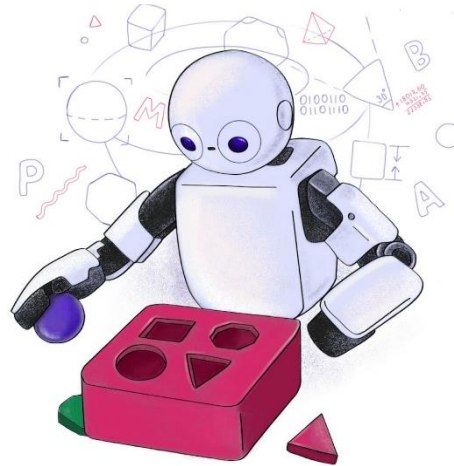


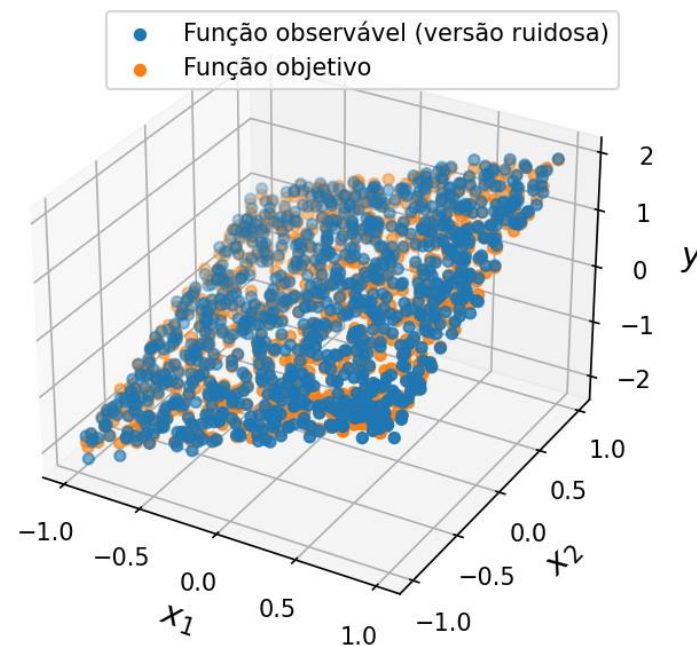
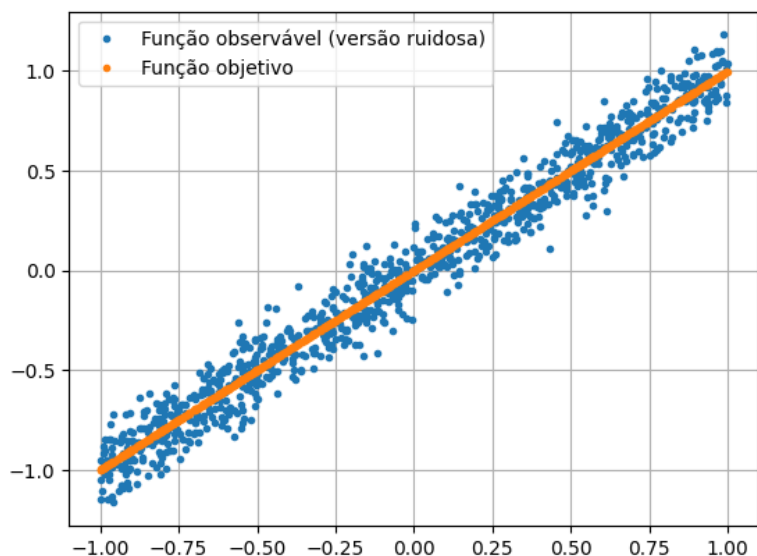
# TP555 - Inteligência Artificial e Machine Learning: *Regressão com Modelos Não- Lineares com Relação aos Atributos*



***Inatel***

Felipe Augusto Pereira de Figueiredo  
felipe.figueiredo@inatel.br

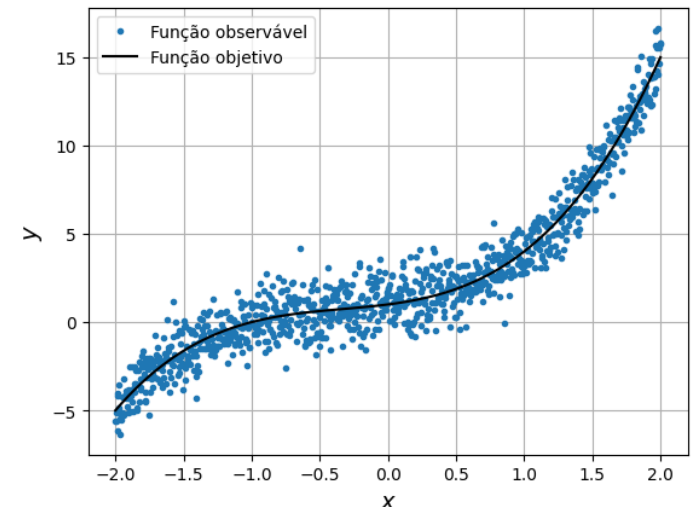
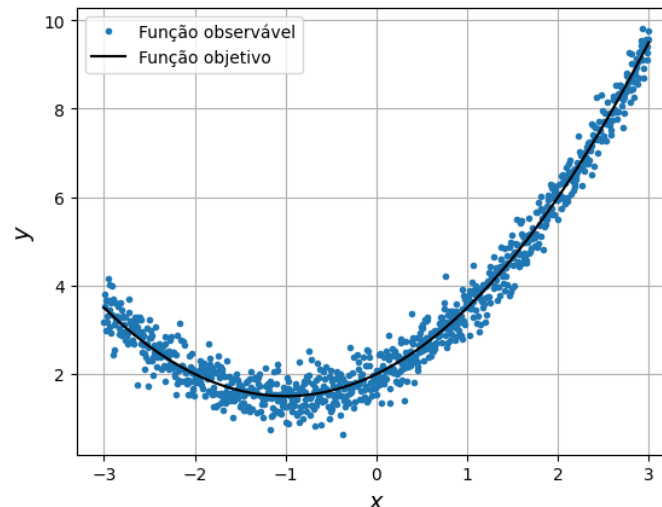
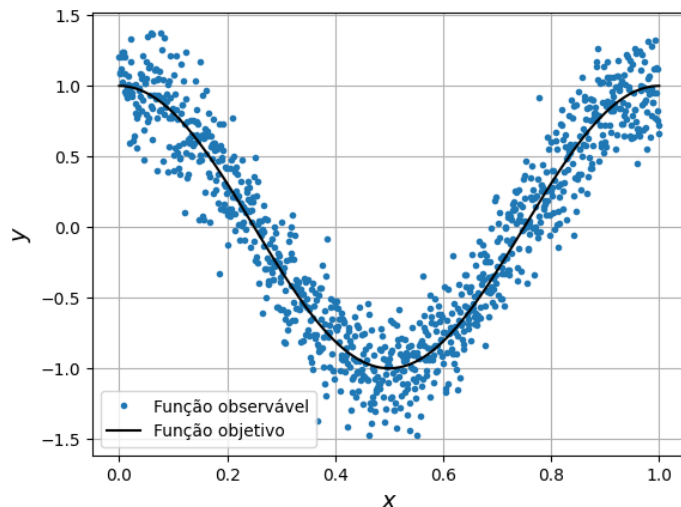
Até agora, usamos *funções hipótese com formato de hiperplanos*, e.g., retas e planos, para aproximar *mapeamentos lineares* entre os atributos e o valor esperado, mas *e se os mapeamentos forem não lineares?*



O que podemos fazer quando *hiperplanos não se ajustam bem aos dados?*

# Mapeamentos não lineares

- Observem as figuras abaixo, uma **reta** claramente *não seria uma boa escolha para aproximar esses mapeamentos não lineares*.
  - *Retas não capturariam o comportamento das funções abaixo*, pois elas não têm *complexidade ou flexibilidade* (i.e., graus de liberdade) o suficiente para isso.
- Portanto, qual tipo de função hipótese seria mais apropriada para aproximar esses comportamentos não lineares?



# Extensão para modelos não-lineares com relação aos atributos

- **Modelos não-lineares** constroem uma aproximação por meio da **combinação linear de funções-base não-lineares**, da forma

$$h(\mathbf{x}) = \hat{y}(\mathbf{x}) = a_0 + a_1 b_1(\mathbf{x}) + \cdots + a_M b_M(\mathbf{x}),$$

onde  $b_m(\mathbf{x}): \mathbb{R}^K \rightarrow \mathbb{R}$ , denota a  $i$ -ésima **função-base**.

- Portanto, por ser linear com relação aos pesos, os **resultados encontrados anteriormente são facilmente estendidos para o caso não-linear** bastando redefinir o **vetor de atributos**,  $\mathbf{x}(i)$ , como o **vetor de funções-base**

$$\mathbf{x}(i) = [1, b_1(\mathbf{x}(i)), \dots, b_M(\mathbf{x}(i))]^T.$$

- Exemplos de funções base:

- $b_m(\mathbf{x}) = x_1 * x_2$
- $b_m(\mathbf{x}) = \log_{10} x_1$
- $b_m(\mathbf{x}) = x_1^2$

