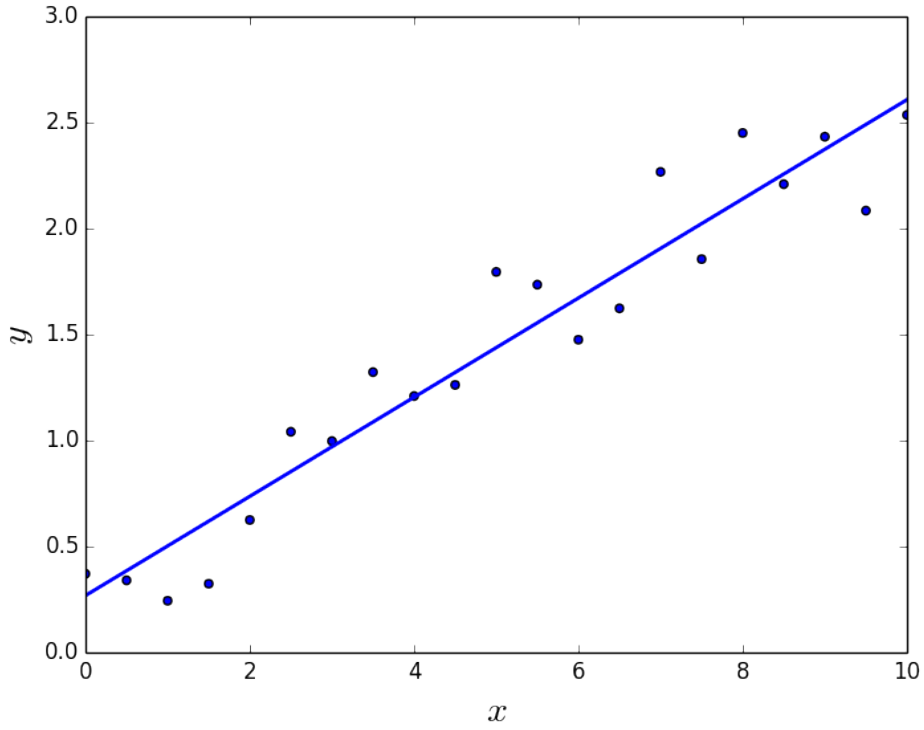


Figura 12: A reta dada pela Regressão Linear para $\mathrm{X} = \{1\} \times \mathbb{R}$

Aproximação por Projeção. Para exemplificar a derivação da fórmula da regressão linear, restrinjamos primeiro a $\mathrm{X} = \mathbb{R}$. Na Regressão Linear, queremos encontrar uma reta que minimize a distância a pontos (x_1, y_1), ..., (x_N, y_N) no plano. Vimos que a distância entre

- a reta (que origina em 0) com inclinação w e
- os pontos

é medida pela distância

$$\|\mathrm{X}w - y\|$$

entre o vetor $\mathrm{X}w$ e o vetor y, onde

- pomos $\mathrm{X} = (x_1, ..., x_N)$, e
- pomos $y = (y_1, ..., y_N)$.

Então, qual é o w que minimize esta distância? Os possíveis valores $\{Xw$ para w em $\mathbb{R}\}$ formam uma reta (em $\mathbb{R}^N$, o plano para $N = 2$), e o ponto Xw_0 da reta que é o mais próximo de y é a projeção Py a esta reta; em fórmulas

$$Xw_0 = Py.$$

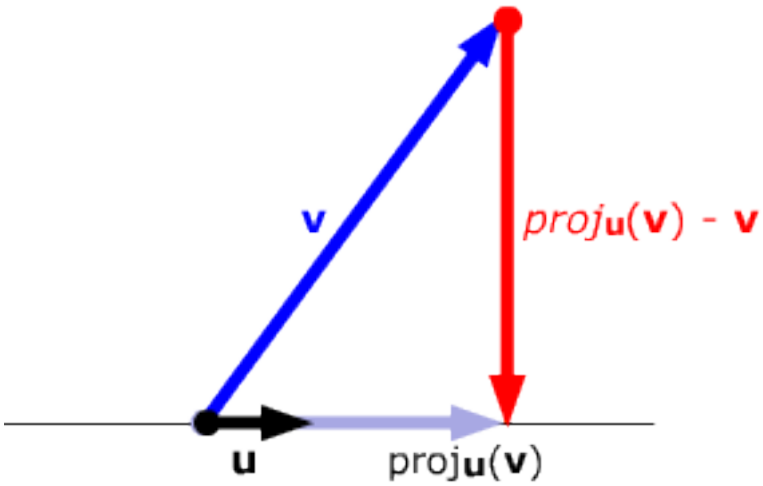


Figura 13: Projeção de um vetor a uma reta

O ponto Xw_0 da reta é a projeção de y se, e tão-somente se, a sua diferença $y - Xw_0$ é *ortogonal* (ou *perpendicular*, isto é, forma um ângulo reto) à reta; em fórmulas, se o seu produto escalar é zero,

$$X^T(y - Xw_0) = 0$$

ou, equivalentemente, se

$$X^TXw_0 = X^Ty.$$

Obtemos

$$w_0 = X^+y$$

para o vetor $X^+ = (X^TX)^{-1}X^T$ escalado e transposto de X.

Em geral, para X de dimensão $d < N$ maior, obtemos

- que $\mathrm{X} = (x_1, ..., x_\mathrm{N})$ é uma *matriz* cujas N linhas são os vetores $x_1, \ldots, x_\mathrm{N}$ com d entradas, e
- que w é um *vetor* (de pesos) com d entradas.

enquanto $y = (y_1, ..., y_\mathrm{N})$ permanece um vetor de coluna com N entradas (porque a saída $\mathrm{Y} = \mathbb{R}$ não mudou).

Os possíveis valores $\{\mathrm{X}w$ para w em $\mathbb{R}^d\}$ formam um *espaço* (de dimensão d em $\mathbb{R}^\mathrm{N}$; por exemplo, um plano para $d = 2$). O ponto $\mathrm{X}w_0$ do espaço que é o mais próximo de y é a projeção $\mathrm{P}y$ de y a este espaço; Obtemos, quando a *matriz* quadrática $\mathrm{X}^\mathrm{T}\mathrm{X}$ é invertível (o que vale quase sempre porque $\mathrm{N} > d$, isto é, tem mais exemplos que parâmetros a especificar), da mesma forma

$$w_0 = \mathrm{X}^+ y$$

para o *pseudo-inverso* $\mathrm{X}^+ = (\mathrm{X}^\mathrm{T}\mathrm{X})^{-1}\mathrm{X}^\mathrm{T}$ de X.

Aproximação por Derivação. Recordemo-nos de Equação (1),

$$\mathrm{E}_\mathrm{D}(w) = \|\mathrm{X}w^\mathrm{T} - y\|^2/\mathrm{N}.$$

Obtemos

$$\begin{aligned}\|\mathrm{X}w^\mathrm{T} - y\|^2/\mathrm{N} &= (\mathrm{X}w^\mathrm{T} - y)^\mathrm{T}(\mathrm{X}w^\mathrm{T} - y)/\mathrm{N} \\ &= [w\mathrm{X}^\mathrm{T}\mathrm{X}w^\mathrm{T} - 2w\mathrm{X}^\mathrm{T}y + y^\mathrm{T}y]/\mathrm{N},\end{aligned}$$

pela aplicação das igualdades

$$(\mathrm{AB})^\mathrm{T} = \mathrm{B}^\mathrm{T}\mathrm{A}^\mathrm{T} \quad \text{e} \quad (\mathrm{A}^\mathrm{T})^\mathrm{T} = \mathrm{A}$$

para toda matriz A e B, em particular, para A ou B um vetor linha ou coluna v (= matriz com uma única linha ou coluna); além, para dois vetores v e w, da simetria do seu produto escalar $v^\mathrm{T}w = w^\mathrm{T}v$.

Como a função $\mathrm{E}_\mathrm{D}(w)$ em w é (por Equação (1)) diferençável, podemos encontrar o seu mínimo no ponto w_0 em que o seu derivado iguala 0. Isto é,

$$\nabla\mathrm{E}_\mathrm{D}(w_0) = 0$$

onde ∇ é o *gradiente* de E_D, o vetor linha

$$\nabla\mathrm{E}_\mathrm{D}(w) = (\partial\mathrm{E}_\mathrm{D}(w)/\partial w_1, \partial\mathrm{E}_\mathrm{D}(w)/\partial w_2, ..., \partial\mathrm{E}_\mathrm{D}(w)/\partial w_d)$$

cujas entradas são as derivadas parciais.

Para derivar

$$\mathrm{E_D}(w)_\mathrm{D} = [w\mathrm{X}^\mathrm{T}\mathrm{X}w^\mathrm{T} - 2w\mathrm{X}^\mathrm{T}y + y^\mathrm{T}y]/\mathrm{N},$$

observamos primeiro

- que $\nabla_w(y^\mathrm{T}y) = 0$, porque é constante, não depende de w, e
- que $\nabla_w 2w\mathrm{X}^\mathrm{T}y = 2\mathrm{X}^\mathrm{T}y$, porque é linear em w.

Para calcular $\nabla_w w\mathrm{X}^\mathrm{T}\mathrm{X}w^\mathrm{T}$, abreviemos $\mathrm{A} = \mathrm{X}^\mathrm{T}\mathrm{X}$, e observemos que, para a função $f : w \mapsto w\mathrm{A}w^\mathrm{T}$,

$$\begin{aligned} f(\mathbf{w}+\mathbf{h}) - f(\mathbf{w}) &= (\mathbf{w}+\mathbf{h})\mathrm{A}(\mathbf{w}+\mathbf{h})^\mathrm{T} - \mathbf{w}\mathrm{A}\mathbf{w}^\mathrm{T} \\ &= \mathbf{w}\mathrm{A}\mathbf{h}^\mathrm{T} + \mathbf{h}\mathrm{A}\mathbf{w}^\mathrm{T} + \mathbf{h}\mathrm{A}\mathbf{h}^\mathrm{T} \\ &= \nabla_w f(w) + \mathrm{O}(\|h\|) \end{aligned}$$

com

$$\nabla_w f\, h = \mathbf{w}\mathrm{A}\mathbf{h}^\mathrm{T} + \mathbf{h}\mathrm{A}\mathbf{w}^\mathrm{T} = \mathbf{w}\mathrm{A}\mathbf{h}^\mathrm{T} + \mathbf{w}\mathrm{A}^\mathrm{T}\mathbf{h}^\mathrm{T}$$

onde usamos a simetria $v^\mathrm{T} \cdot w = w^\mathrm{T}v$ para $v = h$ e $w = \mathrm{A}x$.

Se aplicamos estas equações a $\mathrm{E_D} = [w^\mathrm{T}\mathrm{X}^\mathrm{T}\mathrm{X}w - 2w^\mathrm{T}\mathrm{X}^\mathrm{T}y + y^\mathrm{T}y]/\mathrm{N}$, então obtemos o vetor linha

$$\nabla \mathrm{E_n}(w) = (2/\mathrm{N}) \cdot (\mathrm{X}^\mathrm{T}\mathrm{X} \cdot w - \mathrm{X}^\mathrm{T}y).$$

Logo, $\nabla \mathrm{E_n}(w) = 0$ se, e tão-somente se, $\mathrm{X}^\mathrm{T}\mathrm{X} \cdot w = \mathrm{X}^\mathrm{T}y$ se, e tão-somente se,

$$w = \mathrm{X}^+y \quad \text{com } \mathrm{X}^+ = (\mathrm{X}^\mathrm{T}\mathrm{X})^{-1}\mathrm{X}^\mathrm{T}.$$

Computação. Concluímos que o valor mínimo do erro específico

$$\mathrm{E_D}(w)$$

é atingido se, e tão-somente se,

$$\mathrm{X}^\mathrm{T}\mathrm{X}w = \mathrm{X}^\mathrm{T}y,$$

isto é, se e tão-somente se, para $\mathrm{X}^\mathrm{T}\mathrm{X}$ invertível,

$$w = \mathrm{X}^\dagger y$$

onde

$$X^{\dagger} := (X^{T}X)^{-1}X^{T},$$

é o *pseudo-inverso* (ou, mais exatamente, o `Moore-Penrose inverso`) de X. Este vetor de pesos w é a única solução ótima.

Esta computação do pseudo-inverso é a parte computacionalmente mais custosa da regressão linear.

Para acelerar a computação de X^{+}, computa-se uma fatoração

$$X = QR$$

onde

- a matriz Q é *ortonormal*, isto é, $Q^{T}Q = 1$, a matriz cujas únicas entradas diferentes de 0 são as da diagonal com o valor 1, e
- a matriz R *triangular*.

por exemplo, pelo *método de Gram-Schmidt*: Denotem P_1, P_2, ... as projeções aos vetores x_1, x_2, ..., dadas por

$$v \mapsto x_1^{T}v \cdot x_1, v \mapsto x_2^{T}v \cdot x_2, ...$$

Para cada $i = 1, 2, \ldots$ o vetor

$$\mathbf{q}_i := x_i - P_1 x_i - \cdots - P_{i-1}x_i$$

é ortogonal a $\mathbf{q}_1, \ldots, \mathbf{q}_{i-1}$. Logo, os vetores

$$q_1 := \mathbf{q}_1/\|q_1\|, q_2 := \mathbf{q}_2/\|q_2\|, ...$$

formam uma base ortonormal.

Isto é,

- as colunas da matriz Q ortonormal são dadas pelos coeficientes dos vetores q_1, q_2, ..., e
- a linha número i da matriz R é dada pelos coeficientes $x_1^{T}x_i, \ldots, x_{i-1}^{T}x_i$; em particular, a matriz é triangular.

Porém, este método é frágil, isto é, pequenos erro de arredondamento em resultados intermediários podem levar a grandes erros nos resultados finais. A este fim, prefere-se o *método de Gram-Schmidt modificado* onde

$$\mathbf{q}_i \coloneqq x_1 - (1 - \mathrm{P}_1) \cdots (1 - \mathrm{P}_{i-1}) x_i.$$

Graças a fatoração $\mathrm{X} = \mathrm{QR}$ com QQ^{T}, obtemos

$$\mathrm{X}^{+} = (\mathrm{R}^{\mathrm{T}}\mathrm{R})^{-1}\mathrm{QR}^{\mathrm{T}}.$$

A computação do inverso de uma matriz triangular em mais rápida do que a de uma matriz geral pois

- os coeficientes de uma matriz A inversa se computam pelos *determinantes* de (sub-)matrizes $\mathrm{A}_{i,j}$ de A, e
- o determinante de uma matriz *triangular* é o produto dos seus coeficientes *diagonais*.

Resumo. Concluímos que os passos do algoritmo da `Regressão Linear` consistem em:

1. Dado $\mathrm{D} = \{(x_1, y_1), \ldots, (x_{\mathrm{N}}, y_{\mathrm{N}})\}$, define a matriz X e o vetor y por

$$\mathrm{X}^{\mathrm{T}} = (x_1, \ldots, x_{\mathrm{N}}) \quad \text{e} \quad y^{\mathrm{T}} = (y_1, \ldots, y_{\mathrm{N}}).$$

2. Calcula o *pseudo-inverso* $\mathrm{X}^{\dagger}$ da matriz X, isto é, para $\mathrm{X}^{\mathrm{T}}\mathrm{X}$ invertível,

$$\mathrm{X}^{\dagger} = (\mathrm{X}^{\mathrm{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathrm{T}}.$$

3. Devolve $w = \mathrm{X}^{\dagger} y$.

Animação. Para ganharmos um intuito como a Regressão Linear calcula a reta que aproxima melhor os pontos, há uma implementação online (em `JavaScript`) da Universidade do Illinois.

4.3 Regressão Logística

A Regressão Logística aplica-se quando

- a saída *da amostra* é binária, isto é, $y_1, \ldots, y_N$ em $\{\pm 1\}$, e
- a função alvo é uma medida de probabilidade, isto é, a sua imagem está no intervalo $[0,1]$ (e *não* em Y).

Por exemplo, nas questões,

- quão provável é que o cliente quite as contas, ou
- quão provável é que o paciente sofra um ataque cardíaco.

Na Regressão Logística,

- a entrada $X = \mathbb{R} \times \cdots \times \mathbb{R}$, e
- saída $Y = [0,1]$.

e

- as hipóteses $H = \{h(x) = \theta(w_1 x_1 + \cdots + w_d x_d - b) : w_1, ..., w_d, b \text{ em } \mathbb{R}\}$

que são medidas de probabilidade obtidas pela composição com certa *função logística*.

Por exemplo, $X = \mathbb{R}$, onde x mede a pressão sanguínea (ou arterial) do paciente, e y indica se sofreu um ataque cardíaco ou não.

Enquanto o `Perceptron` decide para cada ponto se é preto ou branco (isto é, se é ao lado direito ou esquerdo da reta), a Regressão Logística agrisalha os pontos: quanto mais preto o ponto, tanto mais provável o evento em questão. Por exemplo, se cada ponto representa um cliente, que amortize as dívidas.

Com efeito, a função divina y que queremos conhecer é uma *variável aleatória condicional* (binária).

Moralmente, y,

- em vez de ser uma função que associa a cada argumento x um único valor $y = \pm 1$,
- é uma *variável aleatória condicional* (binária) que associa a cada argumento x uma *probabilidade* $P(y = \pm 1|x)$ de cada valor $y = \pm 1$.

