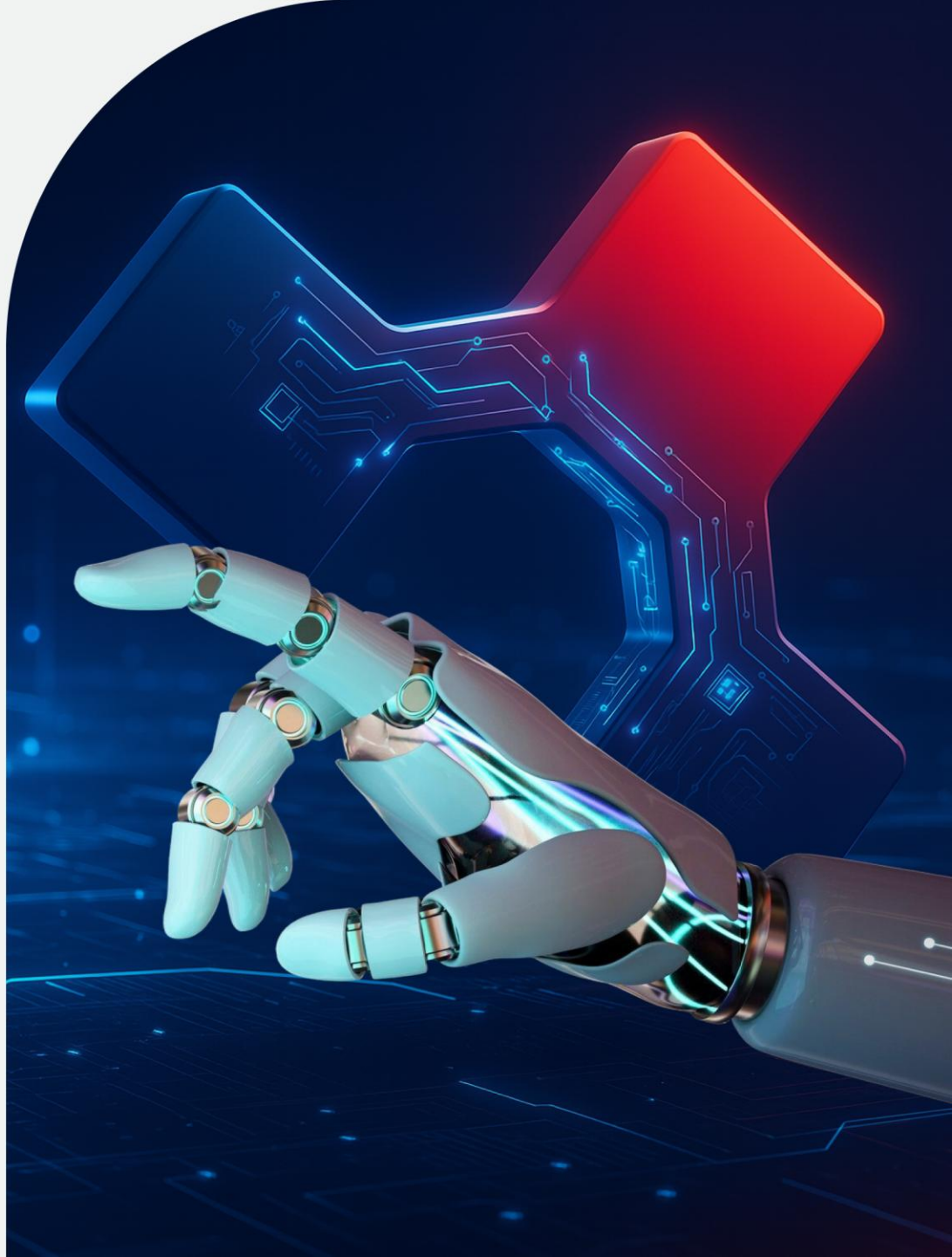
NCIA

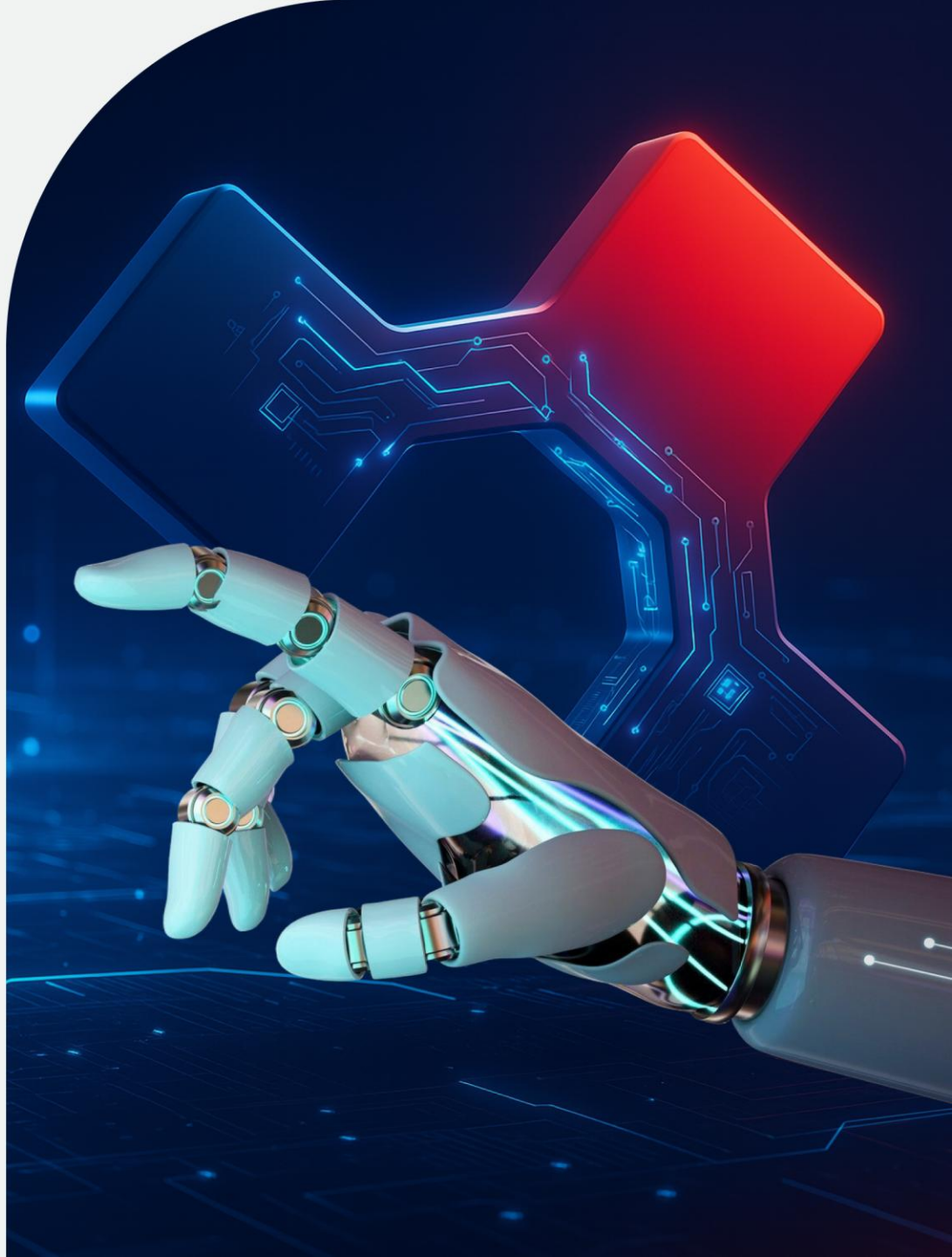
Núcleo de Capacitação em Inteligência Artificial



NCIA | FOXCONN FPF tech

Training Models

Linear Regression; The Normal Equation; Computational Complexity; **Gradient Descent**; Batch Gradient Descent; Stochastic Gradient Descent; Mini-Batch Gradient Descent; Polynomial Regression; Learning Curves;