# Linearização

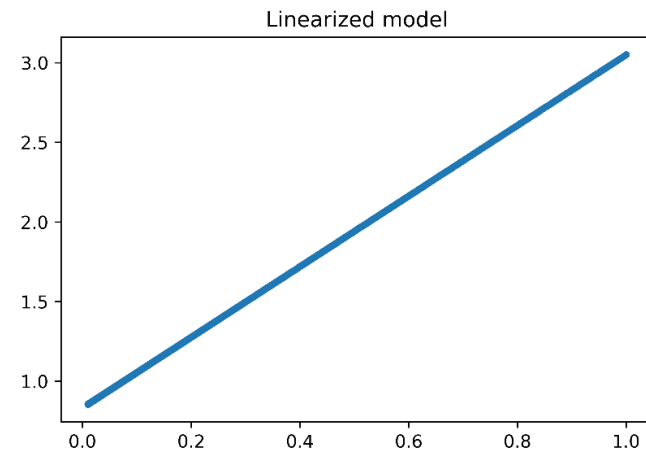
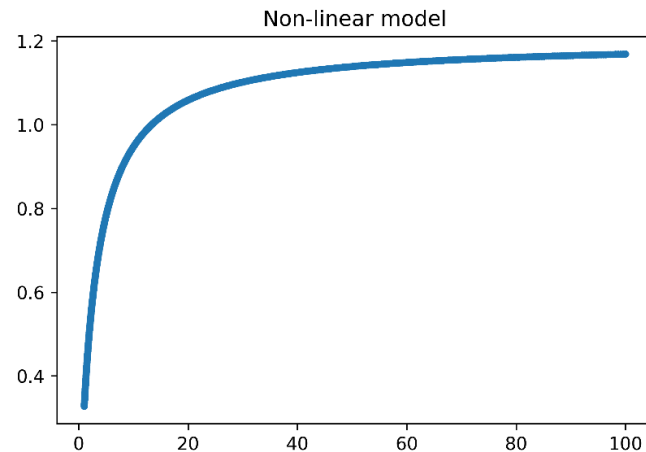
- Em geral, processos e modelos físicos *não são lineares* (e desta vez em relação aos pesos também), o que dificulta sua interpretação.
- Porém, existem **modelos não-lineares** que são *intrinsecamente lineares*, pois podem ser tornados lineares em relação aos pesos *através de uma modificação*.
- Por exemplo, as equações abaixo podem ser transformadas em uma combinação linear de funções base não lineares.

$$y = a_0 x_1^{a_1} e^{a_2 x_2},$$

$$y = \frac{a_0 x_1}{a_1 + x_1}.$$

- A vantagem é que modelos lineares são mais fáceis de interpretar.

# Exemplo: Linearização



A equação,

$$y = \frac{a_0 x_1}{a_1 + x_1},$$

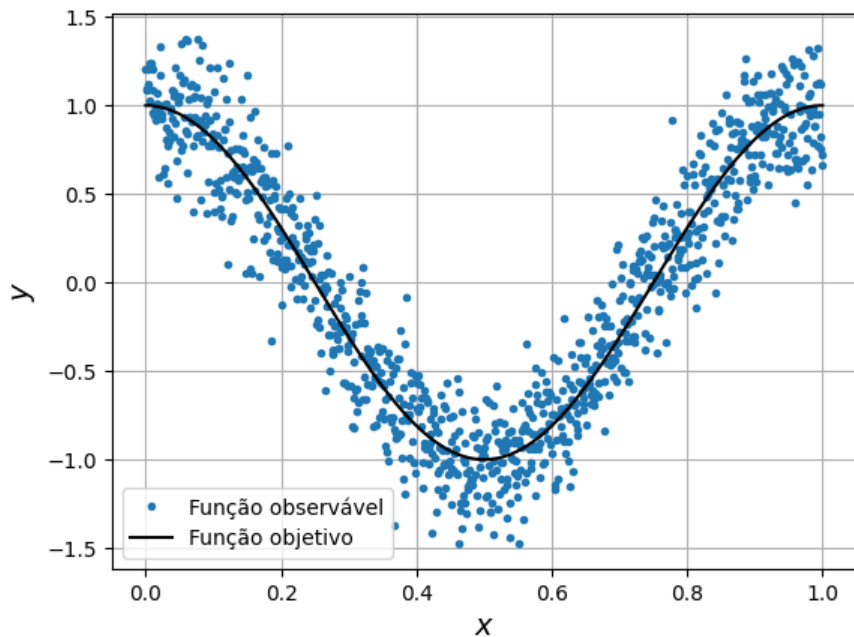
pode ser reescrita como

$$\frac{1}{y} = \frac{a_1 + x_1}{a_0 x_1} = \frac{1}{a_0} + \frac{a_1}{a_0} \frac{1}{x_1} = a'_0 + a'_1 x'_1,$$

que é linear em relação aos pesos transformados.

- Depois do modelo ser linearizado, os pesos podem ser encontrados com a equação normal ou com o gradiente descendente.

# Regressão polinomial



- O teorema da aproximação de **Stone-Weierstrass** diz que “qualquer função contínua no intervalo fechado  $[a, b]$  pode ser uniformemente aproximada por um **polinômio**”.
- Portanto, podemos aproximar **qualquer tipo de mapeamento (linear ou não linear)** com **polinômios**, bastando apenas **encontrar o grau (ou ordem) ideal**.



# Regressão polinomial

- Exemplo de um polinômio de grau 4\* com três atributos,  $x_1$ ,  $x_2$  e  $x_3$

$$y(x_1, x_2, x_3) = a_0 + a_1x_1 + a_2x_2 + a_3x_1x_3 + a_5x_1^3 + a_4x_1x_2x_3^2$$

- O grau do polinômio é o maior valor resultante da soma dos expoentes dos atributos de um monômio
- Percebam que em alguns monômios existe a **combinação dos atributos originais**, formando novos atributos (ou **funções base**).
- Um problema com polinômios é que a presença de **outliers** nos dados de treinamento impacta o desempenho do modelo.

# Regressão polinomial

- Por simplicidade didática, inicialmente, nós consideraremos **funções hipóteses polinomiais em uma variável** (i.e., com **um** atributo)

$$h(x_1(n)) = a_0 + a_1 x_1(n) + a_2 x_1^2(n) + \dots + a_M x_1^M(n) = \mathbf{a}^T \mathbf{x}(n),$$

onde  $n = 1 \dots N$  é o número da amostra,  $M$  é o grau do polinômio,  $\mathbf{a} = [a_0 \ a_1 \ a_2 \ \dots \ a_M]^T \in \mathbb{R}^{M+1 \times 1}$ ,  $\mathbf{x}(n) = [x_0 \ x_1(n) \ x_1^2(n) \ \dots \ x_1^M(n)]^T \in \mathbb{R}^{M+1 \times 1}$  e  $x_0 = 1$  é o atributo de *bias*, associado ao peso de *bias*,  $a_0$ .

- **Todos resultados encontrados anteriormente** (equação normal, vetor gradiente para implementação do algoritmo do gradiente descendente, escalonamento) **são diretamente estendidos para funções hipótese polinomiais**.

# Regressão polinomial

- Só precisamos nos lembrar que o **vetor de atributos**,  $\mathbf{x}$ , e consequentemente, a **matriz de atributos**,  $\mathbf{X}$ , são **compostos pelas funções base**.

- Por exemplo, para a seguinte **função hipótese polinomial**

$$h(x_1(n)) = a_0 + a_1 x_1(n) + a_2 x_1^2(n) + \cdots + a_M x_1^M(n),$$

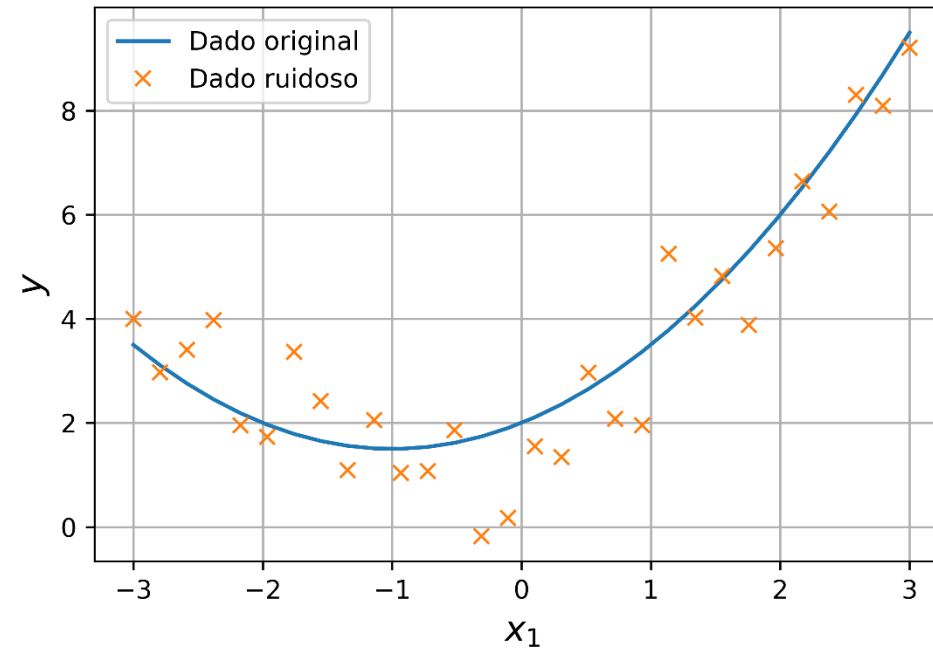
- a **matriz de atributos polinomial**,  $\mathbf{X}$ , fica da seguinte forma

$$\mathbf{X} = \begin{bmatrix} 1 & x_1(0) & x_1^2(0) & \cdots & x_1^M(0) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1(N-1) & x_1^2(N-1) & \cdots & x_1^M(N-1) \end{bmatrix} \in \mathbb{R}^{N \times M+1},$$

onde cada coluna contém uma **função base**.

Porém, o desafio agora é *encontrar o grau do polinômio* que melhor aproxime os dados.