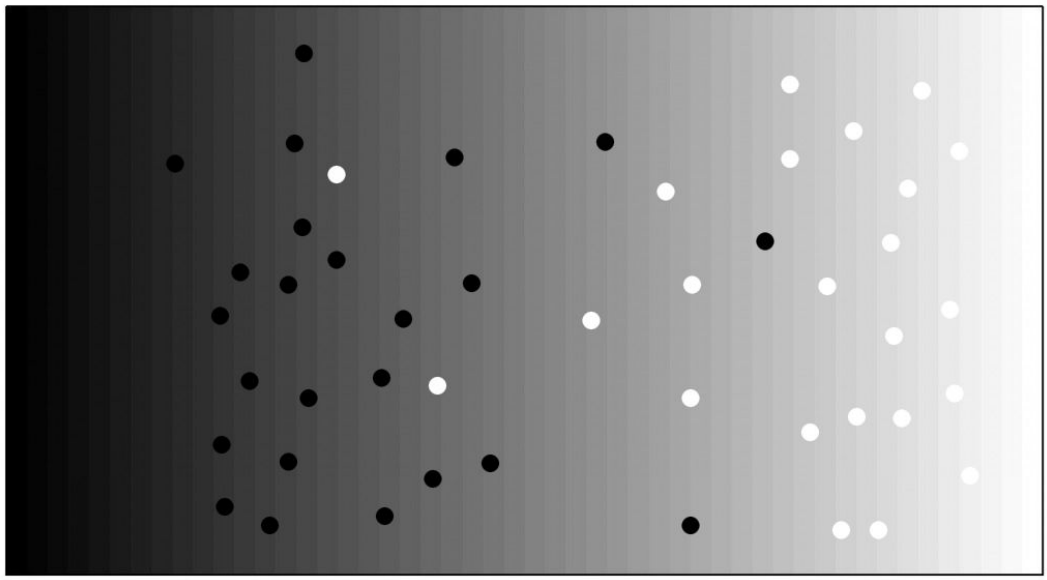


Figura 14: O grau de griso corresponde à probabilidade e os pontos aos exemplos observados

Como $Y = \{\pm 1\}$ é binário,

$$P(y = -1|x) = 1 - P(y = 1|x), \quad (*)$$

basta conhecer $P(y = 1|x)$. Isto é, $P(y = 1|x)$ determina y; em outras palavras, o conhecimento de $P(y = 1|x)$ corresponde ao conhecimento da variável aleatória condicional y.

As nossas hipóteses serão aproximações a $P(y = 1|x)$; logo, por $(*)$, à variável aleatória condicional y.

Função Logística. Para a regressão logística, em que as nossas hipóteses são funções cujos valores são probabilidades, precisamos de uma função (duas vezes) diferençável e monótona $\theta\colon \mathbb{R} \to [0,1]$.

A necessidade da monotonia da função θ baseia-se na concepção que, dado um ponto x, quanto maior o seu valor $w^{\mathrm{T}} \cdot x$, tanto mais provável que a decisão seja positiva. (Por exemplo, quanto maior a pontuação do devedor, mais provável que consiga pagar os créditos.)

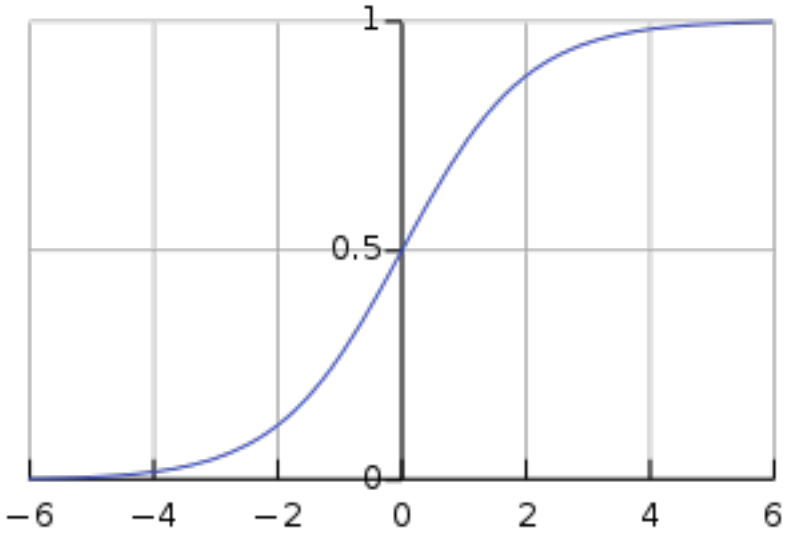


Figura 15: A função logística

A nossa escolha (e a da maioria) é a **função logística** (ou função **sigmoid**) definida por

$$\theta(s) := e^s/(1+e^s) = 1/(1+e^{-s}).$$

(Outra escolha popular para θ é a função tangente hiperbólica $\theta = \tanh\colon \mathbb{R} \to [-1,1]$ definida por $\tanh := (e^s - e^{-s})/(e^s + e^{-s})$.) Pelo gráfico da função logística, vemos que as duas extremidades se aproximam dos valores 0 e 1 rápido; também, satisfaz a simetria $\theta(-s) = 1 - \theta(s)$ que nos será útil a seguir.

`Log-Odds` (= chances logarítmicas).

Relação à Distribuição Binomial. A medida de probabilidade, ou distribuição, mais simples é a de uma variável aleatória binária, isto é, que tem dois possíveis valores 0 e 1. Chama-se de Bernoulli, e para uma probabilidade fixa π em $[0,1]$ é definida por $p(x;\pi) = P(X = x)$ onde

$$P(X = 1) = \pi \quad e \quad P(X = 0) = 1 - \pi$$

Expressemo-la pela função exponencial:

$$\begin{aligned} p(x;\pi) &= \pi^x (1-\pi)^{1-x} \\ &= \exp\left(x \log(\pi) + (1-x)\log(1-\pi)\right) \\ &= \exp\left(x \log\left(\frac{\pi}{1-\pi}\right) + \log(1-\pi)\right) \end{aligned}$$

Isto é,

$$p(x;\eta) = \exp(x\eta - c(\eta))$$

é uma distribuição definida pela exponencial (isto é, faz parte da família exponencial das distribuições) para um parâmetro η e uma constante $c(\eta)$ onde $\eta := \log(\frac{\pi}{1-\pi})$ é a função `logit` (ou `log-odds`) de π. A função logística (ou `expit`) é a função inversa da função `logit`; isto é,

$$\pi = \frac{1}{1+\exp(-\eta)}.$$

Usos.

Dificuldade da Computação da Distribuição Normal.

Substituição.

Comparação à Distribuição Normal. Regressão logística (modelo Logbit) em contraste ao modelo probit que usa a função da densidade da distribuição normal (e regressão de Poisson que usa a distribuição de Poisson).

As funções de densidade da distribuição normal e logística são próximas, uma da outra, para uma escolha de parâmetros apropriados a este fim:

Parâmetros. Como de costume, não explicitamos o parâmetro constante μ, o valor esperado, na hipótese, mas usamos uma saída $X = \{1\} \times \mathbb{R} \times \cdots \times \mathbb{R}$ em que todos argumento tem a primeira entrada constante igual a 1. Isto é, $w_0 = \mu$.

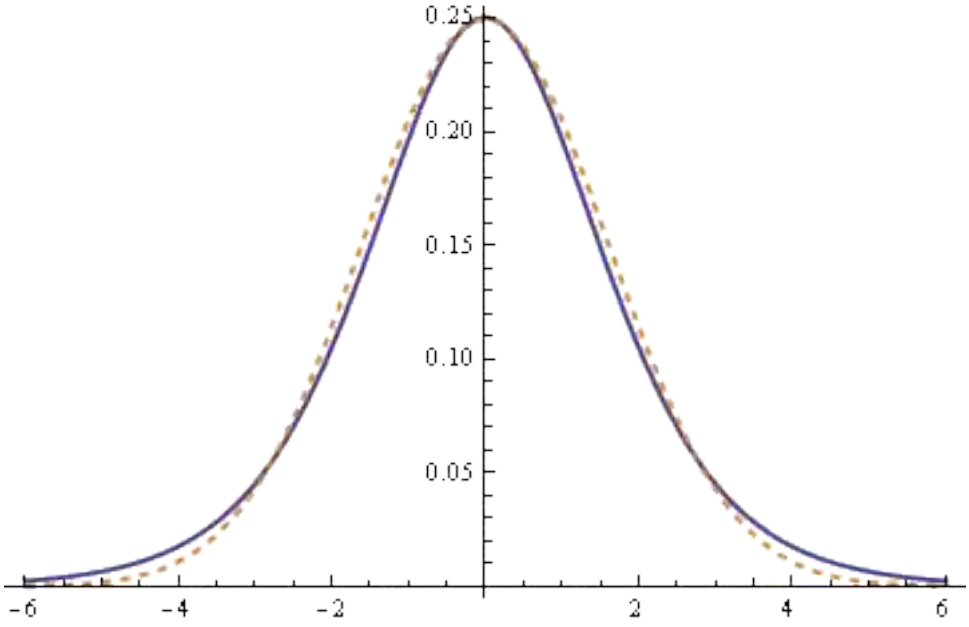


Figura 16: Gráficos de densidade da distribuição normal e logística

Hipóteses. As nossas hipóteses são

$$\mathrm{H} := \{h = \theta(w^{\mathrm{T}}\cdot) \text{ para } w = (w_1, \ldots, w_d) \text{ em } \mathbb{R} \times \cdots \times \mathbb{R}\}.$$

O valor $h(x)$ no intervalo $[0,1]$ corresponde à probabilidade condicional $\mathrm{P}_h(y|x)$ definida por:

$$h(x) \text{ para } y = 1 \quad \text{e} \quad 1 - h(x) \text{ para } y = -1$$

(Por exemplo, dadas as características x do cliente, a probabilidade que

- consiga pagar os créditos (o caso $y = 1$), ou
- falha pagar os créditos (o caso $y = -1$).)

A nossa função alvo é

$$p(x) = \mathrm{P}(y = 1|x),$$

a probabilidade (condicional) que $y = 1$ dado x (por exemplo, dadas as características x de um paciente, a probabilidade que ele sofra um ataque cardíaco, $y = 1$).

Observamos mais exatamente que p (e $p_h := \mathrm{P}_h(y = 1|\cdot)$) é uma **função densidade de probabilidade** condicional (FDP), ou *densidade de uma variável aleatória contínua* condicional, que mede a probabilidade

- de uma variável aleatória Y tomar um valor *fixado* y_0,
- em dependência de um valor variável x de uma variável aleatória X.

Erro. A amostra $\mathrm{D} = \{x_1, \ldots, x_\mathrm{N}\}$ não revela diretamente $p(x) = \mathrm{P}(y = 1|x)$, a probabilidade que $y = 1$ dado x, mas a revela indiretamente: os seus valores $y_1, \ldots, y_\mathrm{N}$ em $\{\pm 1\}$ são distribuídos com probabilidade $p(y_1), \ldots, p(y_\mathrm{N})$.

Para estimar o erro que uma hipótese h faz em aproximar y sobre D, usamos o estimador de **máxima verossimilhança**: *Maximizar* a verossimilhança quer dizer determinar os parâmetros h da distribuição P_h que *maximizam a probabilidade* $\mathrm{P}_h(y_1, \ldots, y_\mathrm{N}|x_1, \ldots, x_\mathrm{N})$.

Como supomos $(x_1, y_1), \ldots, (x_\mathrm{N}, y_\mathrm{N})$ independente e identicamente distribuídos,

$$\mathrm{P}_h(y_1, \ldots, y_\mathrm{N}|x_1, \ldots, x_\mathrm{N}) = \mathrm{P}_h(y_1|x_1) \cdots \mathrm{P}(y_\mathrm{N}|x_\mathrm{N}).$$

Recordemo-nos de que h corresponde a probabilidade $\mathrm{P}_h(y|x)$ dada por:

$$h(x) \text{ para } y = 1 \quad \text{e} \quad 1 - h(x) \text{ para } y = -1$$

Como $h(x) = \theta(w^\mathrm{T} x)$ e $\theta(-s) = 1 - \theta(s)$, esta igualdade pode ser simplificada a

$$\mathrm{P}_h(y|x) = \theta(y w^\mathrm{T} x). \tag{2}$$

Como os valores de $\exp(-x) = 1/\exp(x)$ (e logo da função θ) decrescem a zero rápido, é computacionalmente preferível maximizar esta probabilidade pelo seu logaritmo. Como a função $-\mathrm{N}^{-1} \log$ é monótona decrescente (pelo fator de normalização negativo $-\mathrm{N}^{-1}$), esta probabilidade é máxima se e tão-somente se

$$\mathrm{E}_\mathrm{D}(w) = -\mathrm{N}^{-1} \log \mathrm{P}_h(y_1, \ldots, y_\mathrm{N}|x_1, \ldots, x_\mathrm{N})$$

é mínima. Explicitamente, se e tão-somente se

$$\begin{aligned} \mathrm{E}_\mathrm{D}(w) &= -\mathrm{N}^{-1} \log(\mathrm{P}_h(y_1|x_1) \cdots \mathrm{P}_h(y_\mathrm{N}|x_\mathrm{N})) \\ &= \mathrm{N}^{-1} \log(1/\mathrm{P}_h(y_1|x_1)) + \cdots + \log(1/\mathrm{P}_h(y_\mathrm{N}|x_\mathrm{N})) \end{aligned}$$

é mínima. Substituindo Equação (2), obtemos:

$$\mathrm{N}^{-1} \sum_{n=1,\ldots,\mathrm{N}} \log(1/\theta(y_n w^{\mathrm{T}} x_n));$$

substituindo $\theta^{-1}(s) = (1 + e^{-s})/e^{-s} = e^{-s} + 1$, concluímos:

$$\mathrm{E_D}(w) = \mathrm{N}^{-1} \sum_{n=1,\ldots,\mathrm{N}} \log(1 + \mathrm{e}^{-y_n w^{\mathrm{T}} x_n}).$$

Observamos que $\mathrm{E_D}$ é pequeno se sinal$(w^{\mathrm{T}} x_n) = y_n$, isto é, quanto melhor w classifica $x_1, \ldots, x_{\mathrm{N}}$, tanto menor este erro.

Algoritmo. Veremos como maximizar a *verossimilhança*, isto é, encontrar a medida que maximiza a probabilidade da nossa amostra. Equivalentemente, veremos como minimizar $\mathrm{E_D}$ em dependência de w. Para encontrar um mínimo, usa-se o gradiente de uma função que aponta na sua direção. Com efeito, para estar num mínimo, tem de ser zero.

Gradiente. Geometricamente, por exemplo, para uma função $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ de duas variáveis, no ponto (x_0, y_0),

- a sua primeira derivada parcial $\frac{\partial f}{\partial x}(x_0,y_0)$ é a inclinação do gráfico no ponto (x_0,y_0) na direção do eixo-x, e
- a sua segunda derivada parcial $\frac{\partial f}{\partial y}(x_0,y_0)$ é a inclinação do gráfico no ponto (x_0,y_0) na direção do eixo-y.

O *gradiente* ∇f de uma função $f(x_1, \ldots, x_d)$ no ponto x_0 é definido pelo vetor linha (e *não* coluna)

$$\nabla f(x_0) = (\frac{\partial f}{\partial x_1}(x_0), \ldots, \frac{\partial f}{\partial x_d}(x_0)).$$

Na nossa situação, onde

$$f(w) = \mathrm{E_D}(w) = \mathrm{N}^{-1} \sum_{n=1,\ldots,\mathrm{N}} \log(1 + \mathrm{e}^{-y_n w^{\mathrm{T}} x_n}),$$

para minimizar $\mathrm{E_D}(w)$,

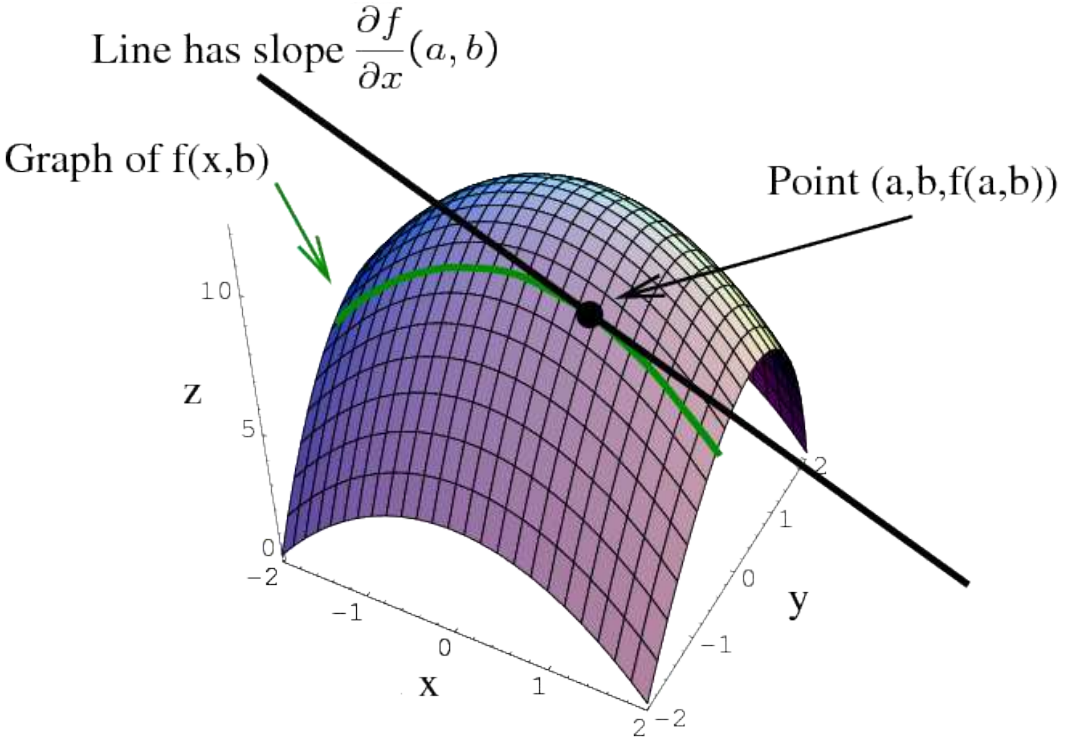


Figura 17: A derivada parcial como tangente na direção do eixo-x

1. calculamos o seu gradiente em w, o vetor linha cujas entradas são as derivadas parciais,

$$\begin{aligned}\nabla \mathrm{E_D}(w) &= (\partial \mathrm{E_D}/\partial w_1(w), \ldots, \partial \mathrm{E_D}/\partial w_d(w)) \\ &= -\mathrm{N}^{-1}\Big(\sum_{n=1,\ldots,\mathrm{N}} y_n x_n/(1+\mathrm{e}^{-y_n w^{\mathrm{T}} x_n})\Big)\end{aligned}$$

   usadano

   - a Regra da Cadeia aplicada a função e^x que iguala à sua própria derivada, e
   - a igualdade $(1+\mathrm{e}^s)/\mathrm{e}^s = 1/(1+\mathrm{e}^{-s})$;

   note-se que cada parcela $x = x_n$ é um vetor linha $x = (x_1, \ldots, x_d)$ em d entradas, e

2. buscamos w tal que $\nabla \mathrm{E_D}(w) = 0$.