NCIA | FOXCONN FPF tech

Regressão Linear

Dimensionalidade dos dados

Valor previsto

Peso

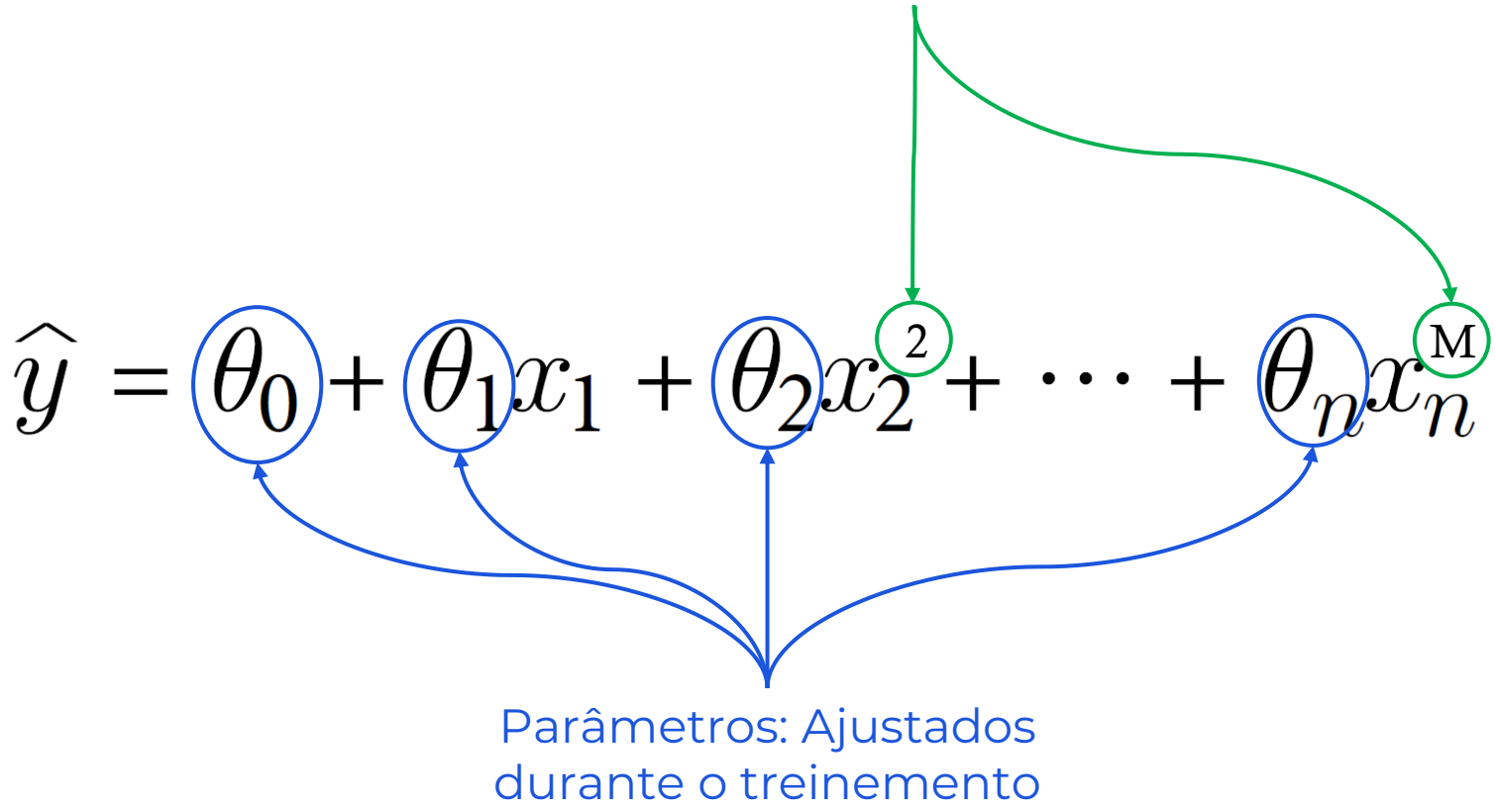
$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

Intercepto (Viés – Bias)

Features de entrada

Treinamento = Encontrar θ que minimiza o erro (Cost Function)
O Erro é medido pela função de custo (RMSE)

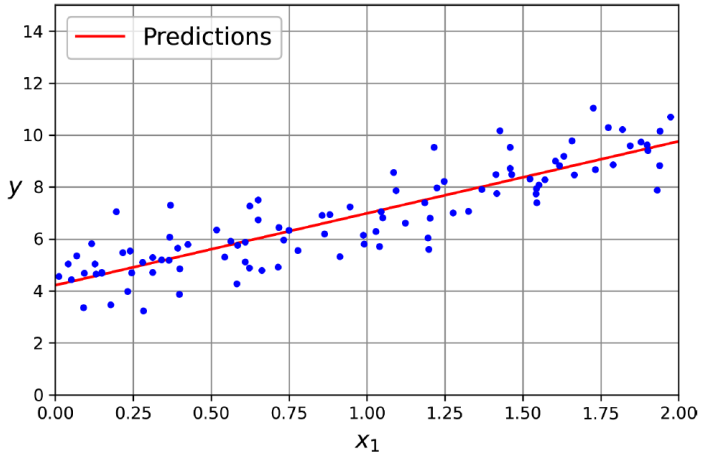
Hiperarâmetros: Definidos manualmente e constantes durante todo o treinamento



Parâmetros: Ajustados durante o treinemento

1. Equação Normal

$$\hat{\theta} = (X^T X)^{-1} \quad X^T \quad y$$

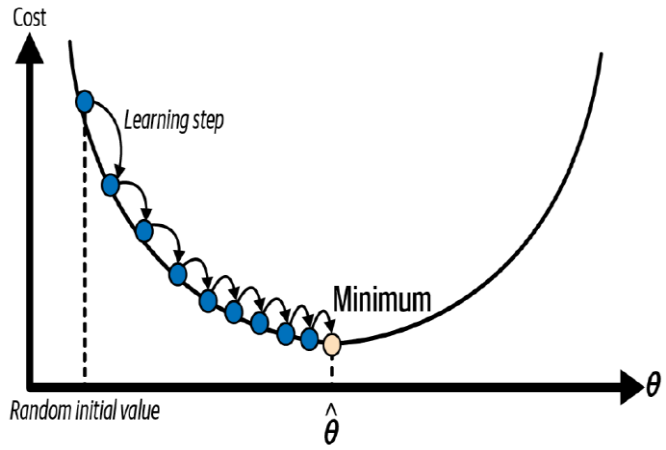


Vantagens: Resultado exato, simples para datasets pequenos
Desvantagens: Custo computacional $O(n^3)$, inviável para muitos atributos