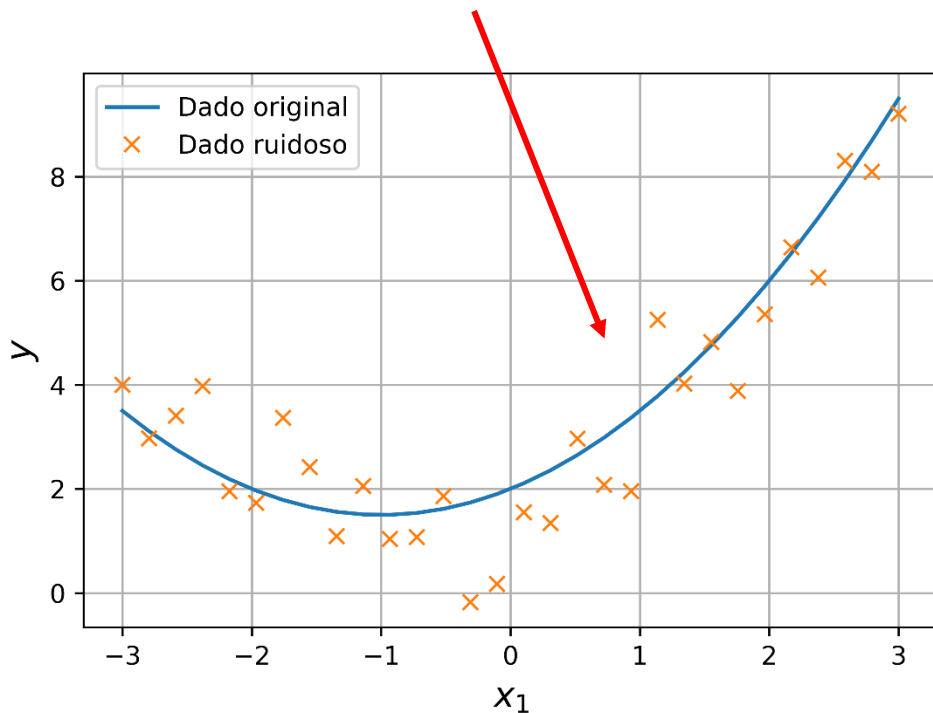
# Exemplo de regressão usando polinômio



Usando apenas os ***dados ruidosos***, ***qual a ordem e pesos de um polinômio*** para que ele se aproxime da ***função objetivo*** da melhor forma possível?

# Exemplo de regressão usando polinômio

Função objetivo: polinômio de ordem 2.



A partir do dados ruidosos, queremos encontrar um polinômio (pesos e ordem) que melhor se aproxime da função objetivo.

- Para exemplificar essa questão da busca pela ordem do polinômio aproximador geramos **30 exemplos** da seguinte função objetivo (polinômio de ordem 2)

$$y(x_1(n)) = 2 + x_1(n) + 0.5x_1^2(n),$$

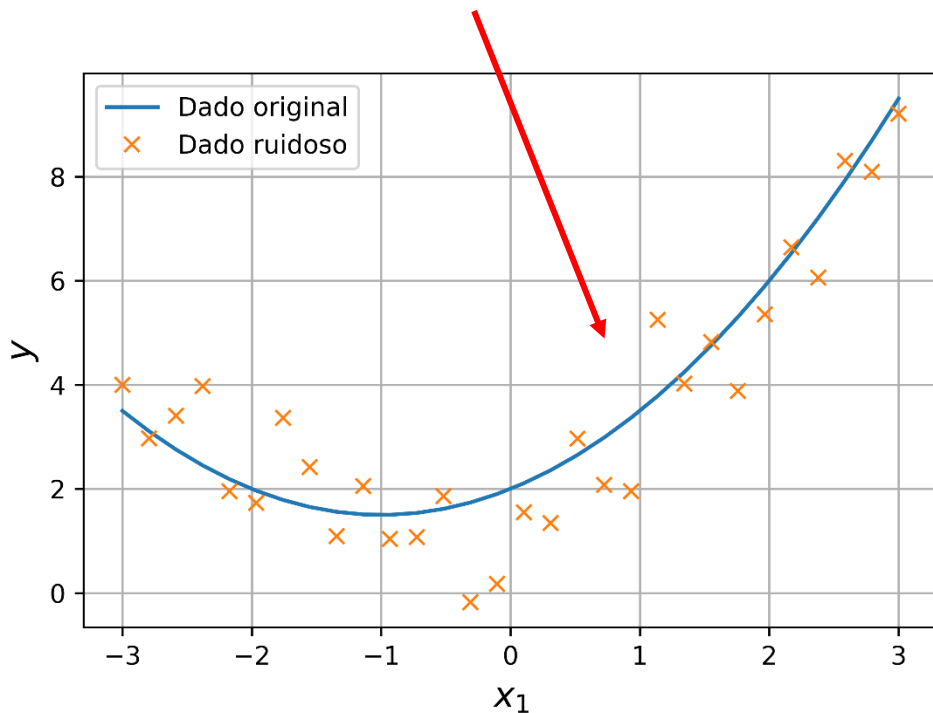
e adicionamos ruído Gaussiano branco,  $w(n)$

$$y_{\text{noisy}}(x_1(n)) = y(x_1(n)) + w(n),$$

onde  $x_1(n)$  são valores linearmente espaçados entre -3 e 3 e  $w(n) \sim N(0, 1)$ .

# Exemplo de regressão usando polinômio

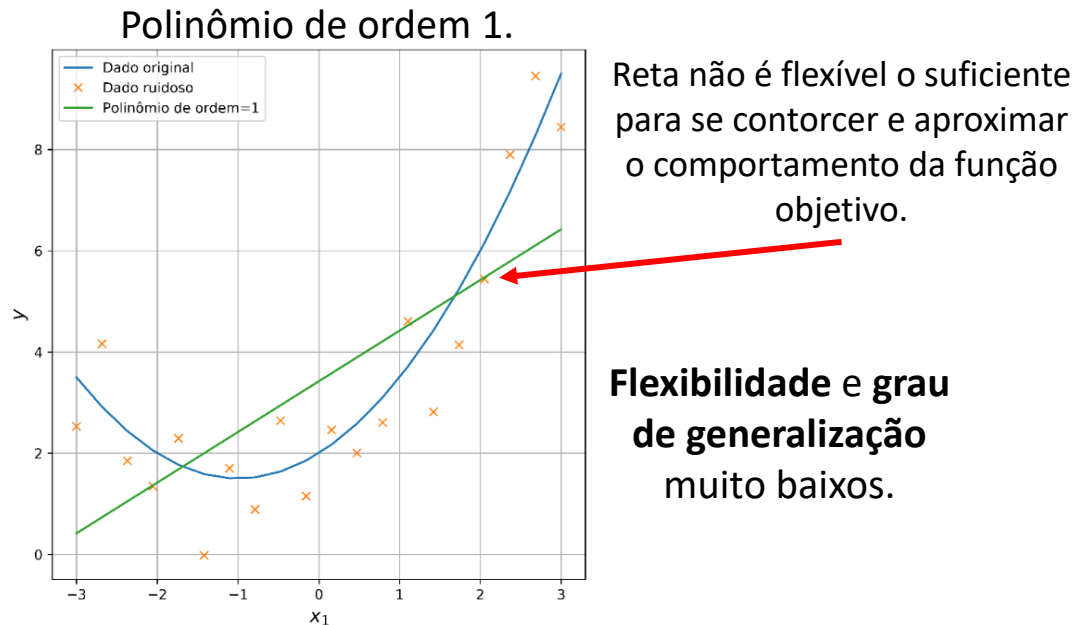
Função objetivo: polinômio de ordem 2.



A partir do dados ruidosos, queremos encontrar um polinômio (pesos e ordem) que melhor se aproxime da função objetivo.

- Vamos usar uma **função hipótese polinomial** para **aproximar a função objetivo a partir dos dados ruidosos**.
- Porém, surge uma dúvida, **e se não soubéssemos a ordem por trás do modelo gerador, qual grau deveríamos utilizar?**

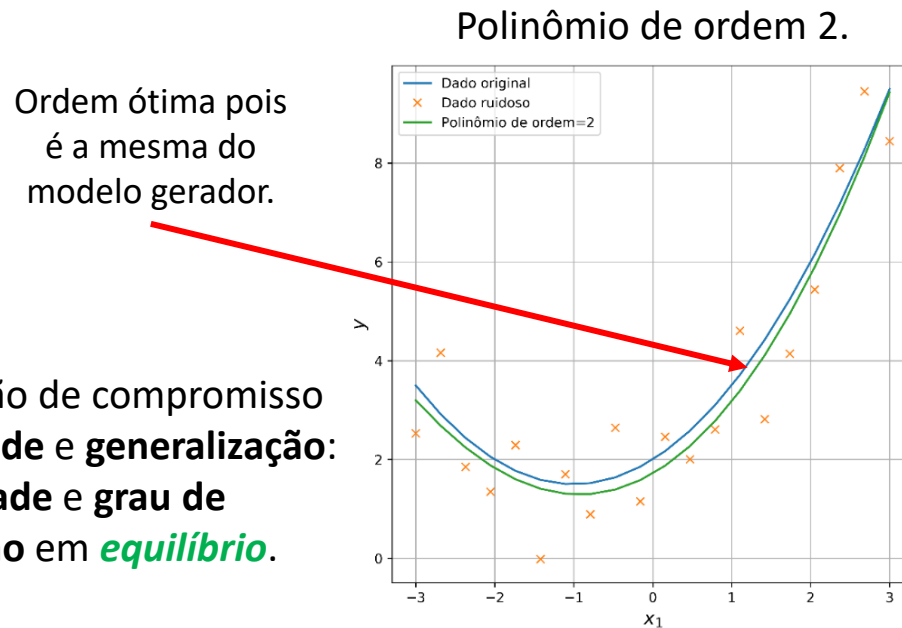
# Regressão polinomial: Qual ordem usar?



- Polinômio de ordem 1 (i.e., reta) não tem flexibilidade o suficiente para aproximar o comportamento por trás das amostras ruidosas, ou seja, a função objetivo.
- O erro (MSE) é alto para exemplos dos conjuntos de treinamento e de validação (i.e., exemplos não vistos durante o treinamento).
- Efeito conhecido como ***subajuste*** ou ***underfitting***: ***flexibilidade*** e ***grau de generalização*** muito baixos.



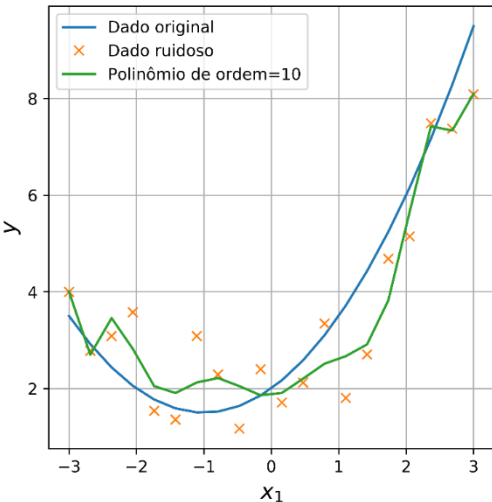
# Regressão polinomial: Qual ordem usar?