Esta equação não tem uma solução *analítica*, isto é, não pode ser expressa por uma fórmula; do lado positivo, a função $\mathrm{E_D}(w)$ é convexa, por isso tem um único mínimo. Logo, nenhum dos métodos corre risco de convergir a um mínimo local (no lugar do mínimo global).

Para minimizar $\mathrm{E_D}(w)$, usamos

1. ou o método de Newton aplicado à derivada $\nabla_w \mathrm{E_D}(w)$,
2. ou o método do máximo declive aplicado à função $\mathrm{E_D}(w)$.

O método de Newton converge mais rapidamente do que o método do máximo declive; porém, a sua computação é mais custoso. Em geral, se a dimensão d não é excecionalmente grande, continua a ser muito mais rápido.

Método de Newton.

Estudamos o método de Newton

1. para funções de uma variável com valores reais, e
2. para funções de múltiplos argumentos com valores vetoriais (tal que o número das variáveis da entrada iguale ao da saída).

O uso do método de Newton para minimizar o erro na regressão logística (= o logaritmo da máxima verossimilhança) chama-se `Fisher scoring`.

Para funções reais.

Dado uma função derivável $f\colon \mathbb{R} \to \mathbb{R}$, o *método de Newton* aproxima iterativamente um zero de f, isto é, define x_1, x_2, ... em $\mathbb{R}$ com $x_n \to x$ tal que $f(x) = 0$. Geometricamente:

Pegamos um ponto (apropriado) x_1,

1. olhamos o seu valor $f(x_1)$ sob f, e
2. a tangente em $f(x_1)$ ao gráfico de f.

Esta tangente intersecta o eixo X em um ponto x_2.

1. Olhamos o seu valor $f(x_2)$ sob f, e
2. a tangente em $f(x_2)$ ao gráfico de f;

e assim por diante.

Moralmente, dado um ponto x_0, a tangente t_{x_0} é a reta que aproxima f em x_0 o melhor possível. Por isso, o zero x_1 de t_{x_0}, que é facilmente computável graças a sua forma simples, aproxima o zero x de f. Em x_1, de novo, a tangente t_{x_1} é a reta que aproxima f em x_1 o melhor; de fato, como x_1 é mais próximo do zero x de f, a nova tangente t_{x_1} aproxima f melhor do que t_{x_0} em volta de x; em particular, o x_2 de t_{x_1} é mais próximo do zero x de f do que o zero x_1 de t_{x_0}.

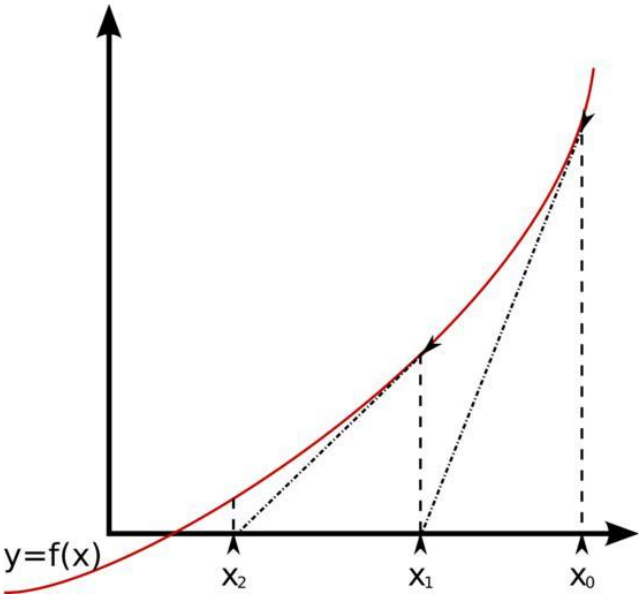


Figura 18: O método de Newton

Como função no argumento h, na iteração n, a tangente t_n em $f(x_n)$ é definida por

$$t_n(x_n + h) := f(x_n) + f'(x_n)h,$$

logo, pondo $h = x - x_n$, obtemos

$$t_n(x) := f(x_n) + f'(x_n)(x - x_n).$$

Como $0 = t(x_{n+1})$, obtemos

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Em outras palavras, o método de Newton é a iterada aplicação do *operador de Newton* $\mathrm{N} = \mathrm{N}_f \colon \mathbb{R} \to \mathbb{R}$ definido por

$$x \mapsto x - \frac{f(x)}{f'(x)}.$$

De fato, se $x_n := \mathrm{N}^n(x_0) \to x$, então $x = \mathrm{N}_f(x) = x - \frac{f(x)}{f'(x)}$, logo $f(x) = 0$.

A única condição é o encontro de um ponto de partida x_0 tal que a sequência $x_1, x_2, \ldots$ converge. Esta não é garantida, mas vale a convergência local quadrática:

Se f é duas vezes diferençável, então pela Fórmula de Taylor

$$f(x+h) = f(x) + f'(x)h + \mathrm{R}''f(x+h,h)h^2$$

com $\mathrm{R}''f(x_h,h) \to f''(x)/2$ para $h \to 0$. Para $x + h = x_{n+1}$ e $x = x_n$, e porque $h = \frac{f(x_n)}{f'(x_n)}$, obtemos

$$\begin{aligned} f(x_{n+1}) &= f(x_n) + f'(x_n)(x_{n+1} - x_n) + \mathrm{R}''f(x_{n+1}, x_n)\left(\frac{f(x_n)}{f'(x_n)}\right)^2 \\ &= \mathrm{R}''f(x_{n+1}, x_n)\left(\frac{f(x_n)}{f'(x_n)}\right)^2. \end{aligned}$$

Em particular,

$$|f(x_{n+1})| \leq \frac{|\mathrm{R}''f(x_{n+1}, x_n)|}{|f'(x_n)|^2} \cdot |f(x_n)|^2.$$

Pondo

- $\mathrm{M}'' = \sup\{|\mathrm{R}''f(x+h,x)|\}$ $(= \frac{1}{2}\sup\{|f''(x)|\}$, porque $\mathrm{R}''f(x+h,x) = \frac{1}{2}f''(\xi)$ para um ξ entre $x+h$ e x pelo Teorema do Valor Intermediário),
- $m' = \inf|f'(x)| > 0$,

e $\mathrm{C} = \mathrm{M}''/m'^2$, obtemos

$$|f(x_{n+1})| \leq \mathrm{C} \cdot |f(x_n)|^2.$$

Pondo $d_n = \mathrm{C} \cdot |f(x_n)|$, iterativamente

$$d_{n+1} \leq d_n^2 \leq d_{n-1}^{2\cdot 2} \leq d_{n-1}^{2\cdot 2\cdot 2} \leq \ldots \leq d_0^{2^n},$$

isto é,

$$\mathrm{C} \cdot |f(x_n)| \leq (\mathrm{C} \cdot |f(x_0)|)^{2^n}.$$

Concluímos que se $|f(x_0)| \leq 1/\mathrm{C} = m'^2/\mathrm{M}''$, então $f(x_n) \to 0$ quadraticamente. Por exemplo, se $|f(x_0)| \leq 10^{-1} \cdot m'/\mathrm{M}''$, então o número de zeros de $f(x_0)$, $f(x_1)$, $f(x_2)$, ... duplica a cada iteração.

Vemos aqui a convergência do método aplicado três vezes à função $f(x) = x^2 - 2$ com o seu zero $x_0 = \sqrt{2}$:

Iteração	Zero aproximativo	Erro
0	1	−0,4142135
1	1,5	0,0857864
2	1,41666667	0,0024531
3	1,41421569	0,0000021

Para funções de múltiplos argumentos. Suponhamos que $f : \mathrm{X} \to \mathrm{Y}$ é uma função de várias variáveis independentes $x_1, ..., x_d$ tal que o número d das variáveis dependentes da entrada iguale ao da saída, isto é,

$$\mathrm{X} = \mathbb{R}^d \quad \text{e} \quad \mathrm{Y} = \mathbb{R}^d.$$

Isto é, $f = (f_1, ..., f_d)$ onde cada função com valores reais $f_1, \ldots, f_d$ depende de d variáveis $x_1, \ldots, x_d$. Se f é desta forma, então o seu gradiente $f' = \nabla f = \mathrm{grad} f$ é uma matriz quadrática

$$\nabla f = (\partial f_i / \partial x_j)_{i,j=1,...,d}.$$

Em particular, f é desta forma quando f é o gradiente de outra função F, isto é, $f = \nabla \mathrm{F} = \mathrm{grad}\, \mathrm{F}$. Neste caso, o seu gradiente

$$\nabla f = (\partial^2 \mathrm{F} / \partial x_i \partial x_j)_{i,j=1,...,d} = \begin{pmatrix} \frac{\partial^2 \mathrm{F}}{\partial x_1 \partial x_1} & \frac{\partial^2 \mathrm{F}}{\partial x_2 \partial x_1} & \cdots & \frac{\partial^2 \mathrm{F}}{\partial x_d \partial x_1} \\ \frac{\partial^2 \mathrm{F}}{\partial x_2 \partial x_1} & \frac{\partial^2 \mathrm{F}}{\partial x_2 \partial x_2} & \cdots & \frac{\partial^2 \mathrm{F}}{\partial x_2 \partial x_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 \mathrm{F}}{\partial x_d \partial x_1} & \frac{\partial^2 \mathrm{F}}{\partial x_d \partial x_2} & \cdots & \frac{\partial^2 \mathrm{F}}{\partial x_d \partial x_d} \end{pmatrix}.$$

é a matriz $\mathrm{H_F}$ das segundas derivadas de F, a *matriz de Hesse* de F.

Se $\mathrm{H_F}$ é invertível em x (o que corresponde a $f'(x)$), então o operador de Newton se torna

$$\mathrm{N_F}(x) = x + \mathrm{H_F}(x)^{-1}\mathrm{F}(x)$$

O método de Newton continua a convergir rapidíssimo; porém, a computação da matriz de Hesse de F em cada ponto é computacionalmente custoso (enquanto o método do máximo declive se contenta com a do mero gradiente). Contudo, se a dimensão d não é excecionalmente grande, continua a ser muito mais rápido.

Método do Máximo Declive. Usamos o método *numérico* do `Gradient Descent`, o *Método do Máximo Declive* para minimizar $\mathrm{E_D}(w)$.

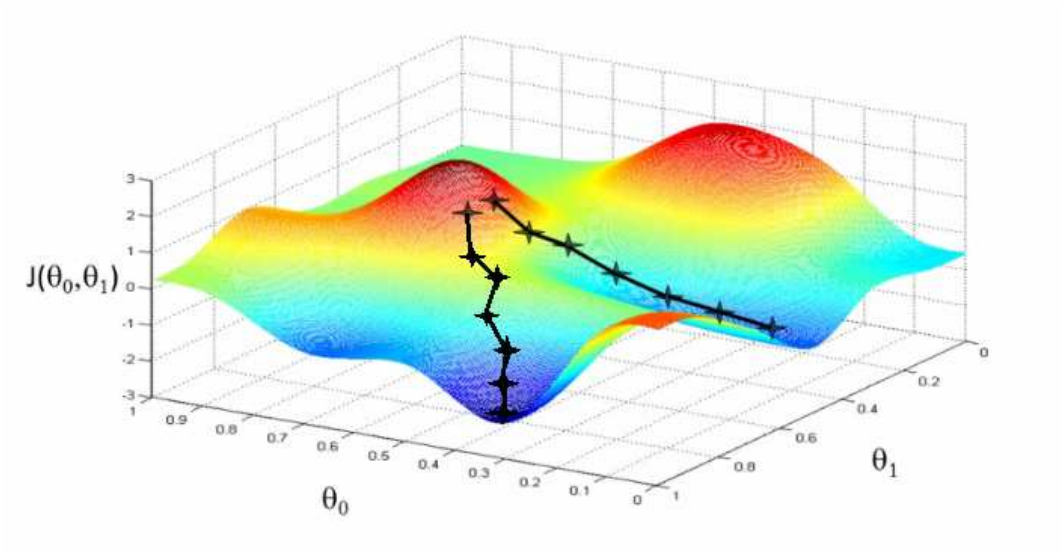


Figura 19: Seguindo iterativamente o máximo declive

Como a função $\mathrm{E_D}(w)$ é convexa, tem um único mínimo (em contraste à imagem). Este pode ser aproximado como segue:

Pelo gradiente, podemos calcular as derivadas

$$\frac{\partial f}{\partial h}(x) := \lim_{t \to 0} \frac{f(x + t \cdot h) - f(x)}{t}$$

em todas as direções $h \in \mathbb{R}^d$ (enquanto, até agora, unicamente, nas dos eixos-x_1, …, x_d).

A derivada $\frac{\partial f}{\partial h}(x)$ da função f em direção h para $h \in \mathbb{R}^d$ é dada por

$$\frac{\partial f}{\partial h}(x) = \nabla f(x) \cdot h^{\mathrm{T}} := \frac{\partial f}{\partial x_1}(x)h_1 + \cdots + \frac{\partial f}{\partial x_d}(x)h_d.$$

onde o produto

$$x \cdot y^{\mathrm{T}} := x_1 y_1 + \cdots + x_d y_d$$

é o *produto escalar* entre os vetores x e y. Tem-se

$$x \cdot y^{\mathrm{T}} = ||x||||y|| \cos \alpha$$

onde α é o ângulo entre x e y.

Em particular, $x \cdot y$ atinge o seu valor máximo, entre todos os vetores de um mesmo comprimento fixo, por exemplo entre todos os vetores unitários, quando x e y são múltiplos um do outro. Em particular, a derivada em x_0 é máxima, se derivamos em direção de $\nabla f(x_0)$; Isto é, geometricamente: O vetor gradiente indica a direção do máximo aclive (ou declive) cujo grau é medido pelo comprimento do gradiente!

Computação. Resumimos o algoritmo para a melhor escolha da hipótese

1. pelo método de Newton aplicado à derivada $\nabla_w \mathrm{E_D}(w)$, e
2. pelo método do máximo declive aplicado à função $\mathrm{E_D}(w)$.

Pelo Método de Newton.

1. Inicializa o vetor de pesos $w(0)$. $(*)$
2. Para $t = 0, 1, \ldots$
    1. Calcula a matriz de Hesse $\mathrm{H}(t) := \nabla^2 \mathrm{E_D}(w(t))$ em $w(t)$.
    2. Calcula o novo zero $w(t+1) = \mathrm{N}_{\nabla f}(w(t)) = \nabla w(t) + \mathrm{H}(t)^{-1} \nabla f(w(t))$.
    3. Decide se continuar (com $w(t+1)$) ou terminar a iteração. $(\dagger)$
3. Devolve o último $w(t)$ (isto é, onde a iteração parou).

onde recordamos

$$\nabla \mathrm{E_D}(w) = -\mathrm{N}^{-1}\Big(\sum_{n=1,\ldots,\mathrm{N}} y_n x_n / (1 + \mathrm{e}^{y_n w^{\mathrm{T}} x_n})\Big)^{\mathrm{T}}$$

e

$$\nabla^2 \mathrm{E_D}(w) = \nabla(\nabla \mathrm{E_D}(w))$$

Quanto a (∗), à escolha inicial de $w(0)$, uma escolha comum para $w(0)$ é 0, ou, para evitar o risco de uma má escolha específica, escolher cada peso $w_1, \ldots,$ w_d independentemente por uma distribuição normal.

Quanto a (†), um bom critério para a decisão sobre a terminação da iteração, ele é dado pela satisfação de

- um limiar mínimo do erro $\mathrm{E_D}$ (isto é, terminamos quando $\mathrm{E_D}$ for menor do que um certo limiar, ou seja, quando o erro é suficientemente pequeno)

enquanto $t = 0,1,\ldots$ não chegou a

- um número máximo T de iterações (como um critério de segurança caso o primeiro critério acima não se aplique).

Pelo Método do Máximo Declive.

1. Inicializa o vetor de pesos $w(0)$ (e define a margem de traslado $\eta > 0$). (∗)
2. Para $t = 0,1,\ldots$
    1. Calcula o gradiente $g_t := \nabla \mathrm{E_D}(w(t))$ em $w(t)$.
    2. Põe a direção $v_t := -g_t/\|g_t\|$.
    3. Atualiza o vetor de pesos $w(t+1) = w(t) + \eta \cdot v_t$.
    4. Decide se continuar (com $w(t+1)$) ou terminar a iteração. (†)
3. Devolve o último $w(t)$ (isto é, onde a iteração parou).

onde recordamos

$$\nabla \mathrm{E_D}(w) = -\mathrm{N}^{-1}(\sum_{n=1,\ldots,\mathrm{N}} y_n x_n/(1+\mathrm{e}^{y_n w^{\mathrm{T}} x_n}))^{\mathrm{T}}$$

Quanto a (∗), às escolhas iniciais de η e $w(0)$,

- uma escolha comum para η é $\eta = 0{,}1$, e
- uma escolha comum para $w(0)$ é 0, ou, para evitar o risco de uma má escolha específica, escolher cada peso $w_1, \ldots, w_d$ independentemente por uma distribuição normal.