Alternativa: pseudoinversa (SVD) → mais eficiente e robusta

2. Gradiente Descendente (GD)

$$\theta := \theta - \eta \cdot \nabla_\theta J(\theta)$$

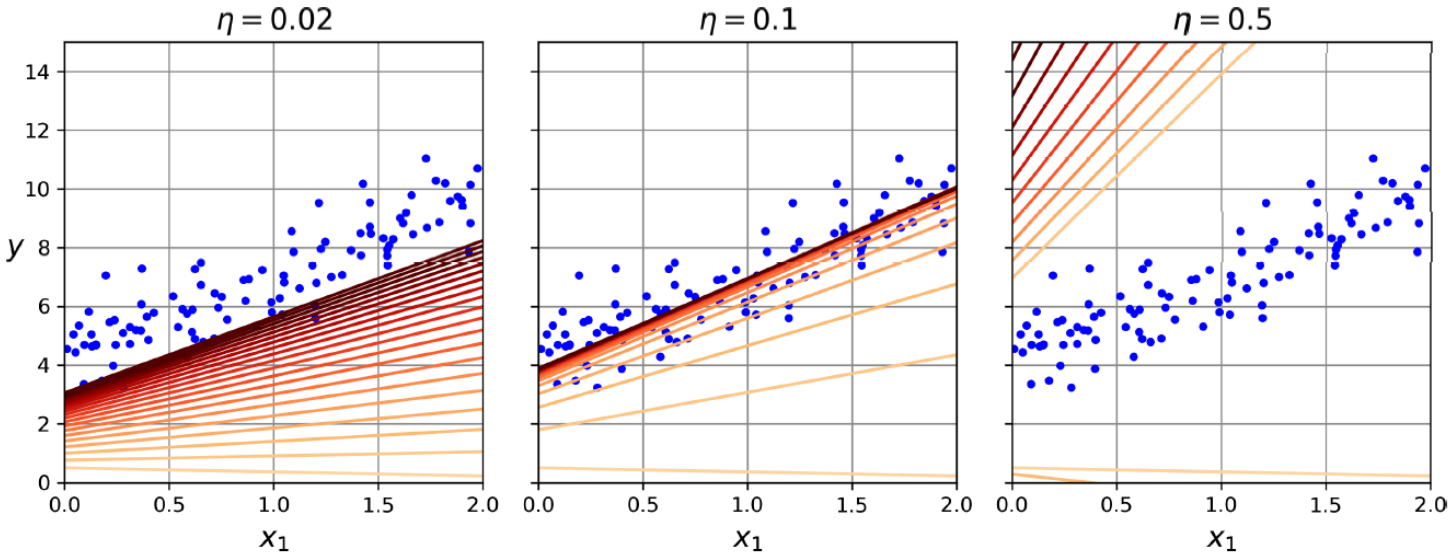


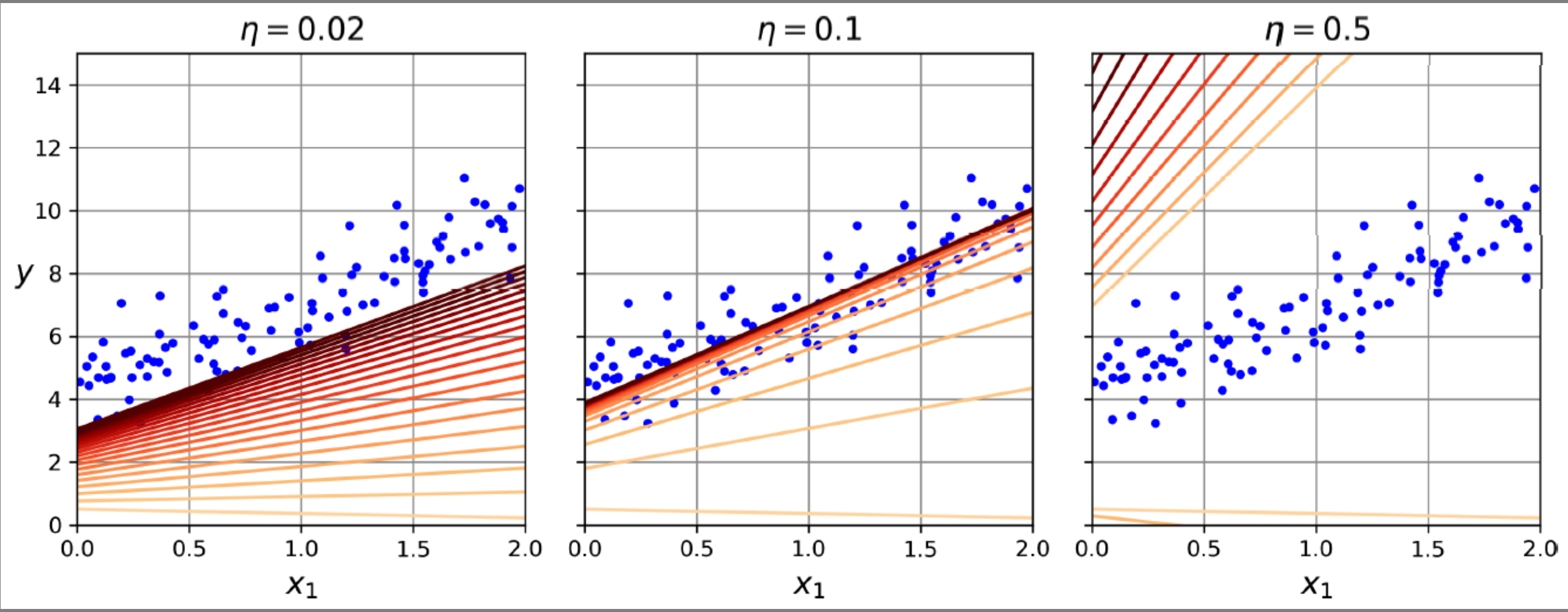
O Gradiente Descendente é uma técnica de otimização que ajusta os parâmetros do modelo para minimizar a função de custo. Ele começa de um ponto inicial (geralmente aleatório), calcula o gradiente (a direção de maior subida da função de custo) e dá passos na direção contrária, definidos pelo *learning rate*.

Se o *learning rate* for muito alto, o algoritmo pode divergir.
Se for muito baixo, converge muito devagar

2.1 Batch Gradiente Descendente (BGD)

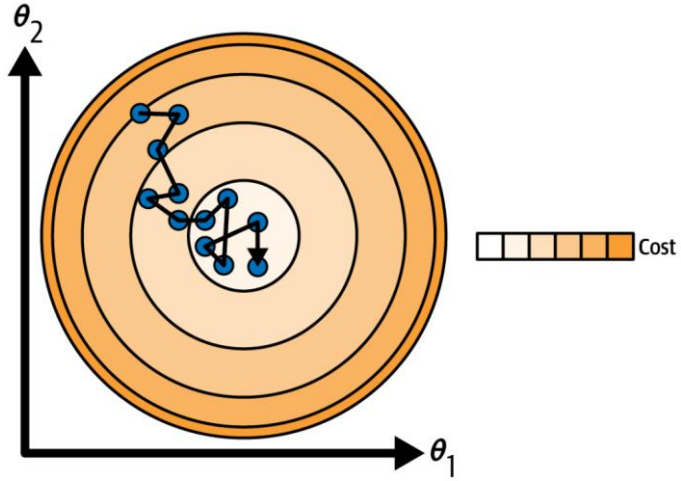
O Batch Gradient Descent é uma técnica de otimização que, a cada iteração, utiliza todo o conjunto de dados de treino para calcular o gradiente da função de custo, garantindo passos estáveis e previsíveis em direção ao mínimo global. No entanto, esse processo pode ser muito lento e custoso em bases de dados grandes, já que cada atualização dos parâmetros exige percorrer todos os exemplos de uma vez.




$\eta = 0.02$
$\eta = 0.1$
$\eta = 0.5$
y
x_1
x_1
x_1
0
2
4
6
8
10
12
14
0.0
0.5
1.0
1.5
2.0

2.2 Gradiente Descendente Estocástico (SGD)

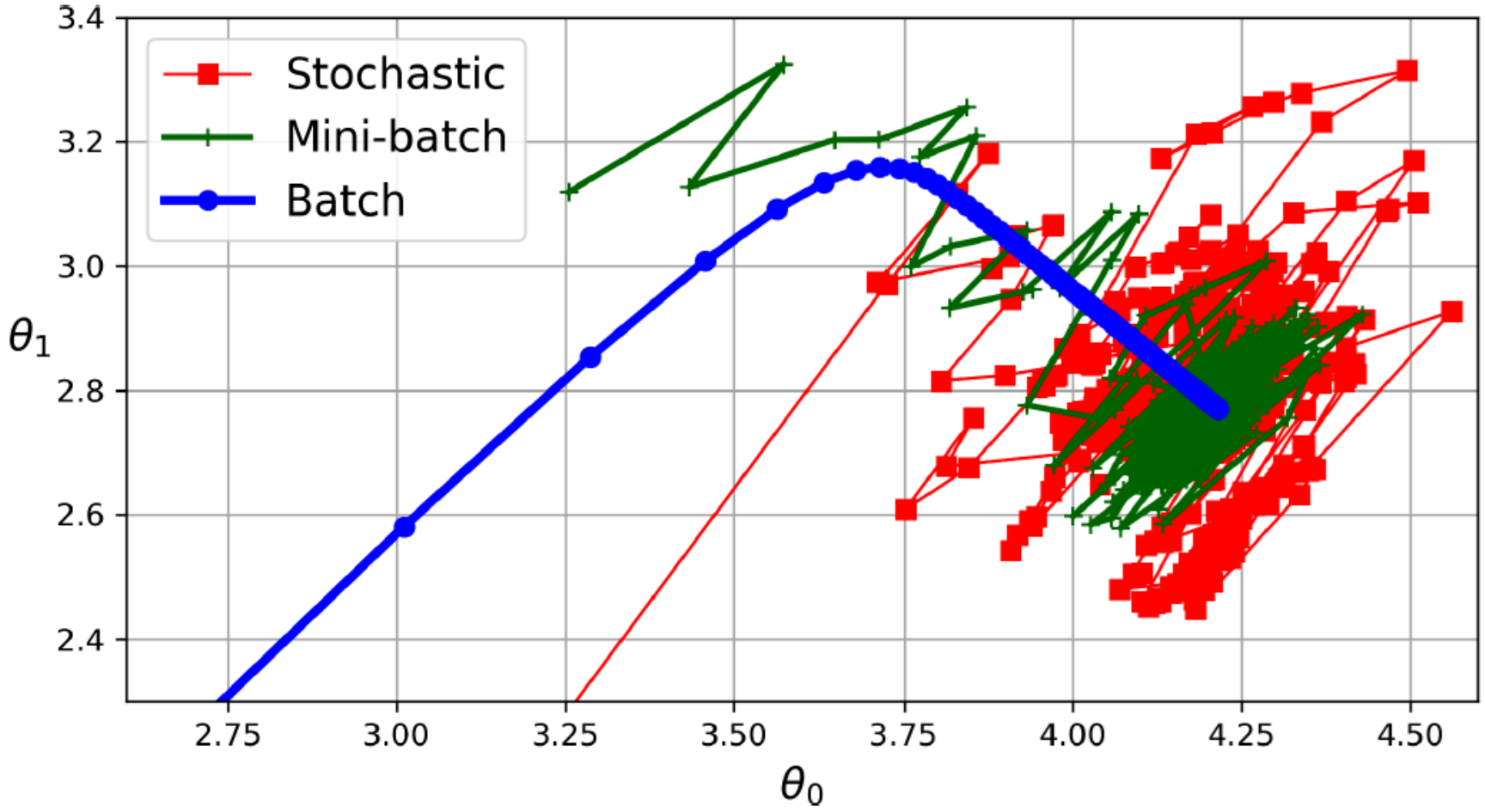
Atualiza os parâmetros do modelo usando apenas um exemplo aleatório do conjunto de dados por vez. Isso torna cada passo muito mais rápido e viável em bases grandes, além de ajudar a escapar de mínimos locais. Porém, o caminho até o mínimo é irregular e os parâmetros nunca se estabilizam completamente, razão pela qual costuma-se reduzir a taxa de aprendizado ao longo do tempo para permitir que o algoritmo se aproxime do mínimo global.