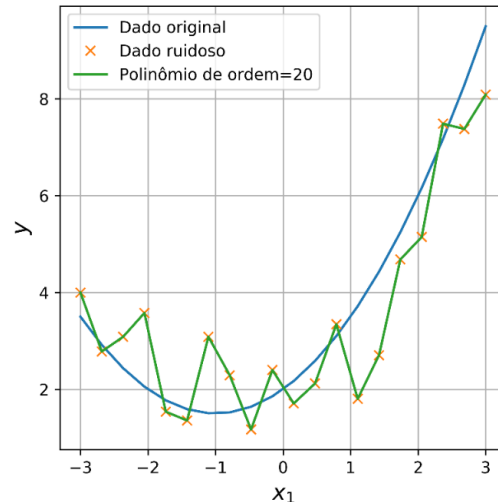
- Porém, como esperado, o polinômio de ordem 2 produz a melhor aproximação da função objetivo, errando pouco para exemplos dos conjuntos de treinamento e validação.
  - Esse modelo encontra uma relação de compromisso entre **flexibilidade** e **grau de generalização**.
  - Essa aproximação será melhor quanto maior for o conjunto de treinamento e/ou menor o ruído.

# Regressão polinomial: Qual ordem usar?

Polinômio de ordem 10.



Polinômio de ordem 20.



Polinômio de ordem 30.



- Polinômios com ordem maior do que 2 tendem a produzir ***aproximações perfeitas*** dos exemplos disponíveis, i.e., o modelo acaba ***memorizando*** os exemplos de treinamento.
- O erro para as amostras do conjunto de treinamento é muito baixo.
- Porém, essa aproximação se distancia bastante do modelo gerador.
- Portanto, esses modelos apresentarão ***erros significativamente maiores*** quando forem apresentados a exemplos de validação.
- Efeito conhecido como ***sobreajuste*** ou ***overfitting: flexibilidade*** muito alta e ***grau de generalização*** muito baixo.

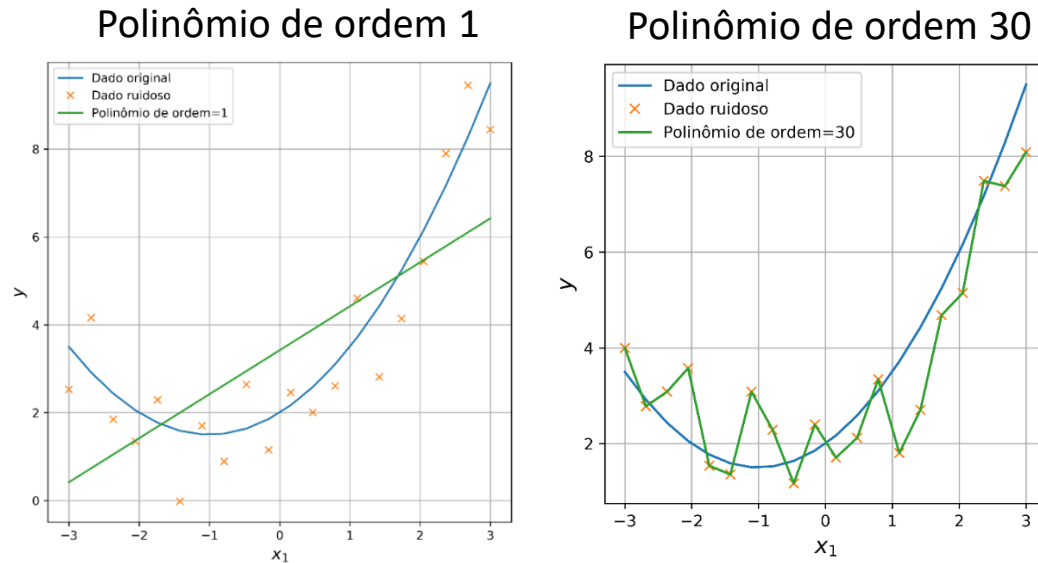
# Resumo sobre subajuste e sobreajuste

- **Subajuste**: situação em que o modelo falha em aproximar o ***mapeamento verdadeiro devido à falta de flexibilidade (ou capacidade)***.
  - Ocorre devido ao modelo não ter graus de liberdade suficientes para a aproximação.
  - O modelo produz ***erros significativos tanto quando apresentado ao próprio conjunto de treinamento quanto a dados inéditos***.
  - Se o modelo está subajustando, mesmo que o número de exemplos aumente indefinidamente, esta situação não vai desaparecer, é necessário ***aumentar a flexibilidade do modelo***, ou seja, no caso da regressão polinomial, sua ordem.

# Resumo sobre subajuste e sobreajuste

- **Sobreajuste**: situação em que o modelo se ajusta tão bem aos exemplos de treinamento que ele aprende até o ruído presente nos mesmos (baixo *erro de treinamento*).
- Porém, o modelo produz erros significativos quando apresentado a dados inéditos (alto erro de *erro de validação*).
  - Ocorre devido ao alto grau de flexibilidade do modelo.
  - Se o modelo está sobreajustando, então é necessário *diminuir sua flexibilidade* ou *aumentar o conjunto de treinamento* até que o erro de validação fique próximo do erro de treinamento.
- Nosso objetivo será encontrar um modelo que apresente uma relação de compromisso entre *flexibilidade* e *capacidade de generalização*.
  - Flexibilidade suficiente para capturar o comportamento geral e generalizar bem.

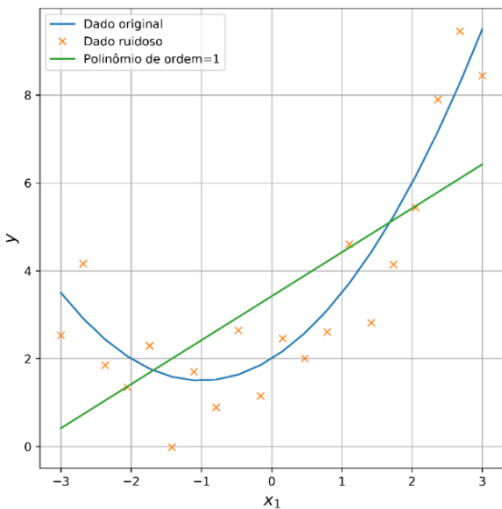
# Observações quanto à regressão polinomial



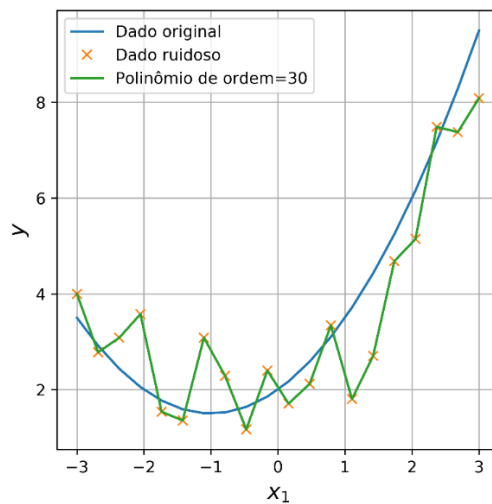
- O objetivo da regressão é encontrar um modelo com **alta capacidade de generalização** a **partir do conjunto de treinamento dado**.
- Porém, vimos que **polinômios de ordem inferior ao da função objetivo não conseguem aproximar o mapeamento verdadeiro**, pois eles **não têm complexidade o suficiente** para capturar a curvatura da função.

# Observações quanto à regressão polinomial

Polinômio de ordem 1



Polinômio de ordem 30

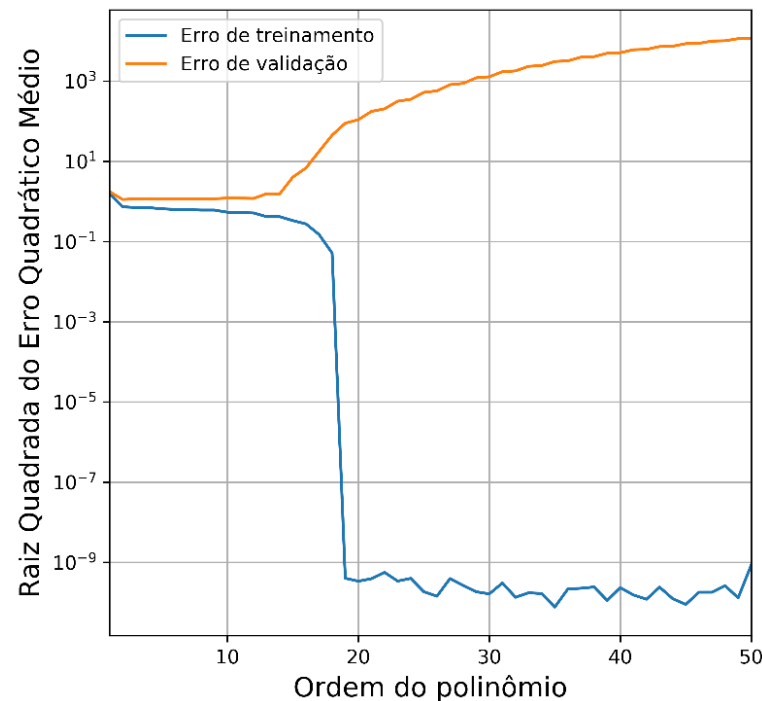


- Notamos também que é **possível encontrar uma aproximação ótima com relação aos exemplos de treinamento** que produz um mapeamento,  $\hat{y} = h(x)$ , que **difere significativamente do mapeamento verdadeiro**.
- Portanto, precisamos encontrar **estratégias** que nos **auxiliem a identificar** um modelo que apresenta um bom **balanço entre flexibilidade e capacidade de generalização**, se distanciando de modelos que **subajustam** ou **sobreajustam**.