Quanto a (†), um bom critério para a decisão sobre a terminação da iteração, ele é dado pela satisfação de

- um limiar mínimo do erro E_D (isto é, terminamos quando E_D for menor do que um certo limiar, ou seja, quando o erro é suficientemente pequeno), e
- um limiar mínimo da variação do erro ΔE_D (isto é, terminamos quando ΔE_D for menor do que um certo limiar, ou seja, as variações do erro são suficientemente pequenas),

enquanto $t = 0, 1, \ldots$ não chegou a

- um número máximo T de iterações (como um critério de segurança caso os dois critérios acima não se apliquem).

5 Transformação Linear

Estudaremos como transformar um problema polinomial em um problema linear, isto é,

- de hipóteses que dependem polinomialmente das entradas
- a hipóteses que dependem linearmente das entradas.

Aí, podemos aplicar um dos nossos algoritmos lineares, como o `Perceptron`, a Regressão Linear ou Logística; depois, invertemos a linearização para responder ao problema inicial.

Em mais detalhes: Dada uma amostra D, o algoritmo A escolhe a hipótese h_{D} em H que minimiza o erro específico $\mathrm{E}_{\mathrm{D}}(h_{\mathrm{D}})$. Em todos os algoritmos lineares, isto é, o `Perceptron`, a Regressão Linear ou Logística, temos a parametrização

$$\mathrm{H} \leftrightarrow \mathbf{R} \times \cdots \times \mathbf{R}$$

por

$$h_0(w^{\mathrm{T}} \cdot) \leftarrow\!\shortmid w$$

onde h_0 é

- o sinal no `Perceptron`,
- a identidade id na Regressão Linear, e
- a função logística $\theta = \exp/(1 + \exp)$ na Regressão Logística.

Isto é, A minimiza a função

$$w \mapsto \mathrm{E}_{\mathrm{D}}(h_0(w^{\mathrm{T}} \cdot))$$

sobre $\mathbf{R} \times \cdots \times \mathbf{R}$; os pontos da amostra $x_1, \ldots, x_{\mathrm{N}}$ entram como *constantes*! Portanto, *antes* de ser dada a amostra, podemos adaptar a entrada X ao algoritmo linear; isto é, aplicar uma função $\Phi\colon \mathrm{X} \to \mathrm{X}'$ que diminua

$$\mathrm{E}_{\Phi(\mathrm{D})}(h_0(w'^{\mathrm{T}} \cdot)) := \mathrm{E}_{\mathrm{D}}(h_0(w'^{\mathrm{T}} \Phi(\cdot)))$$

(onde w' é agora um vetor de pesos tal que $w'^{\mathrm{T}} \cdot$ seja definido sobre X′).

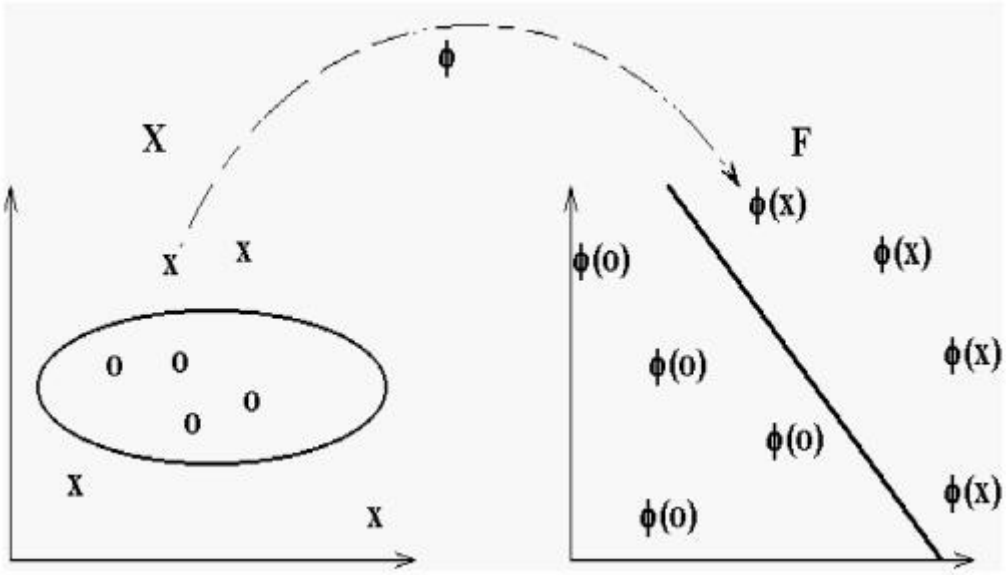


Figura 20: Aplicar uma transformação polinomial para separar linearmente

5.1 Linearizar Elipses em torno da origem

Usamos o `Perceptron` sobre o plano $X = \mathbf{R} \times \mathbf{R}$ para separar os pontos do tipo +1, *positivos*, dos pontos do tipo −1, *negativos*, de uma amostra $x_1, \ldots, x_N$ por uma reta. Suponhamos que a amostra não é linearmente separável, mas que todos os pontos positivos estão cercados por um círculo.

Um círculo no plano com raio r é dado pela equação

$$x_1^2 + x_2^2 = r,$$

isto é, dado por uma equação polinomial quadrática, e a função

$$h(x) = \text{sinal}(x_1^2 + x_2^2 - r)$$

separa entre os pontos positivos e negativos. Se escrevemos

$$h(x) = \text{sinal}(w_0' x_0' + w_1' x_1' + w_2' x_2')$$

onde

$$w_0' = -r, w_1' = 1, w_2' = 1 \quad \text{e} \quad x_0' = 1, x_1' = x_1^2, x_2' = x_2^2,$$

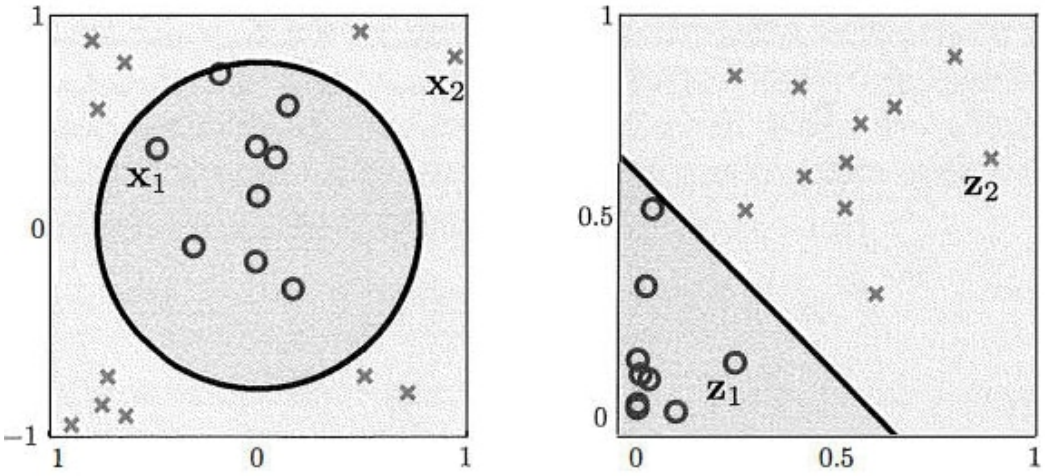


Figura 21: Pontos Cercados

então h é uma função linear (ignorando o sinal) h' em $x' = (x'_0, x'_1, x'_2)$ com (o vetor de) pesos $w' = (w'_0, w'_1, w'_2)$. Explicitamente,

$h = \text{sinal}(h' \circ \Phi)$ com $h' = w'\mathrm{T}\cdot$ e $\Phi\colon \mathrm{X} \to \mathrm{X}'$ dada por $(x_0, x_1, x_2) \mapsto (x_0, x_1^2, x_2^2)$.

A função $\Phi\colon \mathrm{X} \to \mathrm{X}'$ é chamada a *transformação de características*. A transformação leva toda função linear $h'\colon \mathrm{X}' \to \mathrm{Y}$ sobre o espaço transformado X' por

$$h := h' \circ \Phi.$$

a uma função, não necessariamente linear, $h\colon \mathrm{X} \to \mathrm{Y}$ sobre o espaço original.

Se h' anula o erro específico, $\mathrm{E}_{\Phi(\mathrm{D})}h' = 0$, então igualmente h o anula, $\mathrm{E}_{\mathrm{D}} = \mathrm{E}_{\Phi(\mathrm{D})}h' = 0$.

Se Φ é estabelecida antes de ser dada a amostra, então

$$\mathrm{d}_{\mathrm{VC}}(\mathrm{H}_\Phi) \leq \mathrm{d}_{\mathrm{VC}}(\mathrm{H}')$$

onde H_Φ é o espaço das hipóteses $\{h \circ \Phi : h \text{ em } \mathrm{H}\}$ e H' é o espaço das hipóteses sobre X'. Vale somente a desigualdade, mas não necessariamente a igualdade, isto é, não necessariamente toda função em H' pode ser obtida por pré-composição de uma função em H com Φ.

Por exemplo, se Φ é a transformação acima e o algoritmo é o `Perceptron`,

$$\Phi\colon \mathrm{X} \to \mathrm{X}' \text{ dada por } (x_0, x_1, x_2) \mapsto (x_0, x_1^2, x_2^2).$$

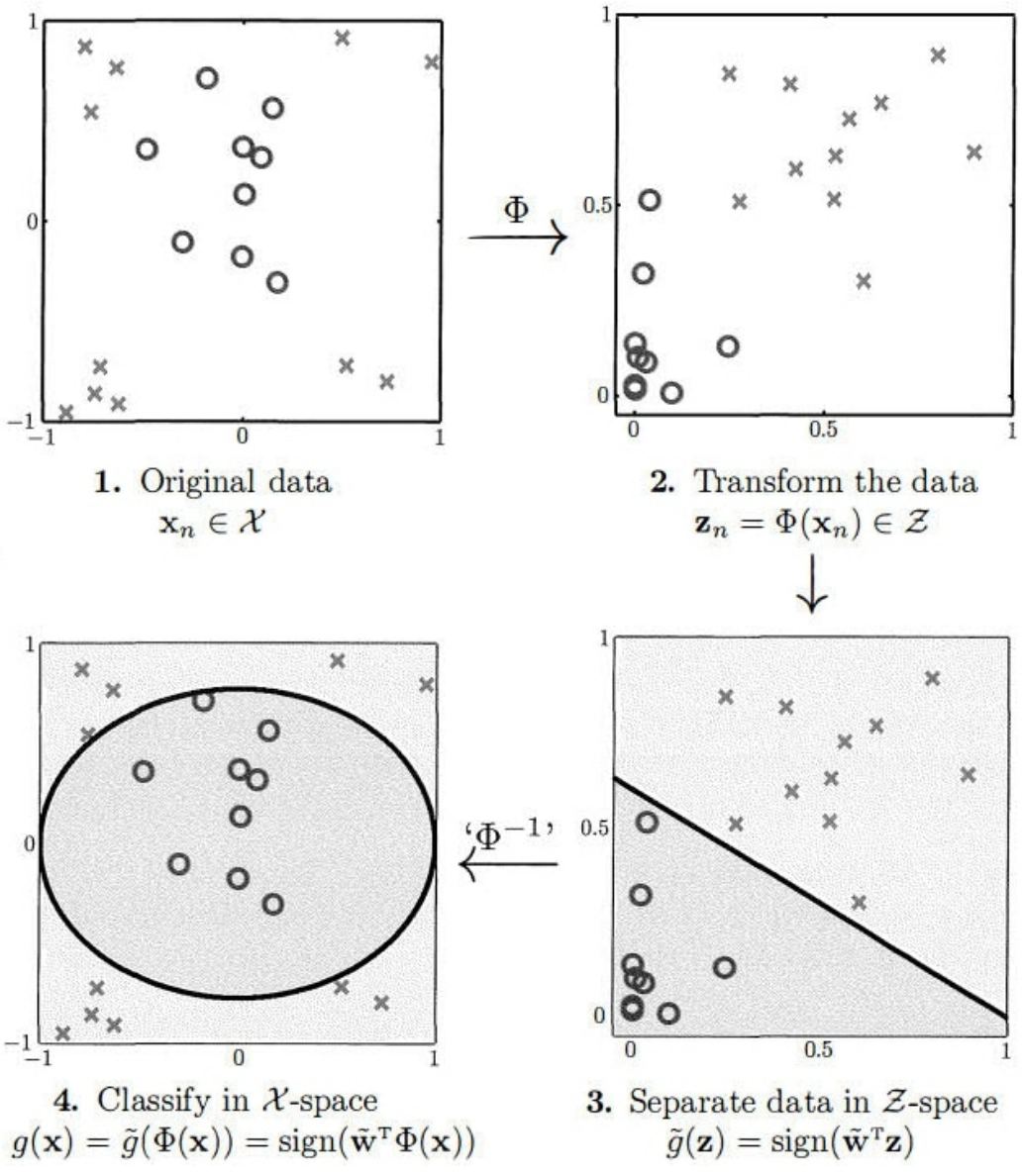


Figura 22: Ida e volta ao espaço transformado

então

$$d_{VC}(H_\Phi) \leq d_{VC}(H') = d_{VC}(H) = 3.$$

Com efeito, a mesma configuração de pontos como em Figura 4 é também linearmente inseparável por uma elipse (que é centrada na origem).

Observação. A aplicação Φ tem de ser escolhida *antes* de ser dada a amostra; isto é, não pela avaliação dos dados, mas pela nossa compreensão do problema. Caso contrário, estamos efetivamente aumentando H e assim $d_{VC}(H)$.

5.2 Generalização da Transformação

Restamos no plano, isto é, $d :=$ dimensão $X = 2$. Enquanto a dimensão de X' é igualmente $d = 2$, somente podemos separar pontos por elipses centradas na origem do plano, o ponto (0,0). Contudo, para obter todas as curvas quadráticas, precisamos aumentar a dimensão de X' a $d' = 5$ para aplicar

$$\Phi\colon (1, x_1, x_2) \mapsto (1, x_1, x_2, x_1^2, x_1 x_2, x_2^2).$$

(Descontamos a primeira coordenada pela sua entrada fixa x_0.) Por esta transformação de características, podemos, por exemplo, trasladar o centro das elipses da origem (0,0) a qualquer ponto no plano.

Mais geralmente, podemos considerar a transformação de características para curvas cúbicas, enfim, para polinômios de grau Q qualquer. Uma tal transformação é chamada a *transformação polinomial de grau* Q.

Mas, cautela! Agora $d = 2$ tornou se $d' = Q(Q+3)/2$. Além de aumentar o esforço computacional do algoritmo, aumenta a dimensão de Vapnik-Chervonenkis, por exemplo, para o `Perceptron` de $d_{VC}(H) = d + 1 = 3$ à $d_{VC}(H') = d' + 1 = [Q(Q+3)/2] + 1$.

Surge outra vez o dilema entre o erro geral e o erro específico:

- Se aumenta d', então
  - diminui E_D, mas
  - aumenta $d_{VC}(H')$ e então Δ_D.
- Se diminui d', então
  - aumenta E_D, mas
  - diminui $d_{VC}(H')$ e então Δ_D.

5.3 O Artifício do Núcleo

Dada uma transformação $\Phi\colon \mathrm{X} \to \mathrm{X}'$ do espaço vetorial X de dimensão *menor* ao espaço vetorial (de re-descrição) X′ de dimensão *maior*. O artifício no *Artifício do Núcleo* consiste em calcular o *núcleo* (= produto escalar transformado) $\mathrm{K} := \Phi(\cdot)\cdot\Phi(\cdot)$ sobre X′ diretamente a partir de X, sem conhecer X′ nem Φ! Como X é de dimensão menor, fica computacionalmente mais econômico.

Em vez de definir a transformação Φ, o produto K é definido diretamente e a existência de Φ é implícita; se K satisfaz uma certa *positividade*, ela é assegurada pelo *Teorema de Mercer*.

Com efeito, recordamos que o objetivo final num problema de aprendizado é a escolha da melhor hipótese h em H pelo algoritmo A a partir da amostra D. Para nós, *melhor* quer dizer *minimizar* certo erro $\mathrm{E}_{\mathrm{D}}h$ sobre a amostra D. Em todos os algoritmos lineares em Secção 4, identificamos h com um vetor de pesos w e obtivemos

- no `Perceptron`, $\mathrm{E}_{\mathrm{D}}h = \#\{x \text{ em D com } w^{\mathrm{T}}x \neq y(x)\}$,
- na `Regressão Linear`, $\mathrm{E}_{\mathrm{D}}h = [(w^{\mathrm{T}}x_1 - y_1)^2 + \cdots + (w^{\mathrm{T}}x_{\mathrm{N}} - y_{\mathrm{N}})^2]/\mathrm{N}$,
- no `Perceptron`, $\mathrm{E}_{\mathrm{D}}h = \mathrm{N}^{-1}\sum_{n=1,\ldots,\mathrm{N}} \log(1 + \mathrm{e}^{-y_n w^{\mathrm{T}} x_n})$.

A observação que importa é que em todos os três casos, para minimizar o erro $\mathrm{E}_{\mathrm{D}}(h)$, basta saber o produto escalar $w^{\mathrm{T}}x$! Em particular, se D é transformado por uma aplicação $\Phi\colon \mathrm{X} \to \mathrm{X}'$, é suficiente saber o valor do produto escalar transformado $w^{\mathrm{T}}\Phi(x)$ para todos os x em D; portanto, é desnecessário conhecer Φ em si! Na prática, em vez de definir Φ, é diretamente definido o produto escalar transformado $\mathrm{K}(w, x) := w^{\mathrm{T}}x$; o Teorema de Mercer define Φ implicitamente.