2.2 Mini-batch Gradiente Descendente (MBGD)

É uma variação do gradiente descendente que combina as ideias do Batch e do Stochastic. Em vez de usar todo o conjunto de dados, como no Batch, ou apenas um exemplo por vez, como no Stochastic, ele utiliza pequenos grupos de exemplos chamados mini-batches. A cada iteração o algoritmo calcula o gradiente com base nesse lote e atualiza os parâmetros. Assim, consegue-se um bom equilíbrio: o treino é mais rápido do que no Batch e menos instável do que no Stochastic.

Enquanto o Batch GD percorre todo o dataset em cada passo e garante uma descida estável porém muito lenta, o Stochastic GD trabalha com apenas um exemplo por vez, o que o torna rápido, mas bastante irregular. O Mini-Batch GD fica no meio do caminho, utilizando lotes pequenos que permitem atualizações frequentes e menos ruidosas. Dessa forma, ele consegue aproveitar tanto a velocidade quanto uma maior estabilidade, sendo considerado um método eficiente em cenários práticos.



Exios:

θ_0 no eixo x

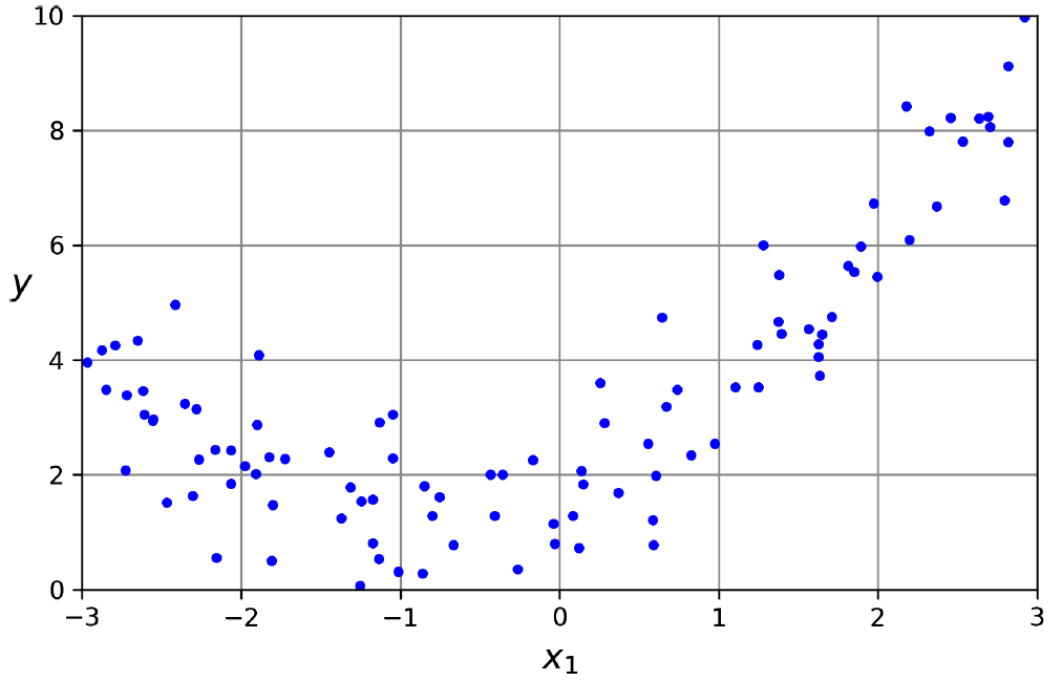
θ_1 no eixo y

São os parâmetros do modelo (por exemplo, intercepto e inclinação numa regressão linear simples).

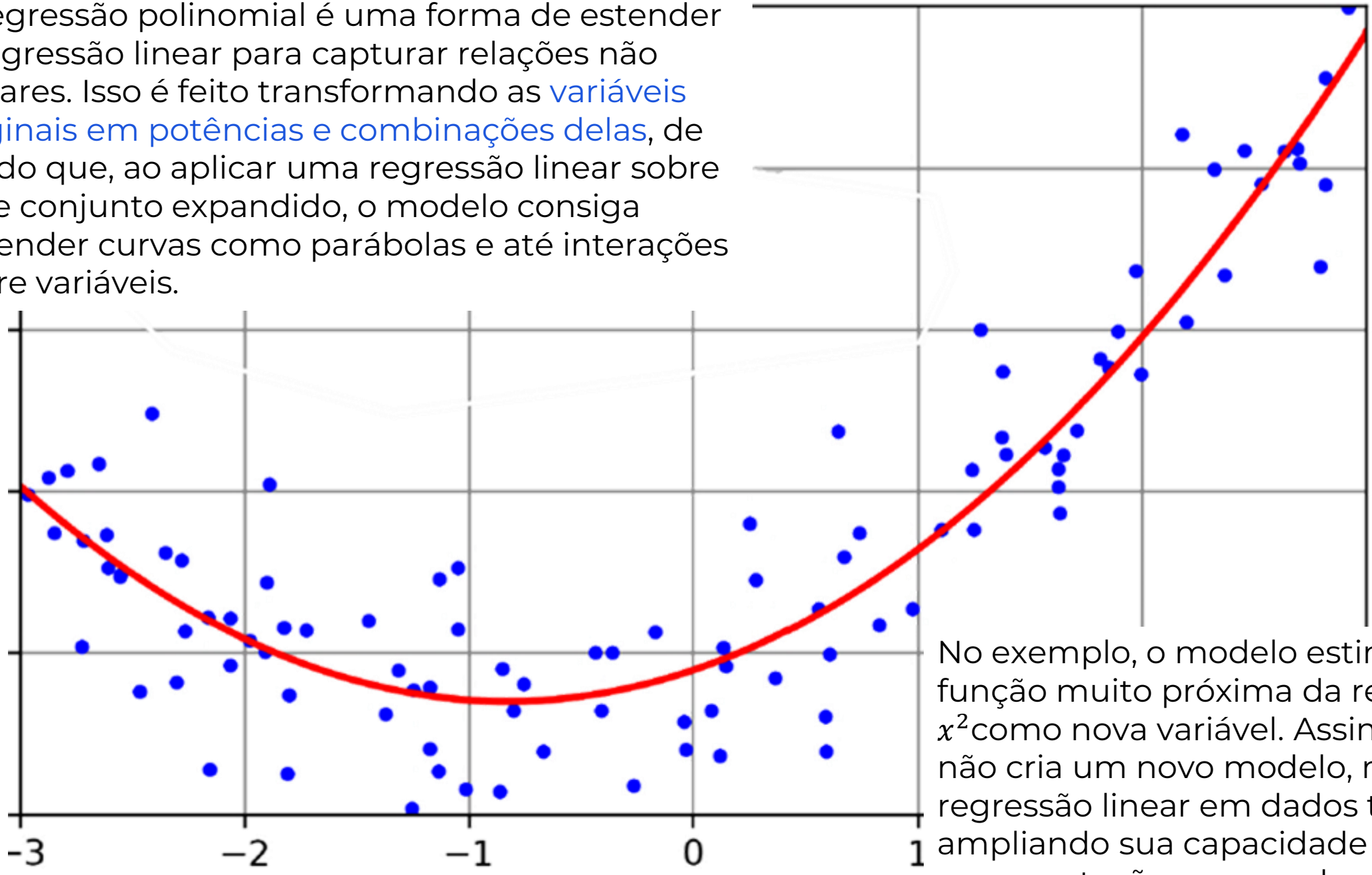
O gráfico mostra como os algoritmos ajustam θ_0 e θ_1 ao longo do treino, ou seja, o caminho percorrido pelos parâmetros até chegarem no mínimo da função de custo.

3. Regressão polinomial

A regressão polinomial surge quando os dados não podem ser bem representados por uma simples linha reta. Imagine que temos um conjunto de pontos que seguem uma parábola, algo como a equação $y = 0.5x^2 + x + 2$,mas com um pouco de ruído aleatório. Se tentarmos ajustar uma regressão linear tradicional, a reta nunca vai conseguir capturar esse padrão curvo, e o erro sempre será alto.