# Observações quanto à regressão polinomial

- Uma estratégia simples e bastante usada para selecionar o modelo é ***comparar os erros de treinamento e validação***.
- Para isso, geralmente, dividimos o ***conjunto total de amostras*** em conjuntos de
  - ***Treinamento***: usado para treinar o modelo.
  - ***Validação***: usado para ajustar os hiperparâmetros do modelo (e.g., passo de aprendizagem, ordem).
  - ***Teste***: usado para avaliar a capacidade de generalização do modelo.
- OBS.: Esses conjuntos devem ter amostras diferentes, mas ***devem ser representativos do fenômeno sendo modelado***.

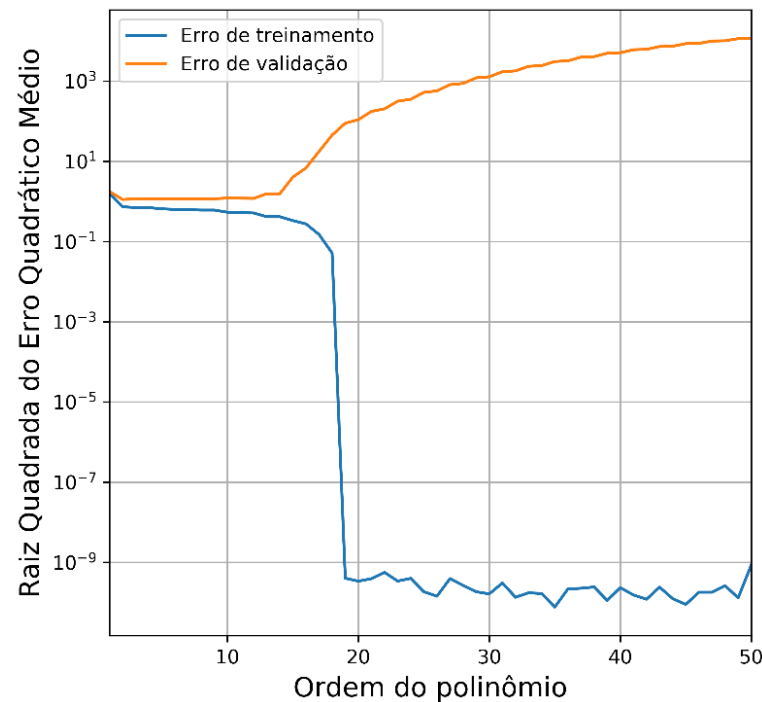
# Erro de treinamento versus erro de validação



- Como vimos, o **erro de treinamento** diminui com o aumento da ordem do polinômio, porém, isso não significa que estamos obtendo um modelo melhor.
- Modelos mais complexos, com erro de treinamento pequeno, não necessariamente são modelos que **generalizam** melhor.
- Esses modelos tendem a apresentar alto **erro de validação**.

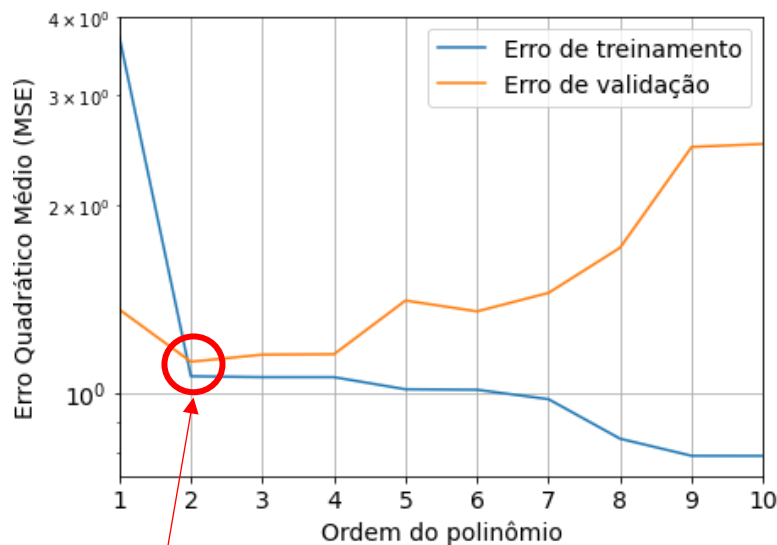


# Erro de treinamento versus erro de validação



- Já modelos com baixa complexidade apresentam alto erro de treinamento e de validação.
- Por outro lado, os ***modelos ideais*** mapeiam valores não vistos durante o treinamento em valores muito próximos aos esperados, i.e., generalizam.
- Portanto, modelos que ***generalizam bem*** apresentam ***erros de validação e de treinamento pequenos e próximos.***

# Flexibilidade e generalização de um modelo



**Modelo ótimo:** relação de compromisso entre flexibilidade e capacidade de generalização.

- A **flexibilidade (ou complexidade)** de um modelo diz respeito à sua **capacidade de aprender as regularidades ou características dos dados do conjunto de treinamento**.
  - Medida através do **erro de treinamento**.
- O **grau de generalização** de um modelo diz respeito a **qualidade da aproximação gerada por ele quando exposto a dados não vistos durante o seu treinamento**.
  - Medido através do **erro de validação**.
- O **modelo ideal** é aquele que encontra uma **relação de compromisso** entre a **flexibilidade** e a **capacidade de generalização**.

# Para casa

- Façam o exercício 5 da lista #3.

# Erros envolvidos no processo de aproximação

- O processo de aproximação (i.e., predição) envolve dois tipos de erros:
  - *Redutível*;
  - *Irredutível*.
- O erro *redutível* é dividido em erros de
  - Variância;
  - Viés;
  - Computação;
  - Amostragem.

# Erros envolvidos no processo de aproximação

- **Erro de *variância* (ou estimação):** é o erro devido à *sensibilidade excessiva do modelo a variações nos dados de treinamento*, ou seja, esse erro *mede o quanto o modelo varia com os dados de treinamento*.
  - Depende do nível de complexidade do modelo.
  - Um modelo com alto grau de complexidade (e.g., polinômio de alto grau) com relação à quantidade de amostras de treinamento apresenta ***alta variância*** e, portanto, se ***sobreajusta***.
  - Em outras palavras, a ***alta variância*** faz com que um modelo ***aprenda o ruído*** presente no conjunto de treinamento, resultando em ***baixo erro de treinamento e alto erro de validação***.

# Erros envolvidos no processo de aproximação

- **Erro de viés (em inglês, bias):** é o *erro devido a suposições erradas feitas sobre os dados*, ou seja, sobre o formato do modelo gerador.
  - Também conhecido como *erro de representação* ou *aproximação*.
  - Depende do nível de **complexidade do modelo** usado para a aproximação.
    - Um modelo com baixa complexidade provavelmente não irá capturar o comportamento do modelo gerador.
  - Modelos com **alto viés** tendem a **subajustar** aos dados de treinamento, perdendo relações importantes entre os atributos e os valores esperados.
  - Ou seja, um **alto valor de viés** leva a **altos erros tanto de treinamento quanto de validação**.
  - Em outras palavras, o *modelo é muito simples* para representar a relação entre as variáveis de entrada e a saída.

# Erros envolvidos no processo de aproximação

- **Erro de computação:** é o erro decorrente do fato de que *nem sempre é possível explorar devidamente o espaço de hipóteses*.
  - Também conhecido como **erro de otimização**.
  - **Motivos possíveis:** mínimos locais, limitação dos recursos computacionais para a busca/treinamento do modelo e uso de representação numérica de precisão finita (i.e., int8 e arredondamento).
    - Por exemplo, erros de arredondamento podem se acumular ao longo do tempo.
- **Erro de amostragem:** ocorre quando o *conjunto de treinamento não é representativo da população de interesse*.
  - Leva a um modelo que *não generaliza bem*.

# Erros envolvidos no processo de aproximação

- **Erro irreduzível:** é o erro devido ao ruído contido nos dados.
  - Também conhecido como ***erro de Bayes***.
  - Como o próprio nome diz, é o erro que ***não pode ser reduzido*** mesmo com modelos ideais (i.e., boa relação de compromisso entre erros de variância e de viés).
  - Por melhor que seja o modelo, os dados geralmente terão uma certa quantidade de ruído que não pode ser removida.



*Como escolhemos a melhor hipótese?*

# Validação cruzada

- A validação cruzada é uma técnica utilizada para **avaliar quantitativamente o desempenho** de um **modelo** e **garantir que ele generalize bem para dados inéditos**, evitando assim problemas de subajuste ou sobreajuste.
- O processo de validação cruzada envolve **dividir o conjunto total de dados em subconjuntos** e realizar **rodadas de treinamento e validação** do modelo com **diferentes combinações desses subconjuntos**.
- A validação cruzada é uma ferramenta importante para **comparar e selecionar modelos** e para **ajustar hiperparâmetros** como, por exemplo, o **passo de aprendizagem**, o **grau do polinômio** da função hipótese, etc.

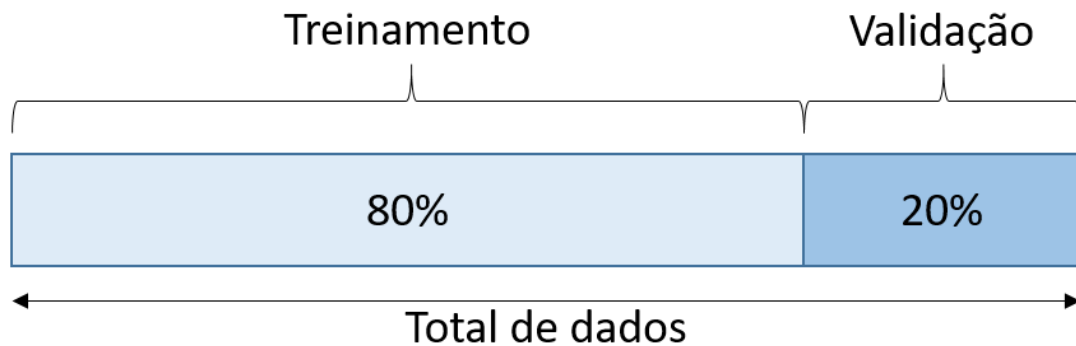
# Validação cruzada

- O **objetivo** da **validação cruzada** é encontrar um **ponto de equilíbrio** entre a **flexibilidade** e a **capacidade de generalização** do modelo.
- Um **modelo equilibrado** é
  - Flexível o suficiente para se ajustar ao comportamento geral dos dados.
  - Capaz de prever saídas próximas às esperadas para exemplos não usados durante seu treinamento.
- A **flexibilidade** de um modelo é **estimada** através do **erro de treinamento** e a **capacidade de generalização** é **estimada** através do **erro de validação** ou **teste**.
  - Erro de treinamento é calculado com os dados usados para o treinamento do modelo.
  - Erro de validação ou teste é calculado com dados inéditos.