Princípio. Em cada um dos nossos algoritmos lineares em Secção 4, o erro específico E_{D} dependia no final somente de um produto escalar. Após uma transformação linear $\Phi\colon \mathrm{X} \to \mathrm{X}'$, este toma a forma:

$$\Phi(x)\cdot\Phi(y)$$

Si o espaço vetorial X′ é de alta dimensão, então a computação deste produto fica impraticável. A ideia é então substituir este produto por uma função K da forma

$$\mathrm{K}(x, y) = \Phi(x)\cdot\Phi(y).$$

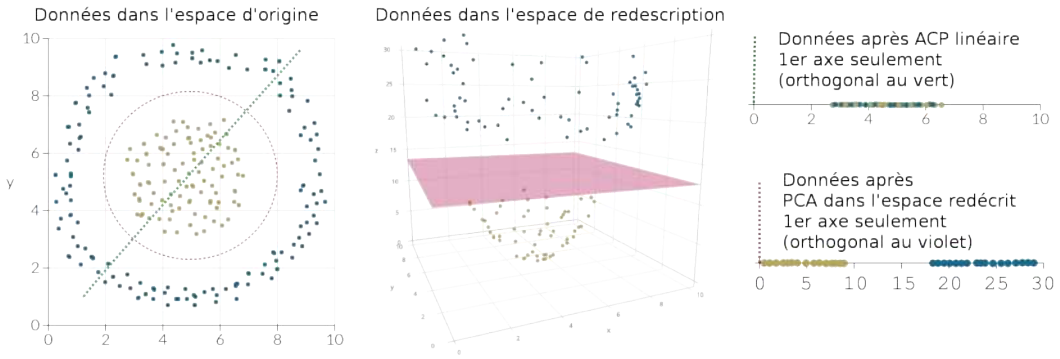


Figura 23: Aumentar a dimensão por uma transformação polinómial para poder separar linearmente

Teorema de Mercer. O Teorema de Mercer (James Mercer, matemático inglês do século 19) mostra que uma função contínua, simétrica e semi-definida K pode ser expressa por um produto escalar em um espaço vetorial de grande dimensão.

Mais exatamente, se o domínio $X \times X$ de K é um espaço mensurável, e se K é *semi-definida*, isto é,

$$\sum_{i,j} K(x_i, x_j) c_i c_j \geq 0$$

para qualquer conjunto finito $\{x_1, \ldots, x_n\}$ no domínio X e conjunto finito $\{c_1, \ldots, c_N\}$ no contra-domínio Y (tipicamente $\mathbf{R}$), então existe uma função $\Phi\colon X \to X'$ onde X' é um espaço vetorial pré-hilbertiano (provavelmente de maior dimensão) tal que:

$$K(x, y) = \Phi(x) \cdot \Phi(y)$$

Vantagem. Assim, o produto escalar em um espaço vetorial de grande dimensão é substituído por um núcleo que é mais facilmente calculável. Esta é uma maneira econômica de transformar um classificador linear em um classificador não-linear. Pelo núcleo, a transformação $\Phi\colon X \to X'$ nem precisa ser explicitada. Desta maneira, mesmo se o espaço vetorial X' é de dimensão infinita, o produto escalar transformado, o núcleo, permanece calculável como exemplificado pelo *núcleo gaussiano radial-simétrica*:

$$K(x, y) = \exp(-\|\mathbf{x} - \mathbf{y}\|^2 / 2\sigma^2).$$

Ele é muito usado para SVMs.

Outro exemplo, popular no processamento automático de linguagem natural, é o *núcleo polinomial*

$$k(x,y) = (x \cdot y)^d$$

para um inteiro $d > 1$.

Aplicações. O artifício do núcleo foi apresentada pela primeira vez em 1964 por Aizerman, mas teve pouca repercussão. Ele pegou pelo artigo "A Training Algorithm for Optimal Margin Classifiers" por Boser et al. (publicado em 1992) que fundou a teoria dos Support Vector Machines. Desde então, aplica-se em algoritmos de aprendizado de máquina como

- Support Vector Machines,
- análise de componentes principais, e
- análise da discriminante linear.

6 Redes Neurais

Recordemo-nos de que, formalmente, em um problema de aprendizado, temos

- a **entrada** X,
- a **saída** Y,
- os **dados** ou a *amostra* $D = \{x_1, ..., x_N\}$ (= um conjunto finito em X sobre o qual a *função alvo* é conhecida, isto é, $y_1 = y(x_1)$, ..., $y_N = y(x_N)$ são conhecidos, e
- a **função alvo** (ou *função de regressão* ou *classificadora*) $y: X \to Y$ (= a função ótima, que sempre acerta, a ser aproximada; chamamo-la também de função *divina* para exprimir que existe e é aproximável com certa probabilidade, mas inatingível)

e o **método de aprendizado**:

- as **hipóteses** $H = \{h: X \to Y\}$ (= um conjunto de funções que aproximam y),
- uma **medida de erro** $E: X \times X \to [0, \infty]$ que mede a diferença entre os valores da hipótese e os da função alvo, e
- um **algoritmo** A que escolhe uma função h em H que aproxima" y "o melhor", em particular, minimiza E sobre D.

Uma **rede neural** (*feed-forward* ou *sem realimentação*) consiste em *neurónios* agrupados em camadas onde a entrada de cada neurónio são as saídas de todos os neurónios da camada anterior.

A primeira camada é a da **entrada**, e a última a da **saída**; as outras chamam se camadas **ocultas**.

Se a rede neural tem múltiplas camadas ocultas, então é uma rede neural **profunda**, as quais são usadas no *Deep Learning*, no aprendizado profundo.

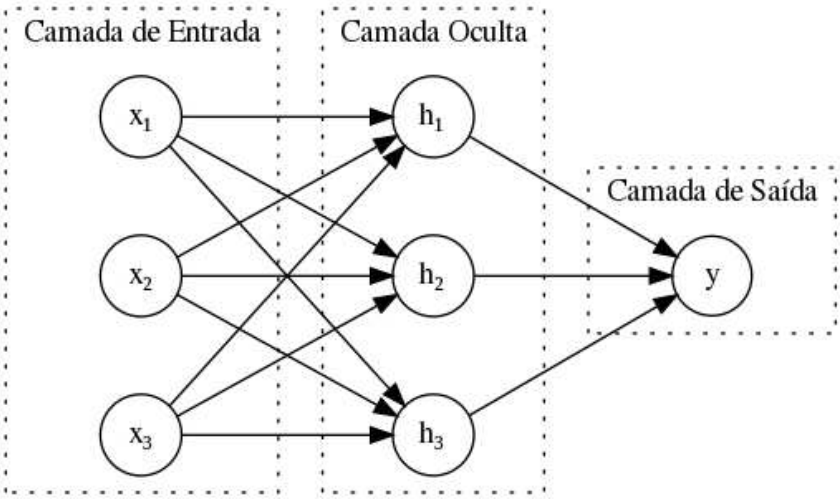
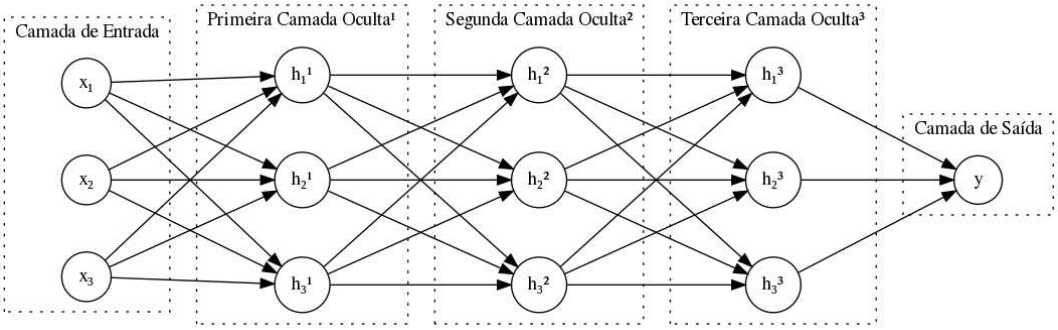


Figura 24: Uma rede neural de uma única camada oculta com três neurónios.



Cada **neurónio** é uma função

$$h\colon \mathbb{R} \times \cdots \times \mathbb{R} \to \mathbb{R}$$

que é uma composição $h(x) = g(f(x))$ de

- uma função afim, $f(x) = ax + b$ (cujo gráfico é uma reta), e
- uma função de *ativação* $g(x)$ que não é afim.

6.1 Funções de Ativação

Para a última camada (para tarefas de classificação) usa-se frequentemente uma Função de Ativação Softmax definida por:

$$\text{Softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{\text{K}} e^{x_j}}$$

onde $\mathbf{x}$ é um vetor de (K) dimensões e (i) é o índice do elemento.

Ela normaliza a saída de um vetor de valores para uma distribuição de probabilidade.

Função de Ativação ReLU (Unidade Linear Retificada). Definida por:

$$\text{ReLU}(x) = \max(0, x)$$

- Não-saturante, o que reduz o problema de gradientes que desaparecem (vanishing gradients) para valores positivos de (x).
- Eficiência computacional devido à sua simplicidade.

Função de Ativação Leaky ReLU. Definida por:

$$\text{LeakyReLU}(x) = \begin{cases} x & \text{se } x > 0 \\ \alpha x & \text{se } x \leq 0 \end{cases}$$

onde α é um pequeno coeficiente positivo.

- Visa resolver o problema de neurônios "mortos" em ReLU, permitindo um pequeno gradiente quando (x) é negativo.
- Mantém as vantagens computacionais da ReLU.

Função de Ativação Sigmoide. Definida por:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- Mapeia a entrada para um intervalo entre 0 e 1.
- Suave e diferenciável em todo o seu domínio.
- Suscetível ao problema de gradientes que desaparecem para entradas com grande magnitude.

Função de Ativação Tangente Hiperbólica (tanh). Definida por:

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

- Mapeia a entrada para um intervalo entre -1 e 1.
- Suave e diferenciável em todo o seu domínio.
- Também pode sofrer do problema de gradientes que desaparecem, embora em menor grau que a sigmoide.

6.2 Rede neural de camada única

Para entender esses diagramas, vamos começar o curso com redes neurais de camada única: Uma rede Perceptron que tem uma única camada oculta e uma única saída consiste em um único neurônio.

Um **neurônio** descreve uma classe de problemas de aprendizado cujas hipóteses são caracterizadas pela composição de uma função de saída não linear com uma função afim; essa classe inclui, entre outros métodos,

- regressão linear,
- o Perceptron ou discriminador linear para classificação binária,
- multiClass-Perceptron para classificação finita, ou seja, entre um número finito de classes,
- regressão logística,
- regressão logística multinomial (ou regressão softmax).

Neles,

- a **entrada** é $X = \mathbb{R} \times \cdots \times \mathbb{R}$ (possivelmente unidimensional), e
- a **saída** Y depende do tipo de neurônio:
  - para regressão linear, $Y = \mathbb{R}$,
  - para o Perceptron, é binário, $Y = \{\pm 1\}$,
  - para o MultiClass-Perceptron, ele é finito, $Y = \{0, 1, \ldots, n\}$,
  - para a regressão logística, $Y = [0, 1]$,
  - para regressão logística multinomial (ou regressão softmax), Y consiste em matrizes finitas $(y_1, y_2, \ldots, y_n)$ cujas entradas não negativas somam 1.

uma rede neural com um único neurônio](./images/single-perceptron.png)

O **método de aprendizado**:

- cada hipótese é a composição $h = g \circ f$ de

    1. uma função de saída não linear, geralmente não constante, limitada, monotonamente crescente e contínua. Os exemplos são

        - para regressão linear, a identidade $g = \mathrm{id}$,
        - para o Perceptron, o sinal $g(x) = 1$ se $x \geq 0$ e -1 caso contrário,
        - para o MultiClass-Perceptron, $g = \mathrm{argmax}$, ou seja, o argumento i com $f_i(x) = \max\{f_1(x), \ldots, f_n(x)\}$,
        - para regressão logística
            * a função logística $\theta\colon \mathbb{R} \to [-1,1]$ definida por $\theta(x) = 1/(1+e^x)$, ou
            * a função tangente hiperbólica $\theta = \tanh\colon \mathbb{R} \to [-1,1]$ definida por $\tanh := (\mathrm{e}^s - \mathrm{e}^{-s})/(\mathrm{e}^s + \mathrm{e}^{-s})$.
        - para a regressão logística multinomial, a função $\mathrm{Softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{\mathrm{K}} e^{x_j}}$

    2. uma função *final* h, ou seja, uma função que é definida por determinados **coeficientes** w_1, w_2, … e por um determinado **limiar** b:

        $$h(x) = w^{\mathrm{T}} x + b = w_1 x_1 + \cdots + w_d x_d + b.$$

        Se a função tiver um único argumento, ela será afim se, e somente se, seu gráfico for uma linha reta. O peso corresponde ao grau de inclinação (também chamado de *coeficiente de ângulo*) e o limite à altura em que o gráfico cruza a ordenada.

- uma **medida de erro** $\mathrm{E}\colon \mathrm{X} \times \mathrm{X} \to [0, \infty]$ que mede a diferença entre os valores da hipótese e os da função-alvo:

    - para regressão linear, a distância ao quadrado,
    - para o Perceptron (e o MultiClass-Perceptron), a contagem do número de valores diferentes entre y e h,
    - para regressão logística (e multinomial), o log-likelihod, a máxima verossimilhança.

- um **algoritmo** A que escolhe uma função h em H que "aproxima y da melhor forma", ou seja, minimiza E em D; para saídas contínuas, como (um intervalo em) $\mathbb{R}$, o método (estocástico) de inclinação máxima é normalmente usado, mas o método de Newton também é possível como alternativa.

6.3 Teorema de aproximação universal para redes de camada única

O Teorema de Aproximação Universal para Funções de Ativação Não Polinomiais afirma que uma rede neural com uma única camada oculta pode aproximar qualquer função contínua em um domínio compacto, independentemente da dimensão dos espaços de entrada e saída, com qualquer grau de precisão desejado, desde que a função de ativação seja contínua e não polinomial e que possamos ter tantos neurônios na camada oculta quantos forem necessários para atingir a precisão desejada.

Teorema de aproximação universal para funções de ativação não polinomiais: Seja $\sigma : \mathbb{R} \to \mathbb{R}$ seja uma função contínua. Então, σ não é polinomial se e somente se, para toda função $f \in \mathrm{C}(\mathrm{K}, \mathrm{Y})$ definida em um conjunto compacto $\mathrm{K} \subseteq \mathbb{R}^n$ com valores em $\mathbb{R}^m$ para quaisquer n e m, e para todo $\varepsilon > 0$, existirem

- uma matriz $\mathrm{A} \in \mathbb{R}^{k \times n}$ e um vetor $b \in \mathbb{R}^k$,
- uma matriz $\mathrm{C} \in \mathbb{R}^{m \times k}$,

para algum $k \in \mathbb{N}$, de modo que $g : \mathbb{R}^n \to \mathbb{R}^m$ é definida por

$$g(x) = \mathrm{C} \cdot (\sigma \circ (\mathrm{A} \cdot x + b))$$

satisfaz

$$\sup_{x \in \mathrm{K}} \|f(x) - g(x)\| < \varepsilon.$$

Interpretação. O Teorema da Aproximação Universal oferece uma garantia teórica de que as redes neurais têm o potencial de modelar qualquer função contínua com qualquer nível de precisão, desde que haja uma função de ativação não polinomial adequada e um número suficientemente grande de neurônios na camada oculta.

Considere uma função f que desejamos aproximar. Essa função recebe um vetor de um espaço de n dimensões e produz um vetor em um espaço de m dimensões. O espaço do qual as entradas são retiradas é compacto, o que significa que é limitado e contém todos os seus pontos-limite, o que é uma situação comum em aplicações práticas.

A rede neural em questão, representada por $g(x)$, é um modelo matemático que tenta imitar o comportamento de f. A rede faz isso transformando o vetor de entrada x usando uma combinação de operações lineares e não lineares. A parte linear envolve a multiplicação de x por uma matriz A e a adição de um vetor b, enquanto a parte não linear é tratada com a aplicação da função de ativação σ ao resultado.

A dimensão k representa o número de neurônios na camada oculta da rede neural. Cada neurônio contribui para a capacidade da rede de capturar um aspecto diferente da função f. As matrizes A e C e o vetor b são os parâmetros da rede neural que são ajustados durante o processo de treinamento para minimizar a diferença entre a saída da rede e os valores reais da função.

A dimensão k é crucial porque determina a capacidade da rede neural de aproximar funções complexas. Um k maior significa mais neurônios na camada oculta, o que, por sua vez, significa mais flexibilidade para a rede neural ajustar a função f. Entretanto, nem sempre o simples fato de aumentar k produzirá melhores aproximações; os parâmetros da rede também devem ser ajustados de forma otimizada, geralmente por meio de treinamento.

Teorema de aproximação universal de Leshno para redes neurais feed-forward de camada única. Seja $\sigma : \mathbb{R} \to \mathbb{R}$ seja uma função de ativação contínua, não constante e não polinomial. Considere $K \subset \mathbb{R}^n$ como um conjunto compacto. Nosso objetivo é mostrar que o conjunto de funções da forma

$$f(x) = \sum_{i=1}^{m} c_i \sigma(a_i^{\mathrm{T}} x + b_i),$$

onde $c_i \in \mathbb{R}$, $a_i \in \mathbb{R}^n$, $b_i \in \mathbb{R}$ e $m \in \mathbb{N}$, é denso em $C(K, \mathbb{R})$, o espaço de funções contínuas em K com a norma do supremo.

Etapa 1: não densidade das funções de ativação polinomial. Seja $\sigma \in C(\mathbb{R}^n)$. Suponha que σ seja um polinômio de grau k. Então, $\sigma(w \cdot x + b)$ também é um polinômio de grau k para todo $w \in \mathbb{R}^n$ e $b \in \mathbb{R}$. O conjunto C_k de polinômios algébricos de grau no máximo k é exatamente o conjunto de funções que podem ser representadas por esse σ. Como C_k é um subconjunto adequado de $C(\mathbb{R}^n)$, ele não é denso em $C(\mathbb{R}^n)$.