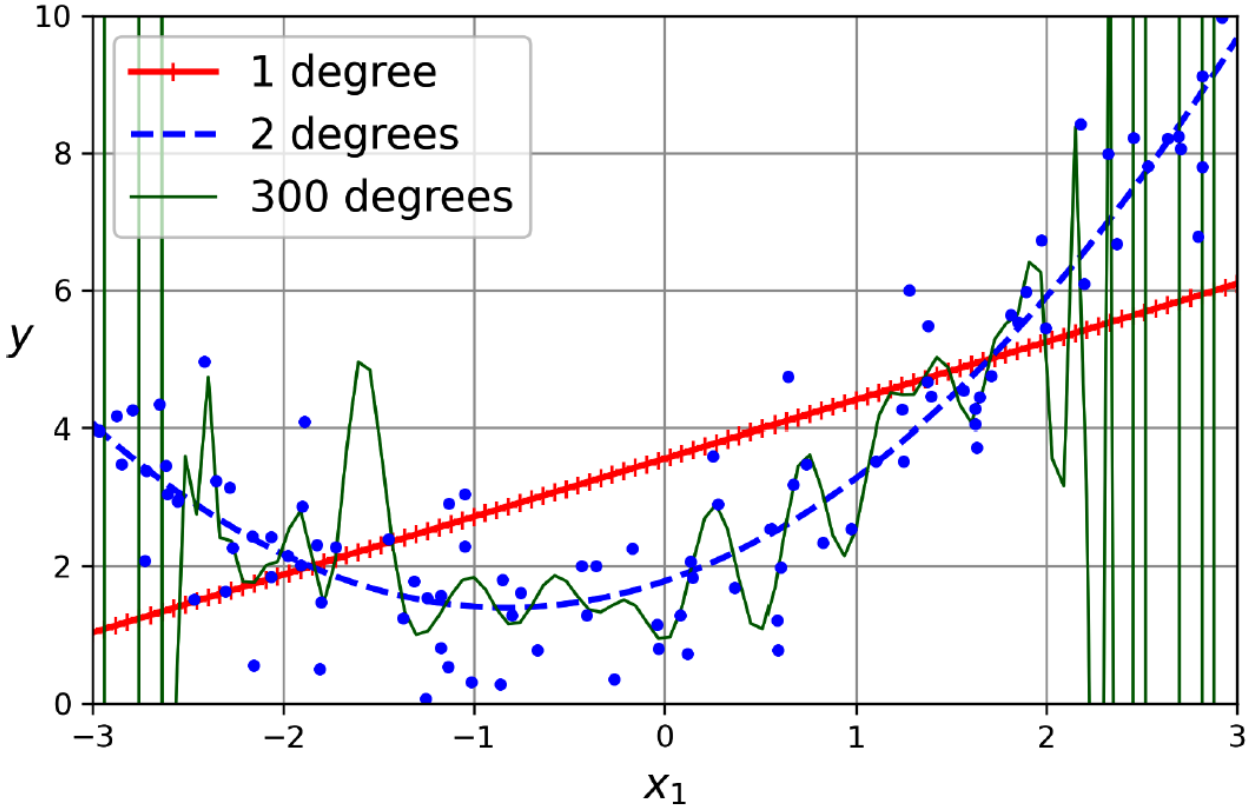
A regressão polinomial é uma forma de estender a regressão linear para capturar relações não lineares. Isso é feito transformando as variáveis originais em potências e combinações delas, de modo que, ao aplicar uma regressão linear sobre esse conjunto expandido, o modelo consiga aprender curvas como parábolas e até interações entre variáveis.



No exemplo, o modelo estimou uma função muito próxima da real ao adicionar x^2como nova variável. Assim, a técnica não cria um novo modelo, mas utiliza a regressão linear em dados transformados, ampliando sua capacidade de representação sem perder a simplicidade do método original.

4. Learning Curves

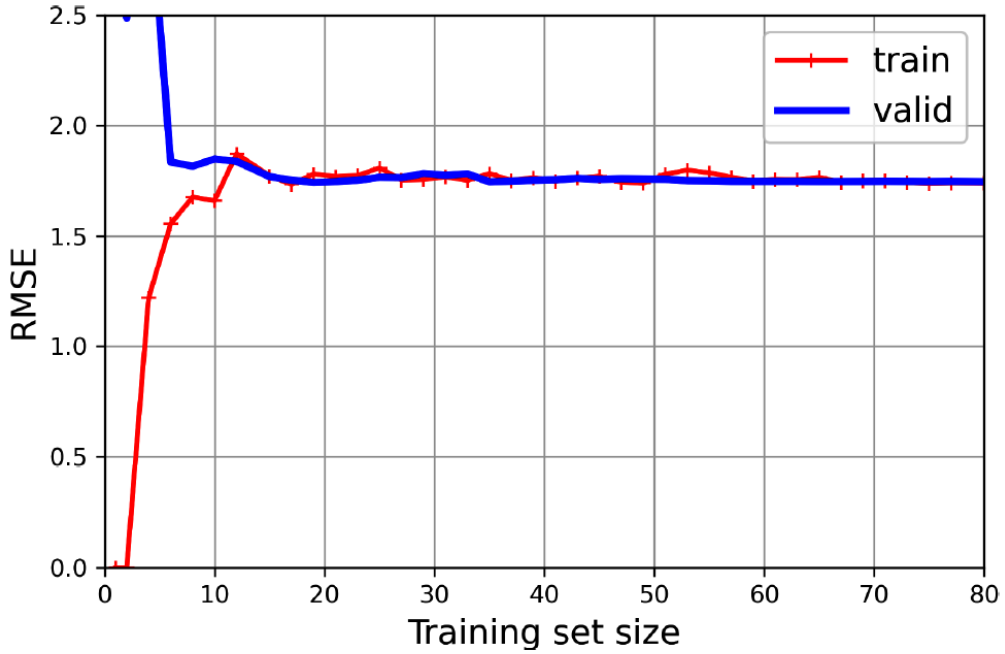
Quando usamos um modelo muito simples, como a regressão linear para dados que seguem uma curva, ele não consegue aprender bem: isso é underfitting. Por outro lado, quando usamos modelos muito complexos, como uma regressão polinomial de grau 300, o modelo acaba “decorando” os pontos de treino, cheio de ondulações — isso é overfitting.



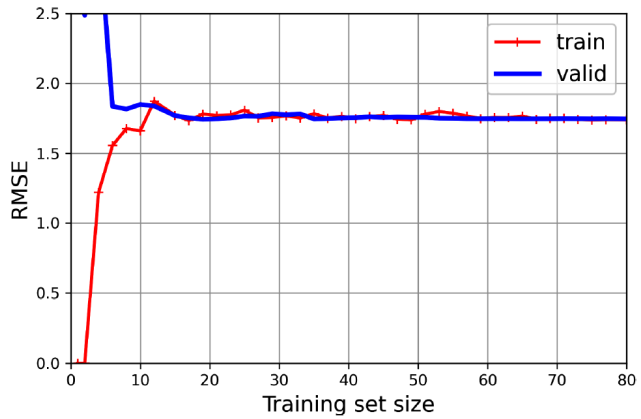
4.1 O que são Learning Curves?

As curvas de aprendizado mostram como o erro no treino e na validação evolui à medida que aumentamos o número de exemplos usados no treinamento. Assim, conseguimos identificar se o modelo está com underfitting ou overfitting.

4.2 Regressão Linear (Underfitting)

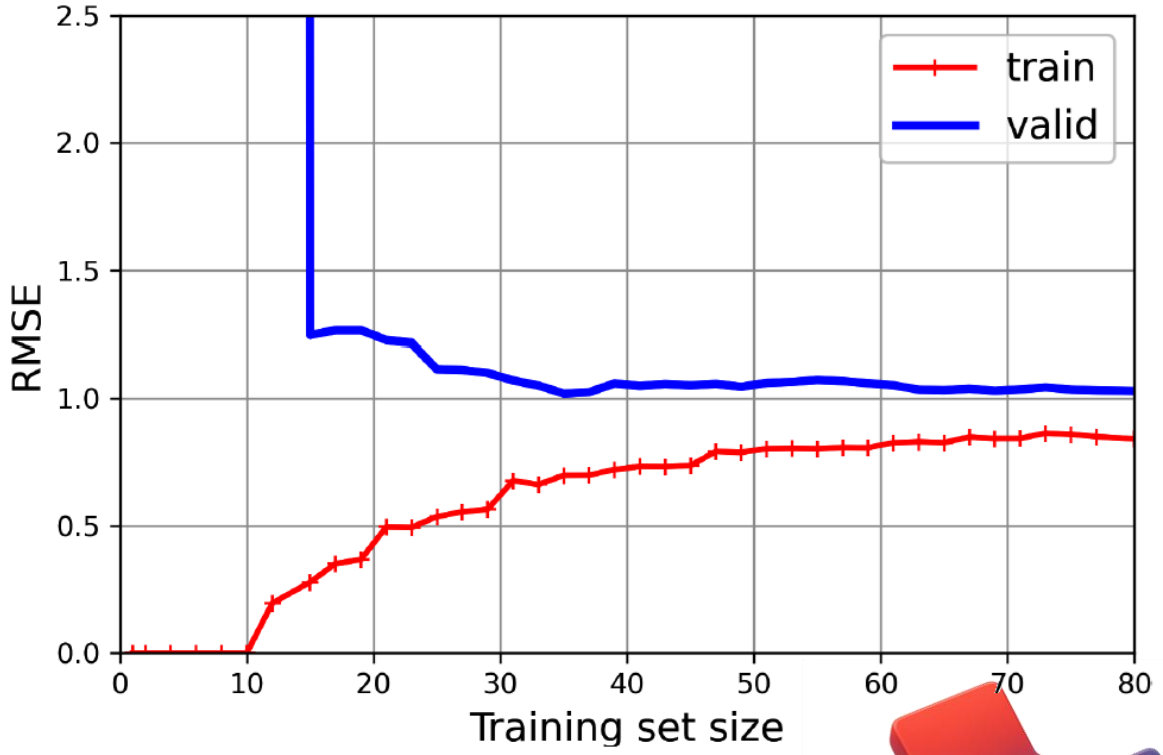


No caso da regressão linear, com poucos dados o erro de treino é quase zero. Conforme aumentamos os exemplos, o erro de treino sobe e para em um nível alto. O erro de validação começa grande, cai um pouco, mas também para em um patamar parecido. Esse comportamento indica underfitting: o modelo é simples demais, não importa quantos dados adicionemos..


train
valid
RMSE
Training set size
0.0
0.5
1.0
1.5
2.0
2.5
0
10
20
30
40
50
60
70
80

4.3 Polinômio de Grau 10 (Overfitting)

Já com uma regressão polinomial de grau 10, o erro de treino é muito baixo, mas o erro de validação fica mais alto e existe uma diferença clara entre as curvas. Esse “gap” é o sinal clássico de overfitting: o modelo aprendeu bem o treino, mas não generalizou. Com muitos mais dados, as curvas tenderiam a se aproximar.