# Validação cruzada

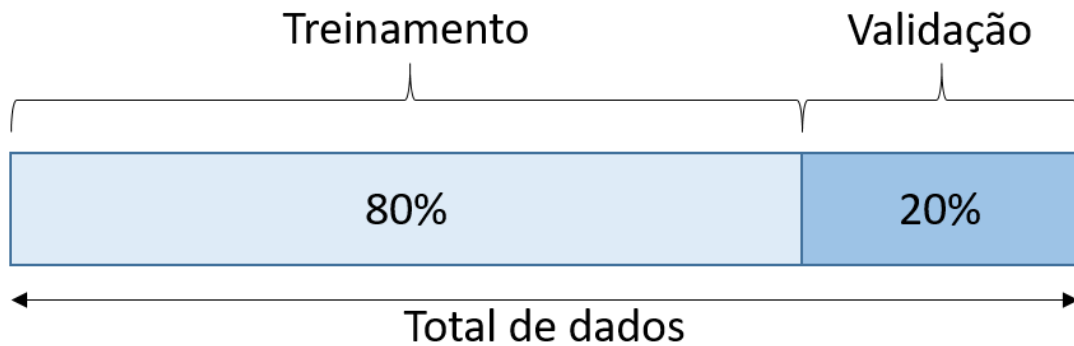
- No caso onde queremos usar a **validação cruzada** para **encontrar o grau ideal da função hipótese polinomial**, o **comportamento destes dois erros** vai nos ajudar a verificar **quais graus fazem o modelo se ajustar demais ou insuficientemente** aos dados de treinamento.
- As estratégias de validação cruzada mais utilizadas e que veremos a seguir são:
  - *Holdout*
  - *k-fold*
  - *Leave-p-out*

# Holdout



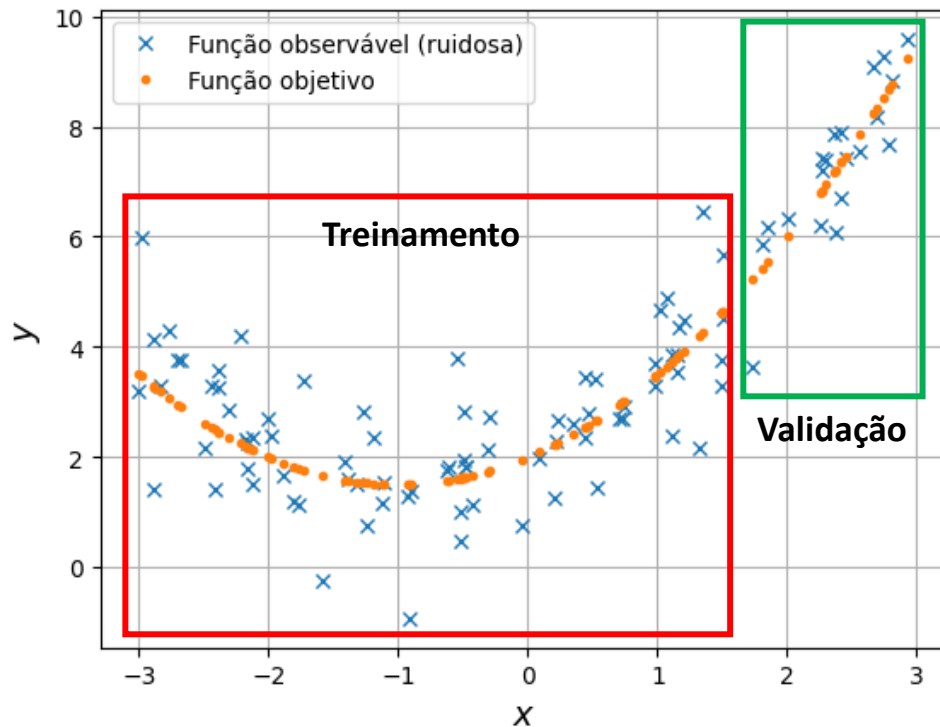
- É a estratégia de validação cruzada *mais simples e rápida*, pois ela *divide* o conjunto total de dados em *apenas dois subconjuntos*, um para *treinamento* e outro para *validação* (ou teste) do modelo.
- Consequentemente, realiza-se *apenas um treinamento e uma validação do modelo*.

# Holdout



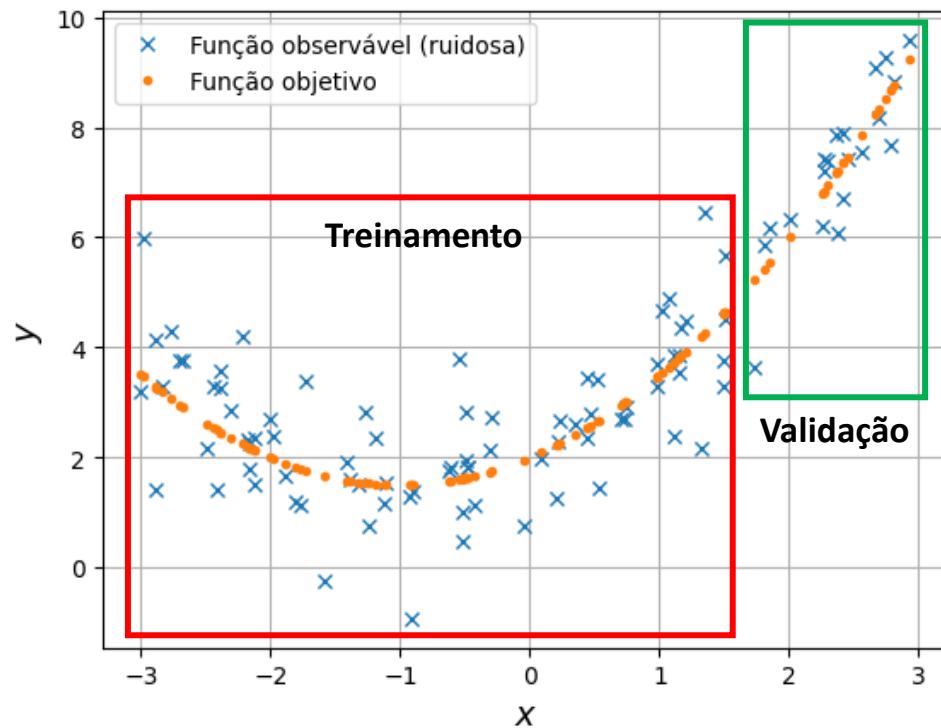
- Em geral, mas é opcional, o conjunto total de dados é ***embaralhado de forma aleatória antes da divisão.***
- Normalmente, divide-se o conjunto total de dados em 70 a 80% para treinamento e 30 a 20% para validação.

# Holdout



- Entretanto, o modelo treinado e validado com esta estratégia pode *apresentar desempenho ruim se a divisão dos dados não for representativa do padrão presente nos dados*.
  - Problema conhecido como *viés de seleção*.
- O desempenho do modelo pode ser muito diferente dependendo da divisão dos dados.

# Holdout

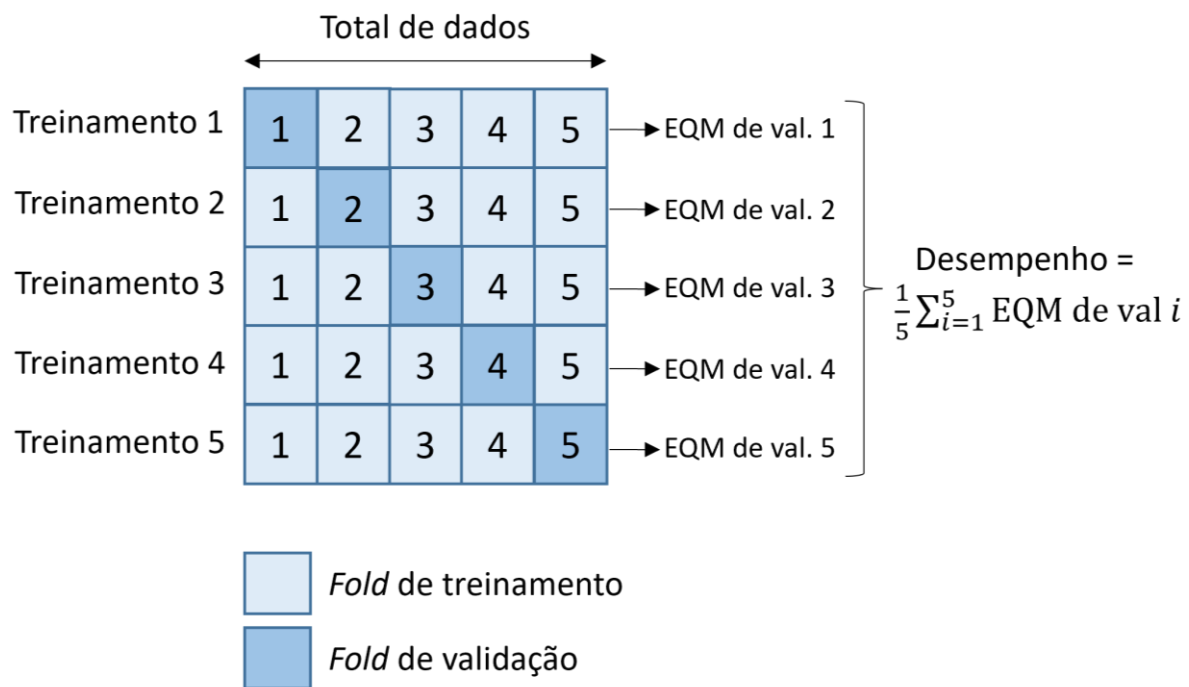


- Além disso, *apenas uma rodada de treinamento e validação* pode *não fornecer* uma *estimativa robusta do desempenho* do modelo.
- Em geral, usa-se o *holdout* quando o *conjunto de dados é muito grande*, o que *minimiza estes problemas*.