Etapa 2: a densidade em $C(\mathbb{R}^n)$ implica a densidade em $C(\mathbb{R})$. Suponha que σ não seja um polinômio. Se $S_{\sigma,1}$ é denso em $C(\mathbb{R})$, então $S_{\sigma,n}$ é denso em $C(\mathbb{R}n)$, uma vez que o espaço V abrangido por funções da forma $f(w \cdot x)$, em que $f \in C(\mathbb{R})$ e $w \in \mathbb{R}^n$, é denso em $C(\mathbb{R}^n)$.

Etapa 3: as funções de ativação suave são densas em $C(\mathbb{R})$. Se $\sigma \in C^\infty(\mathbb{R})$, o conjunto de todas as funções com derivadas de todas as ordens, então $S_{\sigma,1}$ é denso em $C(\mathbb{R})$. Para isso, basta mostrar que $S_{\sigma,1}$ contém todos os monômios pelo Teorema de Weierstrass.

Para todo w e θ em $\mathbb{R}$, o quociente da diferença (de funções) $[\sigma((w+h)\cdot+\theta) - \sigma(w\cdot+\theta)]/h$ está em $S_{\sigma,1}$. Como σ é diferenciável e $S_{\sigma,1}$ é completo, a primeira derivada $\frac{d}{dw}\sigma(w\cdot+\theta)|_{w=0} = \cdot\sigma'(\theta)$ de $\sigma(hx)$ com relação a h em 0, como limite desses quocientes de diferença, está em $S_{\sigma,1}$. Como σ não é constante, há θ_0 de modo que $\sigma'(\theta) \neq 0$. Assim, descobrimos que $\cdot$ está em $S_{\sigma,1}$.

Iterativamente, $\frac{d^k}{dw^k}\sigma(w\cdot+\theta)|_{w=0} = \cdot^k\sigma'(\theta)$ em $S_{\sigma,1}$. Como σ não é polinomial, existe para cada k algum θ_j tal que $\sigma'(\theta_k) \neq 0$.

Portanto, todo monômio $\cdot^k$ está em $S_{\sigma,1}$.

Observação: Neste ponto, já provamos o Teorema de Aproximação Universal para funções de ativação sigmoidais clássicas (suaves).

Infelizmente, a função de ativação mais comum atualmente, a ReLU, não é suave em 0; portanto, é necessário trabalho adicional para uma função de ativação meramente contínua:

Etapa 4: Aproximação de funções de suporte compactas. Para $\varphi \in C_c^\infty(\mathbb{R})$, o espaço de funções suaves com suporte compacto, $\sigma * \varphi \in S_{\sigma,1}$, em que $*$ denota convolução.

Isso é comprovado pela aproximação da integral da convolução usando uma soma de Riemann, que converge uniformemente para $\sigma * \varphi$ em qualquer intervalo compacto.

Etapa 5: densidade em $C(\mathbb{R})$ para convolução não polinomial. Se $\sigma * \varphi$ não for um polinômio para algum $\varphi \in C_c^\infty(\mathbb{R})$, então $S_{\sigma,1}$ é denso em $C(\mathbb{R})$.

Isso decorre das etapas 3 e 4, já que $\sigma * \varphi$ pode ser demonstrado como sendo suave.

Neste ponto, resta mostrar que isso só pode ocorrer se σ for polinomial:

Etapa 6: a convolução polinomial implica a ativação polinomial. Se $\sigma * \varphi$ for um polinômio para todos os $\varphi \in C_c^\infty(\mathbb{R})$, então existe um $m \in \mathbb{N}$ de modo que $\sigma \cdot \varphi$ seja um polinômio de grau no máximo m para todos os $\varphi \in C_c^\infty(\mathbb{R})$.

Etapa 7: a convolução polinomial implica a ativação polinomial. Se $\sigma * \varphi$ é um polinômio de grau no máximo m para todos os $\varphi \in C_c^\infty(\mathbb{R})$, então σ é um polinômio de grau no máximo m (em quase todos os lugares).

Conclusão. O conjunto de combinações lineares finitas de translações e dilatações de σ é denso em $C(K, \mathbb{R})$. Isso significa que, para qualquer função contínua $g : K \to \mathbb{R}$ e qualquer $\epsilon > 0$, existe uma função $f \in \mathcal{A}$ tal que $\|f - g\|_\infty < \epsilon$, em que $\| \cdot \|_\infty$ denota a norma do supremo.

6.4 Rede neural de múltiplas camadas

Uma rede neural feedforward consiste em várias camadas:

- a camada de entrada,
- camadas ocultas e
- a camada de saída.

uma rede neural com duas camadas ocultas](./images/multilayer-perceptron.png)

Hipótese. Cada camada é uma função de várias entradas e saídas $\mathbb{R} \times \cdots \times \mathbb{R} \to \mathbb{R} \times \cdots \times \mathbb{R}$ da seguinte forma:

1. A camada de entrada é a identidade,
2. cada camada oculta é uma matriz $\mathbf{h} = (h_1, h_2, \ldots)$ de **neurônios**. Cada neurônio h é a composição $h = g \circ f$ de
    1. uma função de ativação g *fixa*, ou seja, a mesma para todos os neurônios em todas as camadas *ocultas* (mas não para as da camada de saída). Por exemplo, a função tangente hiperbólica $\theta = \tanh\colon \mathbb{R} \to [-1,1]$ definida por $\tanh := (e^s - e^{-s})/(e^s + e^{-s})$.
    2. uma função afim f que é definida por certos **coeficientes** w_1, w_2, $\ldots$ e por um certo **limiar** b, ou seja

$$f(x) = w^{\mathrm{T}} x + b = w_1 x_1 + \cdots + w_d x_d + b$$

3. A camada de saída é um neurônio em que a função de ativação é uma função de saída que pode ter uma forma diferente, por exemplo,
    - para regressão linear, a identidade $g = \mathrm{id}$,
    - para o Perceptron, o sinal $g(x) = 1$ se $x \geq 0$ e -1 caso contrário,
    - para o MultiClass-Perceptron, $g = \mathrm{argmax}$, ou seja, o argumento i com $f_i(x) = \max\{f_1(x), \ldots, f_n(x)\}$,
    - para regressão logística
        - a função logística $\theta\colon \mathbb{R} \to [-1,1]$ definida por $\theta(x) = 1/(1+e^x)$, ou
        - a função tangente hiperbólica $\theta = \tanh\colon \mathbb{R} \to [-1,1]$ definida por $\tanh := (e^s - e^{-s})/(e^s + e^{-s})$.
    - para regressão logística multinomial, a função ‘softmax

Se todos os neurônios da mesma camada tiverem o mesmo número de entradas que os neurônios da camada anterior, então a rede perceptron é *totalmente conectada*;

A função (de hipótese h que se aproxima da função divina) de uma **rede neural de propagação direta** é a composição sucessiva dos múltiplos da função para cada camada

$$h = \mathbf{h}^1 \circ \mathbf{h}^2 \circ \cdots$$

6.5 Teorema de aproximação universal para redes ReLU limitadas por largura

Lembremos que a função de ativação da ReLU (Unidade Linear Retificada) é definida como $\sigma(x) = \max(0, x)$.

O Teorema de Aproximação Universal para Redes ReLU Limitadas por Largura estende o Teorema de Aproximação Universal (UAT) clássico para redes ReLU limitadas por largura.

Teorema: Para qualquer função integrável de Lebesgue $f : \mathbb{R}^n \to \mathbb{R}$ e para qualquer precisão desejada $\epsilon > 0$, existe uma rede neural ReLU totalmente conectada $\mathbf{A}$ com uma largura $d_m \leq n + 4$ que pode aproximar f com um erro de L^1 de ϵ. A função $\mathrm{F}_{\mathbf{A}}$ representada pela rede $\mathbf{A}$ satisfaz:

$$\int_{\mathbb{R}^n} |f(x) - \mathrm{F}_{\mathbf{A}}(x)| \, dx < \epsilon.$$

Esse teorema mostra que as redes com uma largura limitada, especificamente não mais do que quatro unidades maiores do que a dimensão de entrada, ainda podem aproximar qualquer função integrável de Lebesgue arbitrariamente bem no sentido de L^1 em todo o espaço euclidiano.

Comparação com o Teorema de Leshno. O UAT com limite de profundidade de Leshno mostrou que as redes feedforward multicamadas com uma função de ativação não polinomial, como a função ReLU, são densas no espaço de funções contínuas em subconjuntos compactos de $\mathbb{R}^n$ com relação à norma L^∞.

Esse teorema com limite de largura difere do teorema de Leshno em vários aspectos importantes:

1. **Espaço funcional**: O teorema de Leshno considera funções contínuas em domínios compactos, enquanto o teorema de limite de largura considera funções integráveis de Lebesgue em todo o espaço euclidiano.
2. **Norma**: O teorema de Leshno usa a norma L^∞, que mede o erro máximo entre a função de destino e a função de aproximação. O teorema de limite de largura usa a norma L^1, que mede o erro médio em todo o domínio.
3. **Arquitetura de rede**: O teorema de Leshno não impõe restrições à largura da rede, permitindo que ela cresça conforme necessário. O teorema da largura limitada, por outro lado, restringe a rede a ter uma largura não superior a $n + 4$.
4. **Domínio de aproximação**: O teorema de Leshno exige que o domínio seja compacto, o que não é o caso do teorema da largura limitada.
5. **Função de ativação**: Ambos os teoremas consideram redes com funções de ativação ReLU, mas as implicações de seus resultados são diferentes devido a outras condições diferentes.

Prova (esboço). Para qualquer função integrável de Lebesgue $f : \mathbb{R}^n \to \mathbb{R}$ e qualquer precisão de aproximação predefinida $\epsilon > 0$, construímos explicitamente uma rede ReLU de largura $(n + 4)$ para que ela possa aproximar a função com a precisão determinada (dentro de um erro de L^1 de $\epsilon > 0$). A rede é uma concatenação de uma série de blocos. Cada bloco satisfaz as seguintes propriedades:

1. É uma rede ReLU com profundidade-$(4n + 1)$ e largura-$(n + 4)$.
2. Ele pode aproximar qualquer função integrável de Lebesgue que seja uniformemente zero fora de um cubo com comprimento δ com alta precisão;
3. Pode armazenar o resultado do bloco anterior, ou seja, a aproximação de outras funções integráveis de Lebesgue em cubos diferentes;
4. Ele pode somar sua aproximação atual e a memória das aproximações anteriores.

Etapa 1: Truncamento e decomposição. Dado f e ϵ, selecione N de modo que a integral de $|f|$ fora do cubo $[-N,N]^n$ seja menor que $\epsilon/2$. Defina f_1 e f_2 como as partes positivas e negativas de f truncadas no cubo $E = [-N,N]^n$. Em

seguida, decomponha f_1 e f_2 em funções simples que são somas de funções indicadoras de cubos $\mathrm{J}_{j,i}$ dentro de E. Essa decomposição pode ser feita de modo que o erro de L^1 seja menor que $\epsilon/4$:

O índice i assume dois valores, 1 e 2, correspondentes às partes positivas e negativas da função f truncada no cubo $\mathrm{E} = [-\mathrm{N},\mathrm{N}]^n$. Especificamente:

- $i = 1$ corresponde a $f_1(x)$, que representa a parte positiva de f dentro do cubo E.
- $i = 2$ corresponde a $f_2(x)$, que representa a parte negativa de f dentro do cubo E.

O índice j enumera o conjunto finito de cubos menores de dimensão n $\mathrm{J}_{j,i}$ que cobrem a região onde f_i é diferente de zero. Esses cubos são usados para construir funções simples que aproximam f_i por uma soma finita de funções indicadoras ponderadas. O processo é o seguinte:

1. Para cada $i \in \{1,2\}$, considere a região $\mathrm{V}_{\mathrm{E}}^i$ sob o gráfico de f_i sobre o cubo E. Essa região é mensurável, e nosso objetivo é cobri-la com um número finito de cubos $(n+1)$-dimensionais $\mathrm{J}_{j,i}$.
2. Usando as propriedades da medida de Lebesgue, podemos encontrar uma cobertura de $\mathrm{V}_{\mathrm{E}}^i$ por um número finito de cubos $(n+1)$-dimensionais $\mathrm{J}_{j,i}$ de modo que a medida da diferença simétrica entre $\mathrm{V}_{\mathrm{E}}^i$ e a união desses cubos seja pequena (menor que $\epsilon/8$ para cada i).
3. O índice j vai de 1 a n_i, onde n_i é o número de cubos $\mathrm{J}_{j,i}$ na cobertura para $\mathrm{V}_{\mathrm{E}}^i$.

Etapa 2: Construção de unidades únicas de ReLU (SRUs). Para cada cubo $\mathrm{J}_{j,i}$, construímos uma Single ReLU Unit (SRU) que aproxima sua função indicadora. A SRU é uma rede ReLU rasa com largura de $n+4$ e profundidade de 3. Ele aproxima a função indicadora do cubo de dimensão n manipulando o espaço de entrada por meio de uma série de transformações afins e ativações ReLU. A saída final é uma função que se assemelha muito à função indicadora dentro de uma margem de erro controlada.

Forma do SRU. Uma Single ReLU Unit (SRU) é uma sub-rede projetada para aproximar a função indicadora de um cubo de dimensão n. A SRU tem uma largura de $n+4$ neurônios e uma profundidade de 3 camadas. A arquitetura

da SRU é adaptada para manipular o espaço de entrada de forma a produzir um valor próximo a 1 dentro do cubo de destino e transições para o fora do cubo.

Aproximação da função indicadora. Considere um cubo n-dimensional $X = [a_1, b_1] \times \ldots \times [a_n, b_n]$ e sua função indicadora ϕ_X definida por ser igual a 1 em X e 0 fora dele. Para aproximar ϕ_X usando uma SRU, construímos uma função linear por partes que imita esse comportamento usando a função de ativação ReLU, definida como $\sigma(z) = \max\{0, z\}$.

Camada 1: Transformação de entrada. A primeira camada do SRU consiste em n neurônios que realizam uma transformação afim em cada coordenada de entrada x_i para preparar a ativação ReLU. Para cada dimensão i, temos dois neurônios:

1. Um neurônio calcula $(x_i - a_i)^+$, que é ativado quando x_i é maior que a_i.
2. Outro neurônio calcula $(b_i - x_i)^+$, que é ativado quando x_i é menor que b_i.

Camada 2: Intersecção de meios-espaços. A segunda camada pega as saídas da primeira camada e as combina para formar a interseção dos meios-espaços correspondentes ao cubo. Cada neurônio dessa camada recebe entradas de dois neurônios da camada anterior correspondentes à mesma dimensão. A saída de cada neurônio nessa camada é o mínimo das duas entradas, o que é obtido pela ativação ReLU:

$$\sigma\left(\sigma(x_i - a_i) - \sigma(b_i - x_i)\right) = \min\{\sigma(x_i - a_i), \sigma(b_i - x_i)\}.$$

Isso garante que o neurônio só seja ativado quando x_i estiver entre a_i e b_i.

Camada 3: Agregação e saída final. A terceira camada agrega os outputs da segunda camada para garantir que todas as dimensões estejam dentro do cubo. Isso é feito somando os resultados da segunda camada e subtraindo uma pequena margem $\delta > 0$ para garantir uma desigualdade estrita:

$$\sigma\left(\sum_{i=1}^{n} \sigma\left(\sigma(x_i - a_i) - \sigma(b_i - x_i)\right) - (n - 1 + \delta)\right).$$

A subtração de $(n - 1 + \delta)$ garante que o resultado seja 1 somente se todas as dimensões n estiverem dentro dos limites do cubo. A saída será 0 se alguma dimensão estiver fora do cubo.

Qualidade da aproximação. O SRU aproxima a função indicadora ϕ_X produzindo uma forma hipertrapezoidal dentro do cubo X. Ao ajustar a margem δ, podemos controlar a inclinação da transição de 1 para 0 nos limites do cubo. Quanto menor for o δ, mais próxima será a aproximação da verdadeira função indicadora. No entanto, existe uma compensação entre a inclinação da transição e a suavidade da aproximação.

Ao integrar a diferença absoluta entre a função indicadora ϕ_X e a saída da SRU em todo o espaço, podemos garantir que o erro seja menor que um limite especificado. Isso é obtido selecionando um δ adequado que equilibre a qualidade da aproximação com o nível de precisão desejado.

Etapa 3: Montagem da rede. A rede completa $\mathbf{A}$ é construída pela concatenação de várias SRUs (Single ReLU Units) para todos os cubos $J_{j,i}$ e camadas adicionais para combinar suas saídas. Cada SRU é responsável pela aproximação da função indicadora de um cubo $J_{j,i}$, e a rede deve somar essas aproximações para formar a aproximação geral de f. A rede $\mathbf{A}$ foi projetada para aproximar a parte positiva f_1 e a parte negativa f_2 da função f separadamente e, em seguida, combinar essas aproximações para produzir a aproximação final de f.

Concatenação de SRUs. Cada SRU é uma sub-rede de profundidade 3 que aproxima a função indicadora de um cubo $J_{j,i}$. A rede $\mathbf{A}$ concatena essas SRUs para todos os cubos $J_{j,i}$ que foram identificados na decomposição de f_1 e f_2. A saída de cada SRU é uma função $\varphi_{j,i}$ que aproxima a função indicadora correspondente $\phi_{j,i}$.

Armazenamento de aproximações de SRU. A rede $\mathbf{A}$ contém camadas que armazenam as saídas das SRUs. Essas camadas são projetadas para reter as informações das SRUs anteriores enquanto processam a SRU atual. Isso é obtido por meio de um conjunto de neurônios que atuam como "unidades de memória", que simplesmente passam os valores de uma camada para a seguinte

sem nenhuma modificação. Esses neurônios garantem que as aproximações de todas as SRUs estejam disponíveis para a soma final.