As curvas de aprendizado são uma ferramenta visual poderosa para entender se o problema é underfitting (modelo muito simples) ou overfitting (modelo muito complexo).

Se ambos erros (treino e validação) são altos e próximos: underfitting.
Se o erro de treino é baixo, mas o de validação é alto: overfitting.

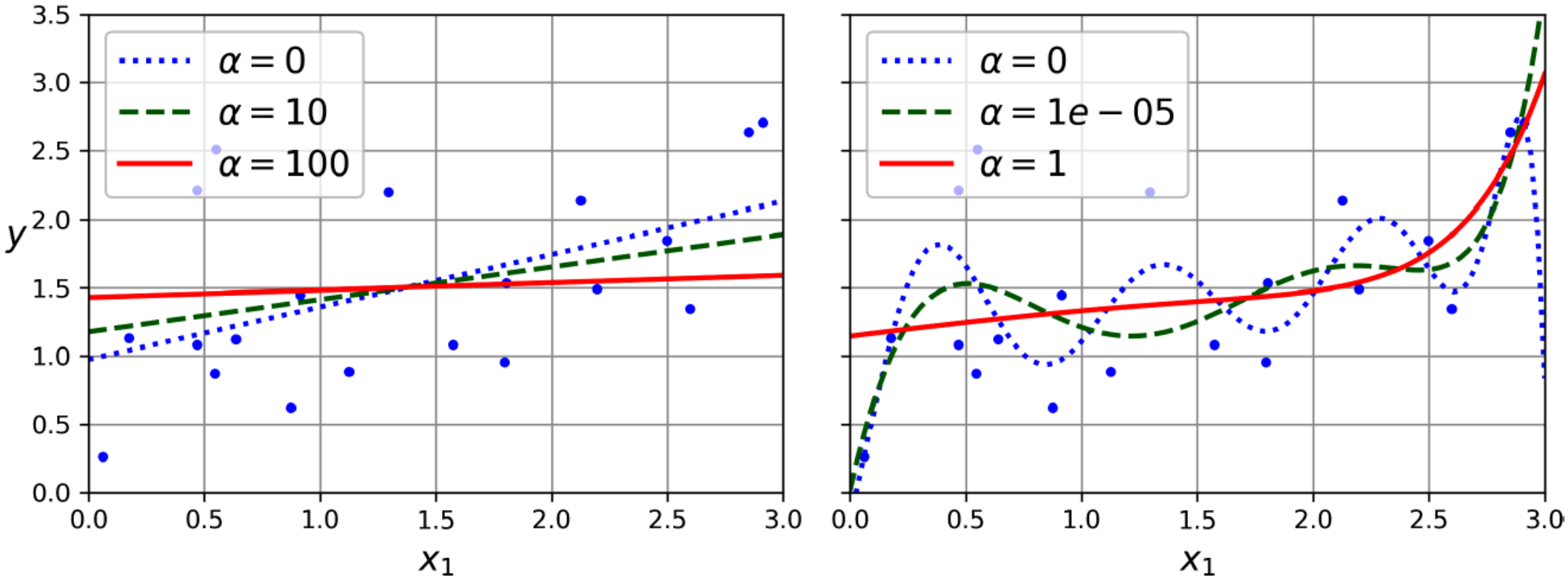
5. Ridge Regression

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + \frac{\alpha}{m}\Sigma_{i=1}^{n}\theta_i^2$$

Na Ridge Regression, também chamada de regularização de Tikhonov, pegamos a regressão linear comum e acrescentamos ao erro quadrático médio (MSE) um termo extra que penaliza pesos grandes. O modelo não só precisa se ajustar aos dados, mas também deve manter seus coeficientes pequenos. O grau dessa penalização é controlado por um hiperparâmetro chamado α (alfa):

Se $\alpha = 0$, é a regressão linear normal.
Se α é muito grande, os pesos quase zeram e a linha fica praticamente horizontal.
Valores intermediários dão o equilíbrio certo entre ajuste e simplicidade.

A regularização Ridge reduz a variância do modelo, tornando as curvas mais estáveis e menos exageradas, mas aumenta um pouco o viés. No gráfico a seguir, vemos dois cenários: à esquerda, modelos lineares simples com diferentes valores de α; à direita, modelos polinomiais de grau 10 também regularizados com Ridge. À medida que α aumenta, as previsões ficam mais “achatadas” e razoáveis.



6. Lasso Regression

A Lasso Regression é uma forma de regressão linear regularizada que usa a norma L1 (soma dos valores absolutos dos pesos) como penalização. Diferente da Ridge Regression, que usa a norma L2 (soma dos quadrados dos pesos) e apenas reduz a magnitude dos coeficientes, a Lasso tem a característica de zerar os pesos menos relevantes. Com isso, ela realiza automaticamente uma seleção de variáveis, resultando em modelos mais simples e esparsos, que mantêm apenas as features mais importantes.

Uma característica importante da Lasso é que ela tende a zerar os coeficientes menos relevantes, ou seja, faz automaticamente seleção de variáveis. Isso gera um modelo mais simples e esparso, com poucos pesos diferentes de zero.

Intuitivamente, isso acontece porque a penalização $\ell 1$ cria “quinas” nas curvas de nível da função de custo. Quando o gradiente descendente chega perto dessas quinas, alguns parâmetros são empurrados para zero rapidamente.

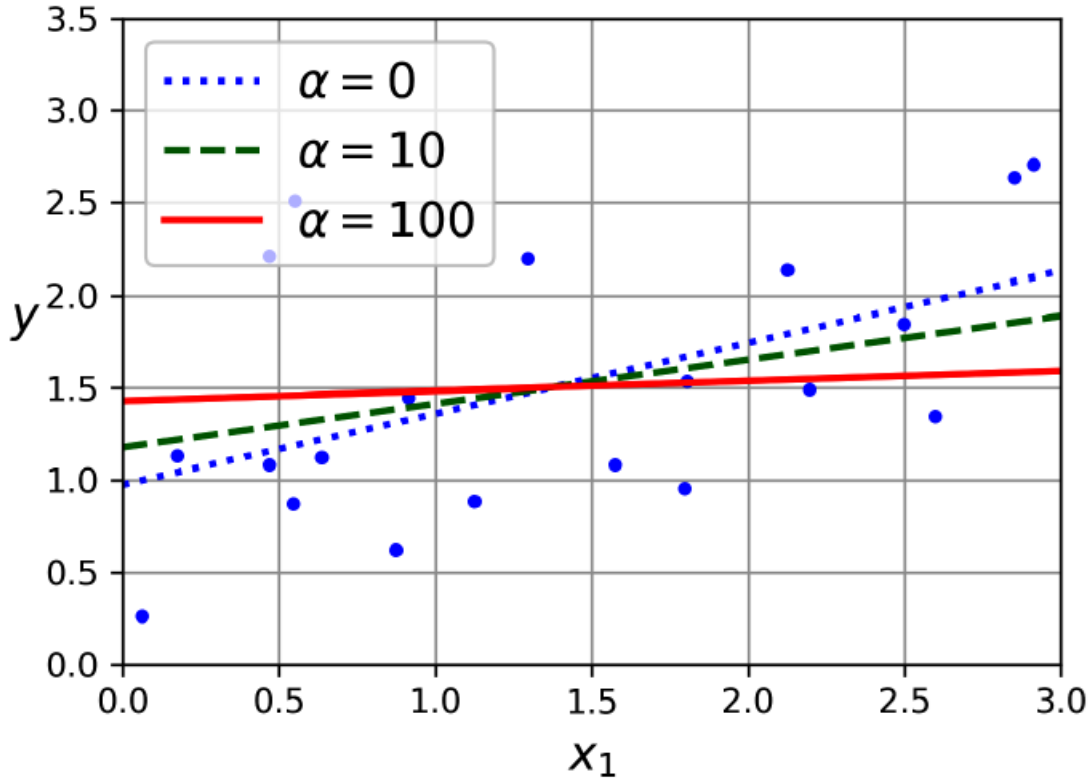
O hiperparâmetro $\boldsymbol{\alpha}$ controla a força da regularização:
- Valores maiores: mais coeficientes zerados.
- Valores menores: modelo mais parecido com a regressão linear comum.