# $k$ -fold

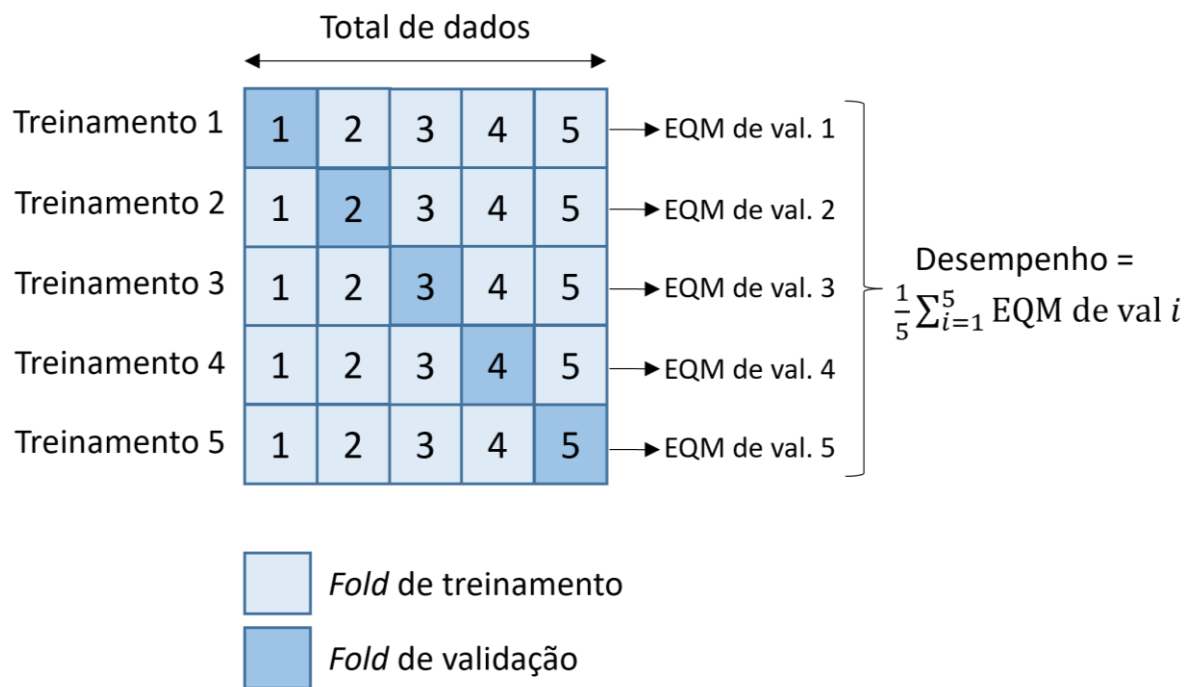
**$k = 5$**



- É uma estratégia mais elaborada do que a do *holdout*.
- A estratégia consiste em embaralhar (opcional) e **dividir o conjunto total de dados em  $k$  partes (ou folds) iguais**.
- O **modelo é treinado  $k$  vezes**, cada vez usando  **$k-1$**  partes como conjunto de treinamento e a parte restante como conjunto de validação.
- O **EQM com o conjunto de validação** é calculado **ao final de cada treinamento**.

# $k$ -fold

$k = 5$



- Ao final dos  $k$  treinamentos, calcula-se a **média** e o **desvio padrão** dos  $k$  EQMs de validação para fornecer uma **avaliação geral do desempenho do modelo**.
- Em geral, utiliza-se  $k = 5$  ou 10.
- Porém,  $k$  deve ser escolhido de forma que os *folds* sejam **representativos do padrão presente nos dados**.

# $k$ -fold

- O  $k$ -fold é a estratégia de validação cruzada mais usada por
  - fornecer **indicações mais claras** sobre desempenho do modelo, devido à média dos  $k$  valores de EQM.
  - **minimizar os possíveis efeitos provocados pelo viés de seleção**, pois o modelo é treinado e validado  $k$  vezes, cada vez com uma divisão diferente dos dados.
    - Isso faz com que a avaliação do modelo se torne menos sensível à divisão dos dados.
- Entretanto, em relação ao *holdout*, o  $k$ -fold tem um tempo de validação maior (cerca de  $k$  vezes), pois deve-se realizar  $k$  treinamentos e validações, enquanto que com o *holdout*, realiza-se apenas um treinamento e validação.

# Leave- $p$ -out

- **Valida** um modelo usando **todas as combinações possíveis** de  $p$  exemplos como **conjunto de validação** e os  $N-p$  restantes como conjunto de treinamento.

- Para um conjunto de dados com  $N$  amostras, essa estratégia produz

$$\binom{N}{p} = \frac{N!}{p! (N - p)!},$$

pares de conjuntos treinamento e validação, portanto, a **complexidade computacional desta estratégia aumenta drasticamente com  $p$** .

- Exemplos para  $N = 100$ :
  - $p = 1 \rightarrow 100$  combinações
  - $p = 2 \rightarrow 4.950$  combinações
  - $p = 5 \rightarrow 75.287.520$  combinações

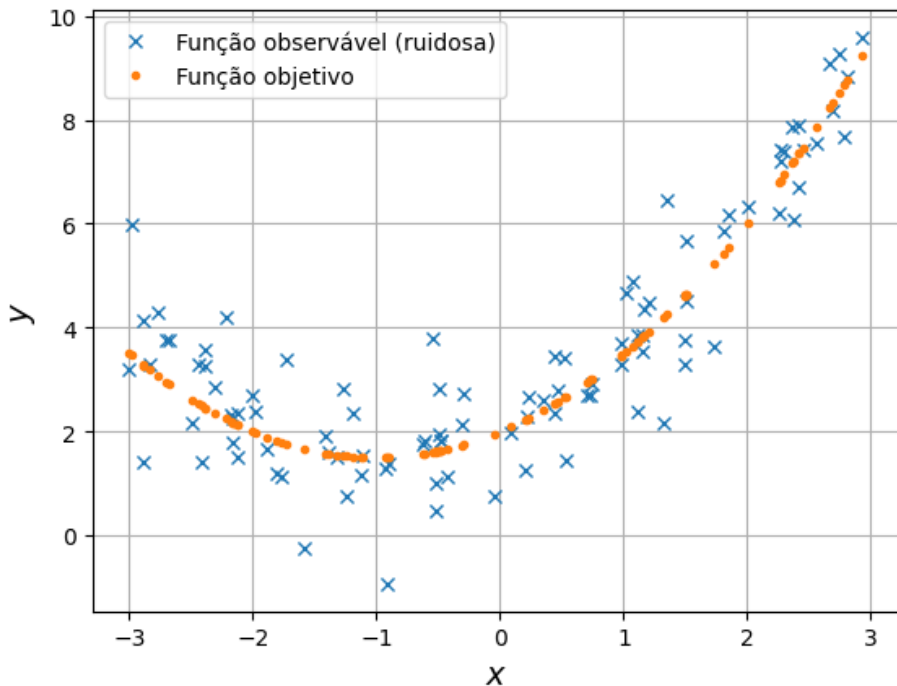
# Leave- $p$ -out

- Fornece *estimativas de erro e desvio padrão mais precisas* do que as abordagens anteriores, pois tem-se *mais etapas de treinamento e validação*.
- **Desvantagem**
  - É uma *estratégia exaustiva*, pois treina e valida o modelo para *todas as combinações possíveis de  $p$  amostras* e, para uma base de dados grande e um valor de  $p$  moderadamente grande, pode se tornar inviável computacionalmente.
- No caso do k-Fold, quando  $k=N$  (i.e., número *folds* igual ao número total de exemplos), então o k-Fold se torna equivalente à estratégia **leave-one-out**, ou seja,  $p = 1$ .

# Qual estratégia utilizar?

- O ***leave-p-out*** dá indicações mais claras de qual ordem usar, ***pois usa um maior número de pares treinamento e validação***, aumentando a confiabilidade da média e do desvio padrão do EQM.
  - Porém, ela é bastante custosa em relação ao tempo necessário para se executá-lo.
  - Portanto, deve-se utilizá-la com ***bases relativamente pequenas e pequenos valores para o parâmetro p***.
- Para bases maiores, o ***k-fold*** é uma opção melhor e mais eficiente do que o ***holdout*** e menos custosa do que o ***leave-p-out***.
- Para bases muito grandes, o ***holdout*** dá boas indicações sobre qual ordem utilizar.
  - A probabilidade dos conjuntos de treinamento e validação obtidos com o ***holdout*** não serem ***representativos*** é inversamente proporcional ao tamanho do conjunto original.