Soma de aproximações. Depois que as SRUs tiverem calculado suas respectivas aproximações, a rede precisará somar essas aproximações para formar a aproximação geral de f_1 e f_2. Isso é feito por camadas adicionais que combinam as saídas das SRUs. Especificamente, a rede tem neurônios dedicados a somar as saídas das SRUs correspondentes a f_1 e, separadamente, as saídas das SRUs correspondentes a f_2.

Para cada parte f_i, a soma é calculada da seguinte forma:

$$\sum_{j=1}^{n_i} b_{n+1,j,i} \varphi_{j,i}(x),$$

em que $b_{n+1,j,i}$ é o comprimento do $(n+1)$-ésimo lado do cubo $\mathrm{J}_{j,i}$, que atua como um peso para a aproximação $\varphi_{j,i}$.

Insuficiência de redes com profundidade limitada. O Teorema 1 implica que há um ponto de ruptura para o poder expressivo das redes ReLU à medida que a largura da rede varia em n, a dimensão de entrada. Não é difícil perceber que, se a largura for muito menor que n, o poder expressivo da rede deve ser muito fraco. Formalmente, temos os dois resultados a seguir:

Teorema (Falha da aproximação universal no espaço euclidiano). Para qualquer função integrável de Lebesgue $f : \mathbb{R}^n \to \mathbb{R}$ que satisfaça que $\{x : f(x) \neq 0\}$ seja um conjunto de medidas positivas na medida de Lebesgue e qualquer função $\mathrm{F}_{\mathbf{A}}$ representada por uma rede ReLU totalmente conectada $\mathbf{A}$ com largura de no máximo n:

$$\int_{\mathbb{R}^n} |f(x) - \mathrm{F}_{\mathbf{A}}(x)| \mathrm{d}x \quad \text{igual a qualquer um } +\infty \quad \text{ou} \quad \int_{\mathbb{R}^n} |f(x)| \mathrm{d}x.$$

Esse teorema diz que, mesmo quando a largura é igual a n, a capacidade de aproximação da rede ReLU ainda é fraca, pelo menos no espaço euclidiano $\mathbb{R}^n$. No entanto, se restringirmos a função em um conjunto limitado, ainda poderemos provar o seguinte teorema:

Teorema (Falha da aproximação universal em um domínio compacto). Para qualquer função contínua $f : [-1,1]^n \to \mathbb{R}$ que não seja constante em

nenhuma direção, existe um $\epsilon^* > 0$ universal de modo que, para qualquer função $\mathrm{F_A}$ representada por uma rede ReLU totalmente conectada com largura menor que n, a distância L^1 entre f e $\mathrm{F_A}$ é de pelo menos ϵ^*:

$$\int_{[-1,1]^n} |f(x) - \mathrm{F_A}(x)| \mathrm{d}x \geq \epsilon^*.$$

Ou seja, para qualquer função contínua $f : [-1,1]^n \to \mathbb{R}$, que varia em todas as direções, há um limite inferior ϵ^* na distância L^1 entre f e qualquer aproximação $\mathrm{F_A}$ que possa ser obtida por uma rede ReLU totalmente conectada com menos neurônios em sua(s) camada(s) oculta(s) do que a dimensionalidade do espaço de entrada.

Prova (esboço). Considere uma rede ReLU totalmente conectada $\mathbf{A}$ com entrada $\vec{x} = (x_1, x_2, \ldots, x_n)$ e valores do nó da primeira camada $y = (y_1, y_2, \ldots, y_m)$ onde $m < n$. A saída dos nós da primeira camada é dada por:

$$y_i = (b_i + \sum_{j=1}^{m} a_{ij} x_j)^+$$

para $i = 1, 2, \ldots, n$ e $j = 1, 2, \ldots, m$, sendo b_i e a_{ij} os parâmetros da rede.

Etapa 1: Existência de uma direção que não afeta. Como $m < n$, existe um vetor diferente de zero $\vec{x}_0 \in \mathbb{R}^n$ ortogonal à extensão dos vetores formados pelas transformações afins da primeira camada. (Ou seja, ele está no núcleo da parte linear da transformação afim) Ou seja, mover-se ao longo de $\vec{x}_0$ não afeta a saída da primeira camada e, portanto, $\mathrm{F_A}$ é constante ao longo de $\vec{x}_0$.

Etapa 2: Estabelecimento do limite inferior ϵ. Demonstramos que, para qualquer função contínua f e um vetor unitário fixo $\vec{x}_0$, existe um ϵ positivo de modo que, para todas as funções contínuas F constantes ao longo de $\vec{x}_0$, a distância L^1 entre f e F é de pelo menos ϵ:

Deixe $\mathrm{U}(a_0, r)$ denotar a bola aberta de raio r centrada em a_0 em $\mathbb{R}^n$

Como f não é constante em nenhuma direção e é contínua, existem dois pontos a_0 e b_0 ao longo de $\vec{x}_0$, um raio $r > 0$ e uma constante c de modo que, para todos os $a \in \mathrm{U}(a_0, r)$ e $b \in \mathrm{U}(b_0, r)$, temos $f(b) - f(a) > c$.

Temos $\mathrm{F}(b - b_0 + a_0) = \mathrm{F}(b)$, porque F é constante ao longo da direção de a para b. Usando a desigualdade triangular, obtemos:

$$|f(b) - \mathrm{F}(b)| + |f(b - b_0 + a_0) - \mathrm{F}(b - b_0 + a_0)| > f(b) - f(b - b_0 + a_0) > c,$$

já que $b - b_0 < r$ e, portanto, $b - b_0 + a_0$ em $\mathrm{U}(a_0, r)$. Integrando ambos os lados sobre a bola $\mathrm{U}(b_0, r)$ com relação a b, obtemos:

$$\int_{\mathrm{U}(b_0,r)} |f(b) - \mathrm{F}(b)| \, \mathrm{d}b + \int_{\mathrm{U}(b_0,r)} |f(b - b_0 + a_0) - \mathrm{F}(b - b_0 + a_0)| \, \mathrm{d}b$$

$$= \int_{\mathrm{U}(b_0,r)} |f(b) - \mathrm{F}(b)| \, \mathrm{d}b + \int_{\mathrm{U}(a_0,r)} |f(b) - \mathrm{F}(b)| \, \mathrm{d}b > 2\mathrm{V}c$$

em que V denota a medida de Lebesgue de $\mathrm{U}(b_0, r)$ (e $\mathrm{U}(a_0, r)$). Portanto, escolhendo $\epsilon = \mathrm{V}c$, temos $|\mathrm{F} - f|_{\mathrm{L}^1} > \epsilon$

Etapa 3: Argumento de continuidade e compacidade. Como ϵ é positivo e contínuo como função de $\vec{x}_0$ (devido à continuidade de f e F), e como o conjunto de todos os vetores unitários $\vec{x}_0$ forma um conjunto compacto (a superfície da esfera unitária em $\mathbb{R}^n$), há um limite inferior uniforme $\epsilon^* > 0$ em todos os vetores unitários $\vec{x}_0$.

6.6 Largura versus profundidade das redes neurais

Em Telgarsky (2016), Telgarsky mostrou que existem funções computáveis por uma rede ReLU profunda com um número polinomial de unidades que não podem ser aproximadas por nenhuma rede superficial (uma rede com apenas uma camada oculta), a menos que ela tenha exponencialmente muitas unidades. O resultado principal implica que, para qualquer número inteiro positivo k, existe uma função $f : \mathbb{R}^d \to \mathbb{R}$ que é calculada por uma rede neural com $2k^3+8$ camadas e $3k^3 + 12$ nós no total, de modo que nenhuma rede mais rasa com no máximo k camadas e menos de 2^k nós pode aproximar f dentro de uma distância L^1 de $\frac{1}{64}$ sobre o cubo unitário $[0,1]^d$.

Ideia de prova. A prova constrói uma função dente de serra específica usando uma rede profunda com ativações ReLU. Essa função oscila rapidamente e tem muitos "dentes", que são essencialmente peças lineares com inclinações alternadas. A construção é tal que cada camada adicional na rede dobra o número de peças lineares, levando a um aumento exponencial no número de oscilações com relação à profundidade da rede.

Para aproximar essa função com uma rede rasa, seria necessária uma peça linear separada para cada dente da função dente de serra. Como o número de dentes é exponencial na profundidade da rede profunda, uma rede rasa exigiria um número exponencial de unidades para atingir um nível semelhante de aproximação.

A prova aproveita a estrutura de composição das redes profundas, em que cada camada pode se basear nas transformações das camadas anteriores, permitindo que funções complexas sejam representadas de forma mais compacta do que em redes rasas. Esse resultado destaca os benefícios da profundidade nas redes neurais, mostrando que a profundidade pode levar a um aumento significativo no poder de representação.

6.7 O método do Máximo Declive

Em vez de um limiar, para facilitar a notação, suponhamos que exista uma entrada adicional igual a 1:

Convenção. Para facilitar a notação, suponhamos de agora em diante que cada camada tenha como entrada um fator igual a 1, isto é,

$$\mathrm{X} = \{1\} \times \mathbb{R} \times \cdots \times \mathbb{R}$$

cujas ênuplas numeremos por $(x_0, x_1, \ldots)$. Desta maneira, o limiar b corresponde ao peso w_0, e não necessita mais ser considerado separadamente.

Medida de Erro. Com a saída não discreta, gostaríamos de tomar em conta o tamanho da diferença $h(x) - y(x)$ em cada ponto x; usaremos a diferença quadrática a este fim, isto é

$$\mathrm{E_D}(h) = [h(x_1) - y(x_1)]^2 + \cdots + [h(x_\mathrm{N}) - y(x_\mathrm{N})]^2.$$

O método do Máximo Declive. O algoritmo usa o método do *Máximo Declive* para minimizar o erro quadrático. Ele busca um mínimo local por um ponto no qual a derivada, o *gradiente*, da medida de erro $\mathrm{E_D}$ como função em w é zero, isto é, todas as derivadas parciais ao longo dos pesos dos neurónios esvaem-se.

Mais exatamente, recordemo-nos de que cada camada l tem múltiplos neurónios h_i^l, e cada neurónio atribui a cada entrada j (= a saída do neurónio j da camada precedente) um peso w_{ij}^l. Para minimizar $\mathrm{E_D}(w)$ como função dos pesos $w = (w_{ij}^l)$, isto é, para calcular o gradiente $\nabla \mathrm{E_D}(w) = \partial \mathrm{E_D}(w)/\partial w_{ij}^l$, dado pelas derivadas parciais, e aproximar o valor zero:

1. Inicializa o vetor de pesos $w(0)$ (e define a margem de traslado $\eta > 0$). (†)
2. Para $t = 0,1,\ldots$
    1. Calcula o gradiente $g_t := \nabla \mathrm{E_D}(w(t))$ em $w(t)$.
    2. Põe a direção $v_t := -g_t/\|g_t\|$.
    3. Atualiza o vetor de pesos $w(t+1) = w(t) + \eta \cdot v_t$.
    4. Decide se continuar (com $w(t+1)$) ou terminar a iteração. (‡)
3. Devolve o último $w(t)$ (isto é, onde a iteração parou).

Quanto a (†), às escolhas iniciais de η e $w(0)$,

- uma escolha comum para η é $\eta = 0{,}1$, e
- uma escolha comum para $w(0)$ é 0, ou, para evitar o risco de uma má escolha específica, escolher cada peso $w_1, \ldots, w_d$ independentemente por uma distribuição normal.

Quanto a (‡), um bom critério para a decisão sobre a terminação da iteração, ele é dado pela satisfação de

- um limiar mínimo do erro $\mathrm{E_D}$ (isto é, terminamos quando $\mathrm{E_D}$ for menor do que um certo limiar, ou seja, quando o erro é suficientemente pequeno), e
- um limiar mínimo da variação do erro $\Delta \mathrm{E_D}$ (isto é, terminamos quando $\Delta \mathrm{E_D}$ for menor do que um certo limiar, ou seja, as variações do erro são suficientemente pequenas),

enquanto $t = 0,1,\ldots$ não chegou a

- um número máximo T de iterações (como um critério de segurança caso os dois critérios acima não se apliquem).

6.8 Vetorização

Para acelerar a computação, as saídas da função afim (de múltiplas saídas) de cada camada é calculada por matrizes:

Saída. Em vez de calcular separadamente, para cada neurónio de uma camada,

$$z_1 = w_1^{\mathrm{T}} x + b_1, z_2 = w_2^{\mathrm{T}} x + b_2, \ldots$$

e

$$a_1 = g(z_1), a_2 = g(z_2), \ldots,$$

é mais eficiente calcular

$$z = \mathrm{W}^{\mathrm{T}} x + b$$

e

$$a = (g(z_1), g(z_2), \ldots)$$

onde $z = (z_1, z_2, \ldots)$, $b = (b_1, b_2, \ldots)$ e W é a matriz com as linhas w_1, w_2, ...

Amostra. Para vetorizar, simultanear, a computação sobre múltiplas entradas, isto é, sobre todos os exemplos x_1, x_2, ... da amostra, pomos

$$\mathrm{X} = (x_1, x_2, \ldots) \quad \text{e} \quad b = (b, b, \ldots)$$

e calculamos

$$\mathrm{Z} = (z_1, z_2, \ldots) = \mathrm{WX} + b$$

6.9 Auto-diferenciação inversa pela Regra da Cadeia

Dada uma rede neural de propagação à frente cuja medida de erro é o erro quadrático. Dada uma amostra x_1, ..., x_{N} com valores $y(x_1)$, ..., $y(x_{\mathrm{N}})$, (a metade d') o erro quadrático é

$$\mathrm{E} = (\|h(x_1) - y(x_1)\|^2 + \cdots + \|h(x_{\mathrm{N}}) - y(x_{\mathrm{N}})\|^2)/2\mathrm{N}.$$

(O fator de normalização 1/2 simplificará a fórmula da derivada a seguir.) A rede é parametrizada pelos pesos $w = (w_{ij}^l)$ dos neurónios. Queremos minimizar E em dependência de w. Para minimizar E em dependência de w, calculemos as derivada parciais

$$\partial \mathrm{E}/\partial w_{ij}^l$$

para, idealmente, encontrar um zero comum entre elas. Como

$$\mathrm{E} = (\|h(x_1) - y(x_1)\|^2 + \cdots + \|h(x_\mathrm{N}) - y(x_\mathrm{N})\|^2)/2\mathrm{N}.$$

e

$$\|h(x) - y(x)\|^2 = (h_1(x) - y_1(x))^2 + \cdots + (h_d(x) - y_d(x))^2,$$

podemos primeiro calcular $\partial \mathrm{E}/\partial w_{ij}^l$ para cada exemplo $x = x_1, \ldots, x_\mathrm{N}$ e cada neurónio $i = 1, \ldots, d$ da saída e depois somar os resultados. Logo, para simplificar as fórmulas, suponhamos um *único* exemplo x e um *único* neurónio de saída, isto é, $\mathrm{N} = 1$ e $d = 1$ tais que

$$\mathrm{E} = \frac{1}{2}(h(x) - y(x))^2.$$

Pela *Regra da Cadeia*,

$$\partial \mathrm{E}/\partial w_{ij}^l = \partial \mathrm{E}/\partial z_j^l \cdot \partial z_j^l/\partial w_{ij}^l \quad (*)$$

onde z_j^l é a função afim do neurónio j da camada l cujo valor em x é computado por

$$z_j^l(x) = w_{0j}^l a_0^{l-1}(x) + w_{1j}^l a_1^{l-1}(x) + \cdots$$

Para o neurónio i na camada l, denotemos

- a sua função afim por $z_\mathrm{I}^\mathrm{L} := \mathrm{F}_i^l$,
- a sua função não-afim por g,
- a sua função composta por $a_i^l := g \circ z_i^l$, e
- o fator ao lado esquerdo do produto em $(*)$, o *erro da saída*, por

$$\delta_j^l := \partial \mathrm{E}/\partial z_j^l.$$

Para calcular o fator ao lado direito do produto em $(*)$, observemos que pela Regra do Produto

$$\partial z_j^l/\partial w_{ij}^l = \partial(w_{0j}^l a_0^{l-1} + w_{1j}^l a_1^{l-1} + \cdots)/\partial w_{ij}^l = a_i^{l-1}.$$

Concluímos

$$\partial \mathrm{E}/\partial w_{ij}^l = \delta_j^l \cdot a_i^{l-1}. \quad (\dagger)$$

A Derivada do Erro da Última Camada. Calculemos o fator ao lado esquerdo do produto em $(*)$, o erro δ^{L} da saída, do *único* neurónio da *última* camada L. Pela Regra da Cadeia,

$$\delta_j^{\mathrm{L}} = \partial \mathrm{E}/\partial z^{\mathrm{L}} = \frac{1}{2}\partial (a^{\mathrm{L}} - y)^2/\partial z^{\mathrm{L}} = (g \circ z^{\mathrm{L}} - y) \cdot g' \circ z^{\mathrm{L}} = (h - y) \cdot g' \circ z^{\mathrm{L}}.$$

Concluímos

$$\partial \mathrm{E}/\partial w_i^{\mathrm{L}} = \delta_j^{\mathrm{L}} a_i^{\mathrm{L}-1} = (h - y) g' \circ z_j^{\mathrm{L}} \cdot a_i^{\mathrm{L}-1}.$$

Expressemos recursivamente as derivadas parciais das camadas anteriores $l = \mathrm{L} - 1, \ldots$ pelas posteriores; comecemos por expressar as de $l = \mathrm{L} - 1$ pelas de $l = \mathrm{L}$:

A Derivada do Erro da Penúltima Camada. Para calcular $\partial \mathrm{E}/\partial w_{ij}^{\mathrm{L}-1}$ do neurónio i da *penúltima* camada $\mathrm{L}-1$, basta por $(\dagger)$ calcular $\delta_i^{\mathrm{L}-1}$. Para calcular o erro $\delta_i^{\mathrm{L}-1}$ do neurónio i da *penúltima* camada $\mathrm{L} - 1$, observemos que z^{L} depende de $a_1^{\mathrm{L}-1}$, $a_2^{\mathrm{L}-1}, \ldots$ Explicitamente,

$$z^{\mathrm{L}} = w_1^{\mathrm{L}} a_1^{\mathrm{L}-1} + w_2^{\mathrm{L}} a_2^{\mathrm{L}-1} + \cdots = w_1^{\mathrm{L}} g \circ z_1^{\mathrm{L}-1} + w_2^{\mathrm{L}} g \circ z_2^{\mathrm{L}-1} + \cdots . \quad (**)$$

Pela Regra da Cadeia

$$\delta_i^{\mathrm{L}-1} = \partial \mathrm{E}/\partial z_i^{\mathrm{L}-1} = \partial \mathrm{E}/\partial z^{\mathrm{L}} \cdot \partial z^{\mathrm{L}}/\partial z_i^{\mathrm{L}-1}.$$

Os dois fatores, por definição respetivamente derivação parcial de $(**)$, são

$$\partial \mathrm{E}/\partial z_j^{\mathrm{L}} = \delta^{\mathrm{L}} \quad \text{e} \quad \partial z^{\mathrm{L}}/\partial z_i^{\mathrm{L}-1} = w_i^{\mathrm{L}} g' \circ z_i^{\mathrm{L}-1};$$

logo

$$\delta_i^{\mathrm{L}-1} = g' \circ z_i^{\mathrm{L}-1} w_i^{\mathrm{L}} \cdot \delta^{\mathrm{L}}.$$

A Derivada do Erro da Antepenúltima Camada. Para calcular $\partial \mathrm{E}/\partial w_{ij}^{\mathrm{L}-2}$ do neurónio i da *antepenúltima* camada $\mathrm{L}-2$, basta por $(\dagger)$ calcular $\delta_i^{\mathrm{L}-2}$. Para calcular o erro $\delta_i^{\mathrm{L}-2}$ do neurónio i da *antepenúltima* camada $\mathrm{L} - 2$, observemos outra vez que $z^{\mathrm{L}-1}$ depende de $a_1^{\mathrm{L}-2}$, $a_2^{\mathrm{L}-2}, \ldots$ Explicitamente,

$$z_j^{\mathrm{L}-1} = w_{1j}^{\mathrm{L}-1} a_1^{\mathrm{L}-2} + w_{2j}^{\mathrm{L}-1} a_2^{\mathrm{L}-2} + \cdots = w_{1j}^{\mathrm{L}-1} g(z_1^{\mathrm{L}-2}) + w_{2j}^{\mathrm{L}-1} g(z_2^{\mathrm{L}-2}) + \cdots . \quad (***)$$

Pela Regra da Cadeia

$$\delta_i^{\mathrm{L}-2} = \partial \mathrm{E}/\partial z_i^{\mathrm{L}-2} = \partial \mathrm{E}/\partial z^{\mathrm{L}-1} \cdot \partial z^{\mathrm{L}-1}/\partial z_i^{\mathrm{L}-2} = \sum_j \partial \mathrm{E}/\partial z_j^{\mathrm{L}-1} \cdot \partial z_j^{\mathrm{L}-1}/\partial z_i^{\mathrm{L}-2}$$

onde a última igualdade expande o produto matricial do penúltimo termo. Os dois fatores de cada termo da soma, por definição respetivamente derivação parcial de $(***)$, são

$$\partial \mathrm{E}/\partial z_j^{\mathrm{L}-1} = \delta_j^{\mathrm{L}-1} \quad \text{e} \quad \partial z_j^{\mathrm{L}-1}/\partial z_i^{\mathrm{L}-2} = w_{ij}^{\mathrm{L}-1} g' \circ z_i^{\mathrm{L}-2};$$

logo

$$\delta_i^{\mathrm{L}-2} = \sum_j \delta_j^{\mathrm{L}-1} \cdot w_{ij}^{\mathrm{L}-1} g' \circ z_i^{\mathrm{L}-2} = g' \circ z_i^{\mathrm{L}-2} \sum_j w_{ij}^{\mathrm{L}-1} \cdot \delta_j^{\mathrm{L}-1}.$$

Por indução, vale esta equação para cada neurónio j em qualquer camada $l = \mathrm{L} - 3, \ldots$ no lugar de $\mathrm{L} - 2$.

6.10 Algoritmo Propagação para Trás

O algoritmo da *Propagação para Trás* (= *Backpropagation*) é um algoritmo para calcular eficientemente o gradiente $\nabla \mathrm{E}$ pela Regra da Cadeia (de funções deriváveis). Em geral, fora do contexto de uma rede neural, este algoritmo é conhecido como *auto-diferenciação inversa.*

Seja n_i^l a função do neurónio i da camada l. Lembremo-nos de que $n = g \circ h$ para uma função de ativação g e uma função afim h. O algoritmo da Propagação para Trás,

1. Calcula para cada entrada $x = x_n$ os valores dos neurónios para a frente, isto é, dada a entrada x,
    1. calcula $h(x)$ e $n(x) = g(h(x))$ para todos os neurónios $n = n_{ij}^l = g \circ h_{ij}^l$ das camadas ocultas $1, 2, \ldots, m-1$ e
    2. calcula $h(x)$ e $n(x) = g_s(h(x))$ para todos os neurónios da última camada m.
2. Calcula para cada entrada $x = x_n$ as derivadas $\partial \mathrm{E}/\partial w_{ij}^l$ dos neurónios para trás, isto é, calcula
    1. o erro δ^m da última camada,
    2. os erros δ^{m-1}, δ^{m-2}, $\ldots$ das camadas precedentes, e

3. as derivadas parciais $\partial \mathrm{E}/\partial w_{ij}^l = \delta_j^l n_i^{l-1}$

3. Calcula a derivada do erro da amostra inteira

$$\partial \mathrm{E}/\partial w_{ij}^l = \partial \mathrm{E}/\partial w_{ij}^l(x_1) + \cdots + \partial \mathrm{E}/\partial w_{ij}^l(x_\mathrm{N})$$

4. Atualiza os pesos por

$$w_{ij}^l(t+1) = w_{ij}^l(t+1) - \alpha \partial \mathrm{E}/\partial w_{ij}^l$$

Observação. A computação das derivadas das funções de ativação é usualmente fácil, por exemplo:

- Se $g = \sigma = 1/(1+\mathrm{e}^{-s})$, então $g' = \sigma(1-\sigma)$, e
- se $g_s = \mathrm{id}$, então $g_s' = 1$.

7 Redes Recorrentes

Redes recorrentes são redes neurais artificiais que contêm ciclos (direcionados). Fazem sucesso

- no processamento de línguas naturais (= NLP = Natural Language Processing), e
- no aprendizado com reforço como no aprendizado dos jogos go e xadrez (por AlphaGo e AlphaZero).

Redes recorrentes de Elman. As matrizes ponderadas dos neurónios são

$$\mathrm{W}(a), \mathrm{W}(x), \mathrm{W}(y), \text{ e } \mathrm{W} = [\mathrm{W}(a)|\mathrm{W}(x)]$$

Pomos, no passo $t+1$,

$$a(t+1) = g(a)(\mathrm{W}[a(t)|x(t)] + b(a)) \quad \text{e} \quad y(t+1) = g(y)(\mathrm{W}(y)a(t) + b(y))$$

onde

- $g(a)$ é, por exemplo, a `ReLU` ou `tanh`;
- $g(y)$ é, por exemplo, a função logística.

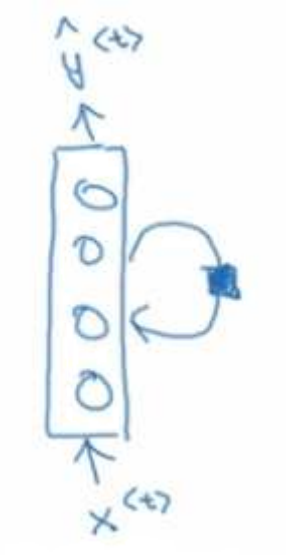


Figura 25: Uma rede recorrente

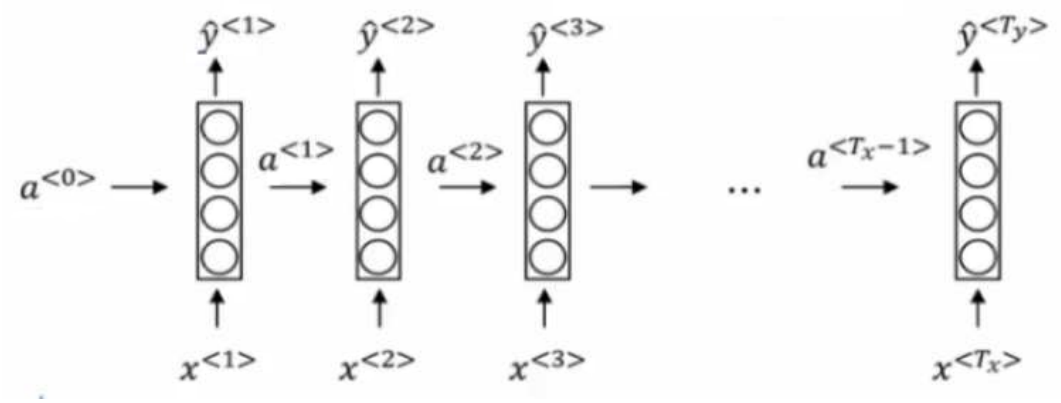


Figura 26: Uma rede recorrente desdobrada

Comparação com Redes de retropropagação. Uma rede recorrente não precisa de ter uma entrada e saída de comprimento fixo, mas ambas podem variar. Esta restrição poderia ser contornada por um tamanho máximo suficiente (da entrada e saída) e preenchimento.

Tipos.

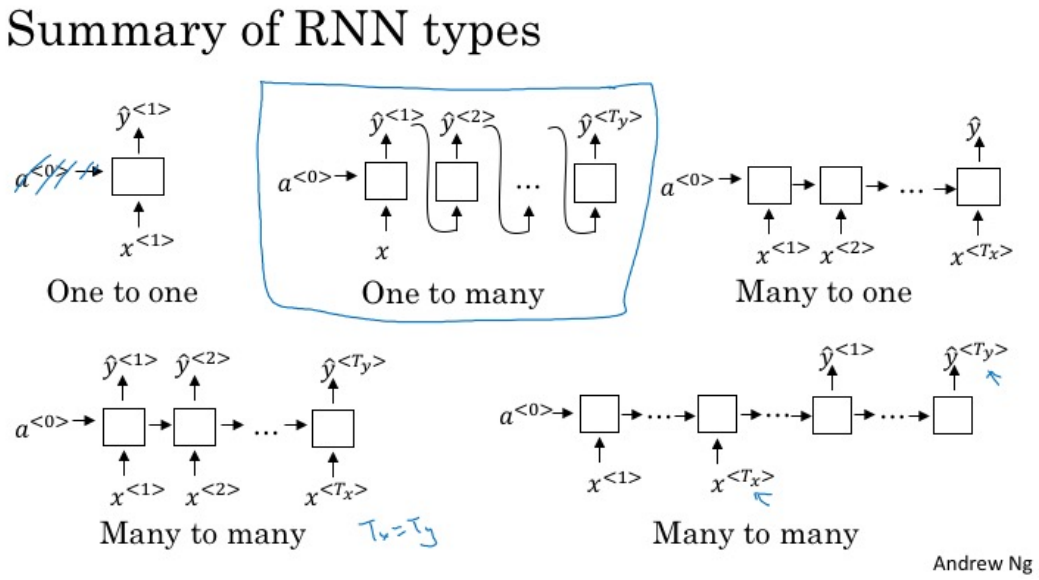


Figura 27: Tipos diferentes

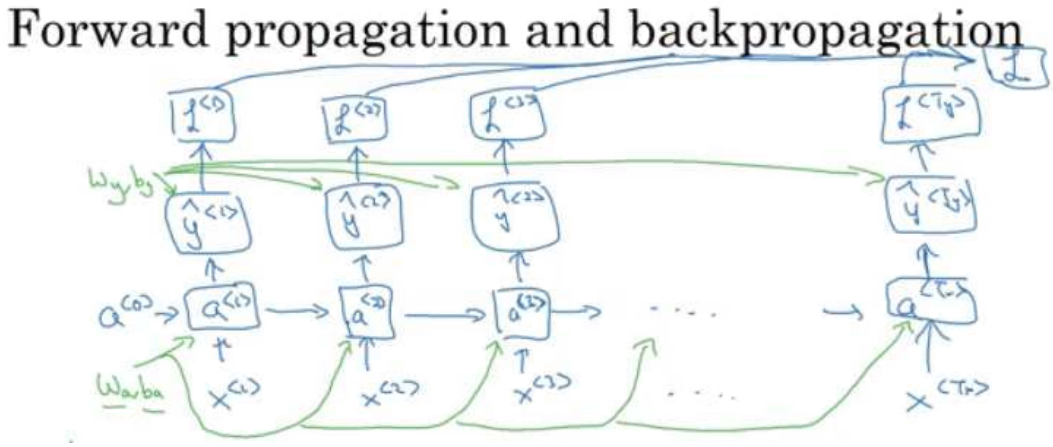


Figura 28: Retropropagação numa rede recorrente desdobrada

Retropropagação através do tempo. –> –> –> –> –> –> –> –> –> –>

7.1 Mapas Auto-Organizáveis

Um *mapa auto-organizável* é usado no aprendizado não-supervisionado; isto é, para encontrar um espaço de rótulos Y e aproximar a função divina $y\colon \mathrm{X} \to \mathrm{Y}$ que rotula as entradas sem erros.

São dados

- o espaço de entradas $\mathrm{X} = \mathbb{R}^d$, e
- uma amostra finita D incluso em X.

Como método, usamos como

- espaço de hipóteses H todos os mapas auto-organizáveis: Um *mapa auto-organizável* é
  - um número finito de neurónios enumerados por $i = 1, \ldots, n$; cada neurónio é
    * um vetor de pesos w em $\mathbb{R}^d$, e
    * uma posição p em $\mathbb{R}^m$, e
  - uma *métrica* (frequentemente chamada de *topologia*) $h\colon \mathbb{R}^m \times \mathbb{R}^m \to [0, \infty[$ que mede a distância entre duas posições p e q; é a composição de uma função de densidade probabilista unimodal, isto é, que tem um único máximo, com a distância euclidiana; usualmente,

    $$h(p,q) := \exp(-\|p - q\|^2/2\sigma).$$

  - um *quociente de aprendizado* η em $[0,1]$.

  A métrica e o quociente de aprendizado frequentemente são parametrizados por um número de iteração tal que $\nu(t) \to 0$ e $h \to h_0$ (na norma de supremo) para $t \to \infty$; usualmente

  - $\eta(t) = \eta^t$, e
  - $\sigma(t) = \sigma^t$

  para $\sigma < 1$ e $\sigma < 1$ positivos.

- como algoritmo, atualizemos para cada exemplo $v(t)$ em D o vetor dos pesos $w(t)$ de cada neurónio por

$$w(t+1) := w(t) + \Delta w(t)$$

onde

$$\Delta w(t) := \eta(t) h(i, i_0; t)(v(t) - w)$$

onde i é o (índice do) neurónio cujo vetor de pesos é w e i_0 é o (índice do) neurónio cujo vetor de pesos minimize

$$\|w - v(t)\|$$

entre os vetores de pesos w de todos os neurónios.

Exemplo. Sejam $v(1), \ldots, v(4)$ dados por

$$[0.8, 0.7, 0.4], [0.6, 0.9, 0.9], [0.3, 0.4, 0.1], [0.1, 0.1, 0.3]$$

e w_1 e w_2 por

$$[0.5, 0.6, 0.8] \quad \text{e} \quad [0.4, 0.2, 0.5].$$

Seja $\eta(t) = 0{,}5$ e $h(i,i_0) = 1$ se, e tão-somente se, $i = i_0$. Calculamos $d_1(t) = \|v(1) - w_1\|$ e $d_2(t) = \|v(1) - w_2\|$

$d_1(t) = (0.5-0.8)^2+(0.6-0.7)^2+(0.8-0.4)^2 = 0.26$ e $d_2(t) = (0.4-0.8)^2+(0.2-0.7)^2+(0.5-0.4)^2$

Como $d_1 < d_2$, atualizamos w_1 e obtemos $w_1 = [0.65, 0.65, 0.6]$. Calculemos novamente

$d_1 = (0.65-0.6)^2+(0.65-0.9)^2+(0.60-0.9)^2 = 0.155$ e $d_2 = (0.4-0.6)^2+(0.2-0.9)^2+(0.5-0.9)^2 =$

Como $d_1 < d_2$, atualizamos w_1 e obtemos $w_1 = [0.625, 0.775, 0.75]$. Calculemos novamente

$d_1 = (0.625-0.3)^2+(0.775-0.4)^2+(0.750-0.1)^2 = 0.67$ e $d_2 = (0.4-0.3)^2+(0.2-0.4)^2+(0.5-0.1)^2$

Como $d_2 < d_1$, atualizamos $w_2 = [0.35, 0.3, 0.3]$; calculamos

$d_1 = (0.625-0.1)^2+(0.775-0.1)^2+(0.750-0.3)^2 = 0.93$ e $d_2 = (0.350-0.1)^2+(0.300-0.1)^2+(0.30$

Como $d_2 < d_1$, atualizamos $w_2 = [0.225, 0.2, 0.3]$.

Referências Bibliográficas

Abu-Mostafa, Yaser S, Malik Magdon-Ismail, e Hsuan-Tien Lin. 2012. *Learning from data*. Vol. 4. AMLBook New York, NY, USA.

Telgarsky, Matus. 2016. «Benefits of depth in neural networks». Em *Conference on learning theory*, 1517–39. PMLR. https://arxiv.org/abs/1602.04485.