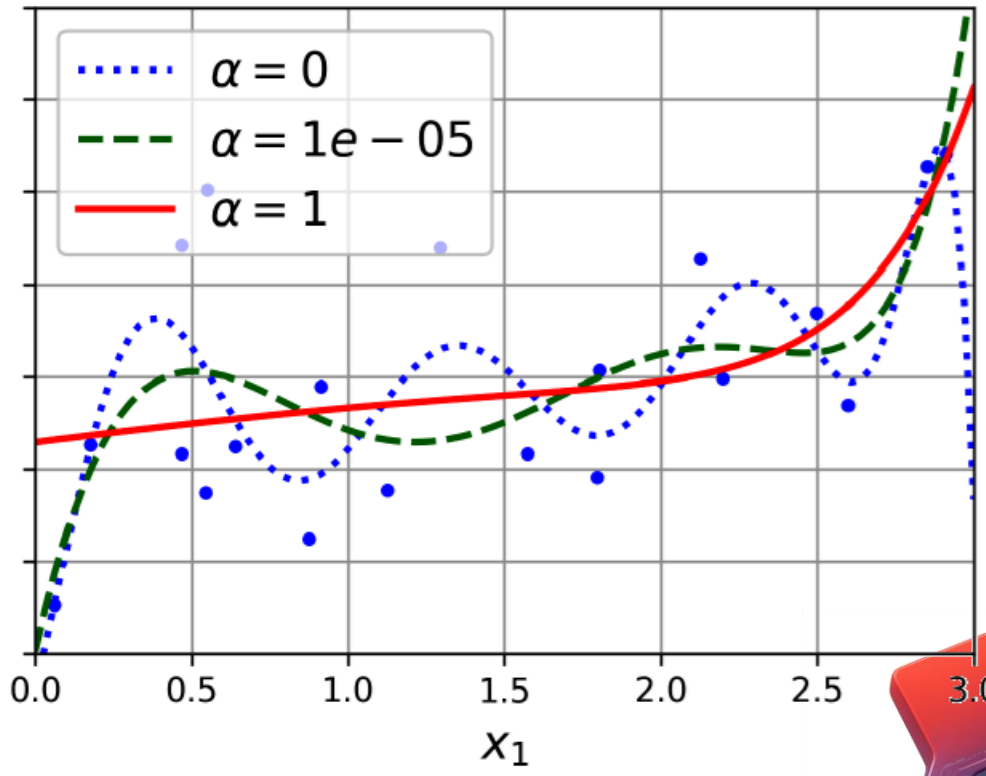
Em resumo, a Lasso não só reduz a magnitude dos pesos como a Ridge, mas também pode eliminar variáveis irrelevantes, tornando-se útil quando suspeitamos que apenas algumas features são realmente importantes.

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + 2\alpha\Sigma_{i=1}^{n} \left|\theta_i\right|$$

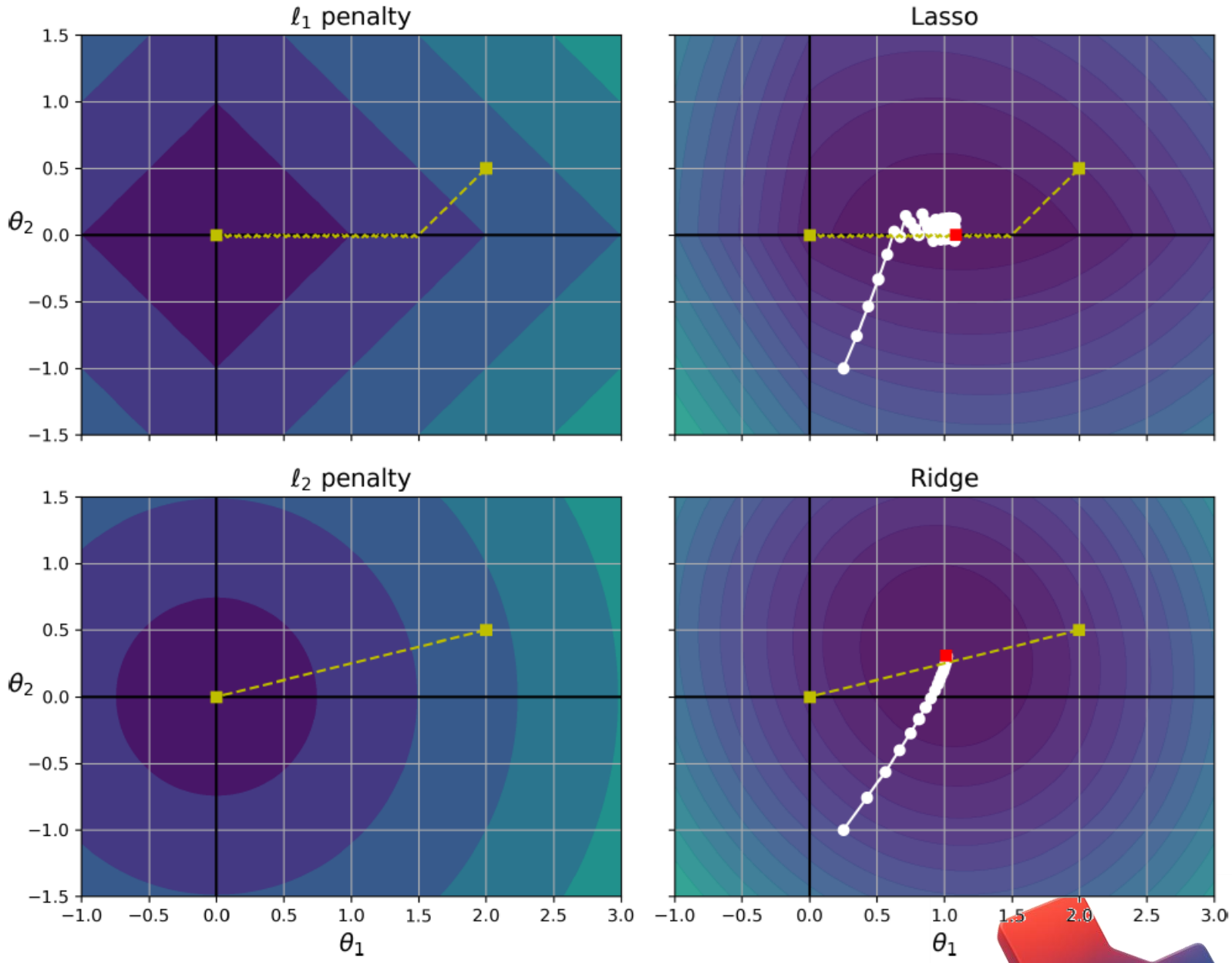
- $J(\theta)$ → função de custo total (o que queremos minimizar).
- $MSE(\theta)$ → erro quadrático médio (mede o quanto o modelo erra nas previsões).
- 2α → fator de regularização; α é o hiperparâmetro que controla a força da penalização.
- $\sum_{i=1}^{n} |\theta_i|$ → soma dos valores absolutos dos pesos (norma $\ell 1$).
- **Interpretação:** além de minimizar o erro de previsão (MSE), o modelo é penalizado quando os pesos ficam grandes. Essa penalização força muitos pesos a irem para **zero**, realizando **seleção automática de variáveis**.

O impacto da penalização $\ell 1$ é que alguns coeficientes não apenas diminuem, mas chegam a zero. Isso significa que a Lasso faz automaticamente uma forma de seleção de variáveis, mantendo apenas as mais relevantes. Assim, ela gera modelos esparsos, mais simples e fáceis de interpretar.





O motivo da Lasso zerar coeficientes pode ser entendido olhando as curvas de nível da função de custo. A penalização $\ell 1$ cria “quinas” ao longo dos eixos. Quando o gradiente descendente chega próximo dessas regiões, ele tende a empurrar alguns parâmetros para zero, mantendo outros. Já a Ridge ($\ell 2$) cria curvas suaves que nunca levam os coeficientes a zero.

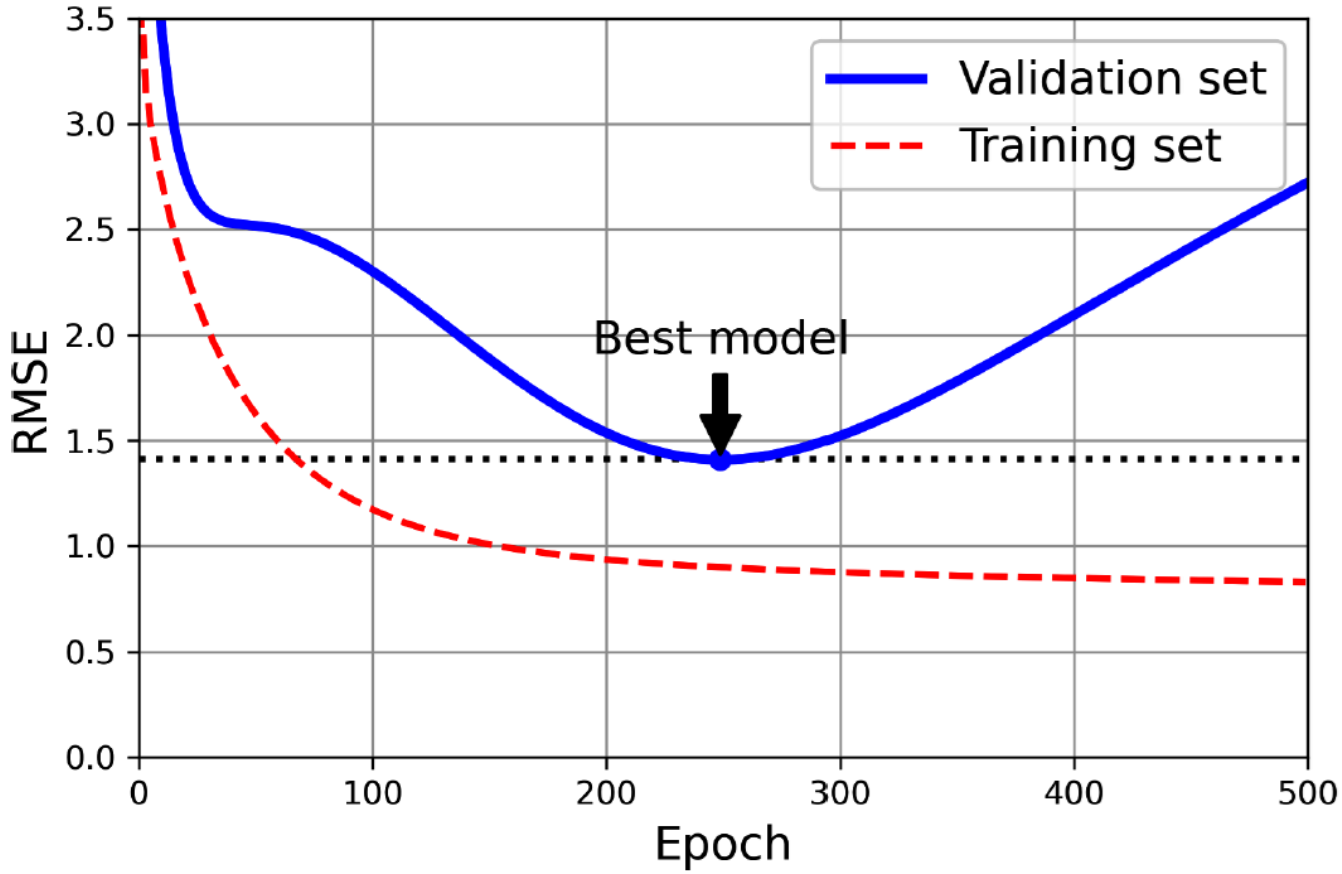


6. Elastic Net Regression

A Elastic Net combina a regularização da Ridge ($\ell 2$) e da Lasso ($\ell 1$) em uma única função de custo. O parâmetro **r** controla a mistura: r = 0 equivale a Ridge, r = 1 equivale a Lasso, e valores intermediários dão um meio-termo.

Na prática, é sempre bom usar alguma forma de regularização. Ridge é um bom ponto de partida. Mas se você acredita que apenas algumas features são relevantes, prefira Lasso ou Elastic Net, porque elas reduzem pesos inúteis a zero. Entre as duas, Elastic Net é geralmente mais estável, especialmente quando temos muitas variáveis ou quando elas são muito correlacionadas.

Outra forma de regularização é o early stopping. Em vez de continuar treinando até o final, interrompemos o processo assim que o erro de validação atinge o valor mínimo. Isso impede que o modelo comece a decorar os dados de treino e evita o overfitting. Geoffrey Hinton chegou a chamá-lo de um “lindo almoço grátis”, pela simplicidade e eficiência.



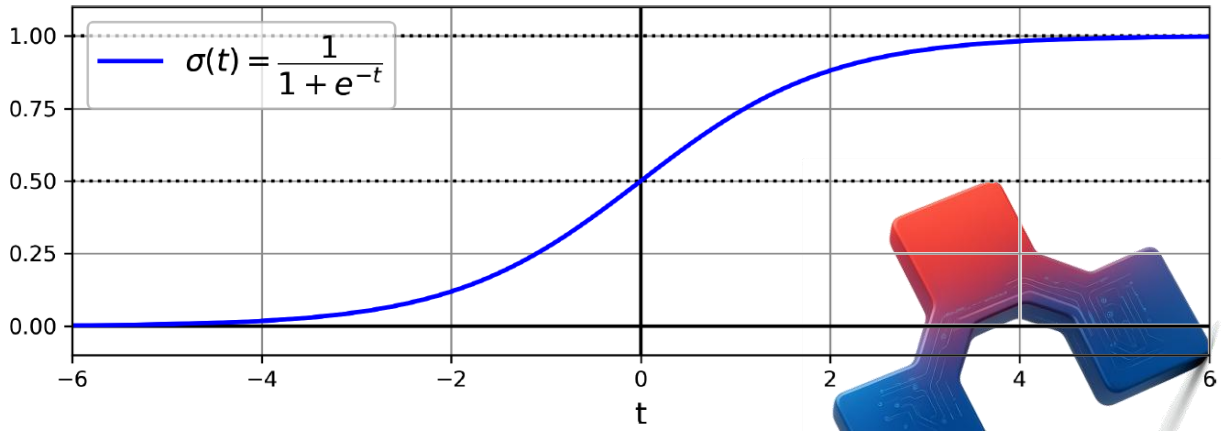
Na prática, implementamos early stopping acompanhando o erro de validação a cada época. Sempre que o erro de validação melhora, salvamos uma cópia do modelo. No fim do treinamento, recuperamos o melhor modelo salvo. Essa técnica é amplamente usada em redes neurais e outros algoritmos iterativos

7. Logistic Regression

A Logistic Regression é um algoritmo de classificação binária. Ela estima a probabilidade de uma instância pertencer à classe positiva (1). Se a probabilidade for ≥ 0,5, prevê 1; caso contrário, prevê 0.

Apesar do nome, a Logistic Regression não serve para prever valores contínuos, e sim para classificação binária. Ela estima a probabilidade de uma instância pertencer à classe positiva, que chamamos de 1. Se a probabilidade for maior ou igual a 0,5, o modelo prevê 1; caso contrário, prevê 0.

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$



De outra forma:

A Logistic Regression é um algoritmo de classificação binária. Em vez de prever valores contínuos como na regressão linear, ela estima a probabilidade de uma instância pertencer à classe positiva (1).
Se a probabilidade for maior ou igual a 0,5, o modelo prevê classe 1; caso contrário, prevê classe 0.

O modelo calcula uma soma ponderada dos atributos e aplica a função sigmoide $\sigma(t)$,que transforma o resultado para o intervalo entre 0 e 1:

$$\hat{p} = \sigma(\theta^T x)$$

Esse $\hat{p}$ é justamente a probabilidade estimada.

Para treinar o modelo, usamos a função de custo chamada log loss. Ela mede o quanto o modelo erra nas probabilidades estimadas.

Se a classe verdadeira for $y = 1$, a penalização é $\log(\hat{p})$
Se a classe verdadeira for $y = 0$, a penalização é $\log(1 - \hat{p})$

Assim, prever 0,01 quando a resposta é 1 gera um custo enorme, enquanto prever 0,99 gera custo quase zero.

Fórmula do custo individual:

$$c(\boldsymbol{\theta}) = \begin{cases} -\log(\hat{p}) & \text{if } y = 1 \\ -\log(1 - \hat{p}) & \text{if } y = 0 \end{cases}$$

Fórmula do custo médio (log loss):

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\Sigma_{i=1}^{m}\left[y^{(i)}log\left(\hat{p}^{(i)}\right) + \left(1 - y^{(i)}\right)log\left(1 - \hat{p}^{(i)}\right)\right]$$

Essa é a função que o modelo busca minimizar.

Não existe uma fórmula direta (equação fechada) para encontrar os parâmetros θ da Logistic Regression. Por isso, usamos algoritmos iterativos, como o gradiente descendente.

A boa notícia é que a função de custo log loss é convexa, isso significa que não existem mínimos locais; o gradiente descendente sempre converge para a melhor solução global (desde que a taxa de aprendizado seja adequada).

$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \left(\sigma\left(\boldsymbol{\theta}^\top \mathbf{x}^{(i)}\right) - y^{(i)} \right) x_j^{(i)}$$

Essa equação mostra que:
- Calculamos a diferença entre a probabilidade prevista e o valor real ($\hat{p} - y$.(
- Multiplicamos pelo valor da feature x_j.
- Fazemos isso para todos os exemplos e tiramos a média.
- O resultado indica como ajustar cada peso θ_j.

Assim, a cada iteração os parâmetros vão se ajustando para que as probabilidades previstas se aproximem da realidade.

FPFtech