# Validação cruzada para encontrar o grau do polinômio aproximador



- Para exemplificar o uso das estratégias de validação cruzada para encontrar o grau ideal do polinômio aproximador, vamos usar a seguinte função observável

$$y_{noisy} = y + w,$$

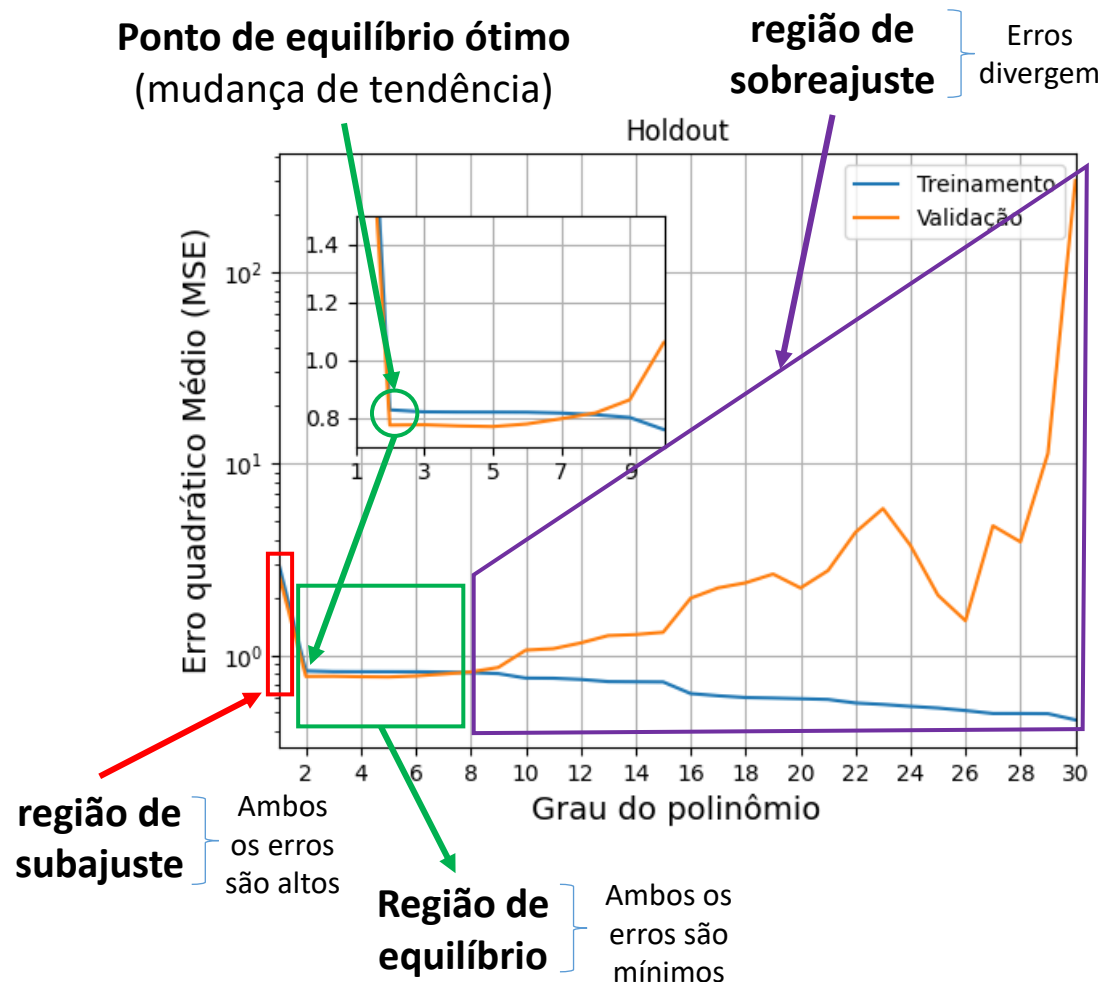
onde  $y$  é a função objetivo e  $w$  é o ruído, o qual tem amostras retiradas de uma distribuição Gaussiana com média zero e variância unitária.

- A função objetivo é um polinômio de segundo grau definido como

$$y = 2 + x + 0.5x^2,$$

onde  $x$  é o atributo, o qual varia entre -3 a 3.

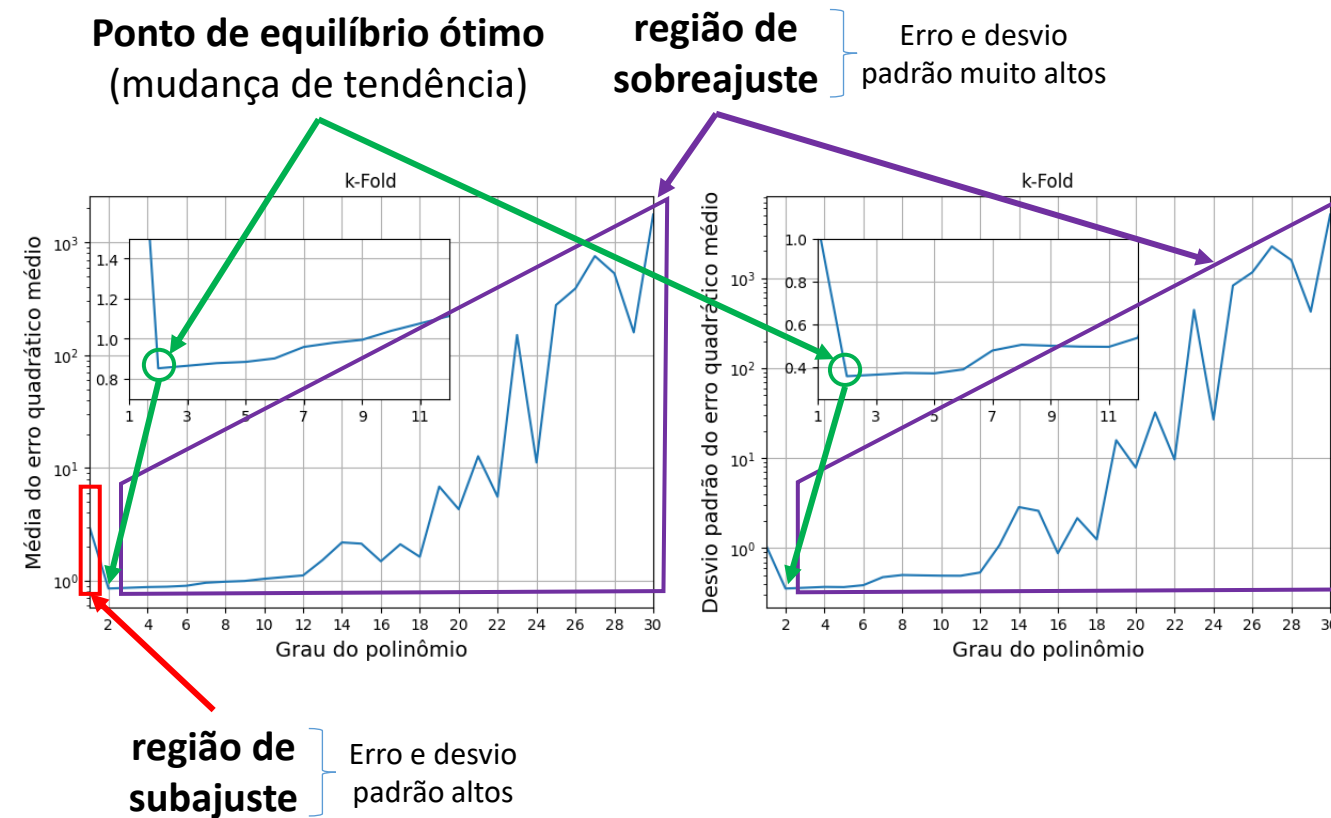
# Usando *holdout* para encontrar o grau do polinômio aproximador



- Divisão: 70% para o conjunto de treinamento e 30% para o conjunto de validação.
- Tempo médio para validação cruzada *holdout* com  $N = 100$  é de  $\approx 150$  [ms].
- Erro de treinamento **diminui** conforme o grau do polinômio aumenta.
- Erro de validação **aumenta** conforme o grau do polinômio aumenta.
- Qual grau escolher?
  - Valor para o qual **ambos os erros sejam mínimos** (balanço entre flexibilidade e capacidade de generalização) e que tenha **menor complexidade computacional**.



# Usando $k$ -fold para encontrar o grau do polinômio aproximador



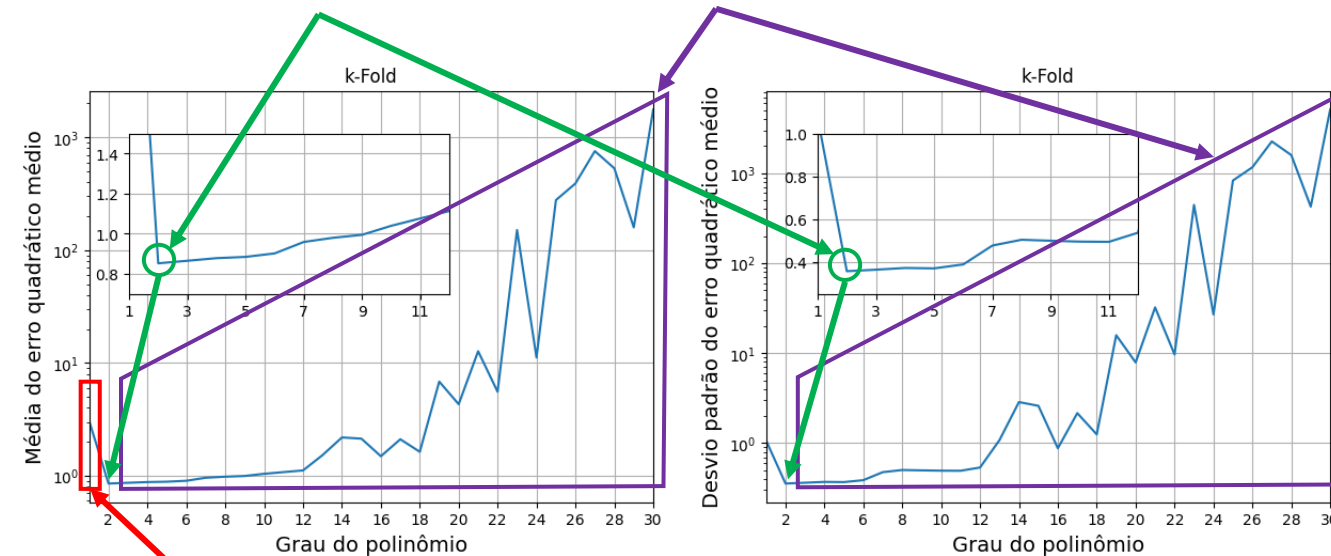
- $k = 10$  folds: 10 treinamentos com 9 folds para treinamento e 1 para validação.
- Os gráficos mostram a média e o desvio padrão do EQM de validação para as 10 etapas de treinamento para cada grau avaliado.
- A média e o desvio padrão do EQM diminuem, passando pelo ponto de equilíbrio, e depois aumentam com o grau do polinômio.
- Qual grau escolher?
  - Valor onde **ambos, média e desvio padrão do EQM, sejam mínimos** e que tenha **menor complexidade computacional**.

\* Tempo médio para validação cruzada  $k$ -fold com  $N = 100$  e  $k = 10$  exemplos é de  $\approx 1.5$  [s].

# Usando $k$ -fold para encontrar o grau do polinômio aproximador

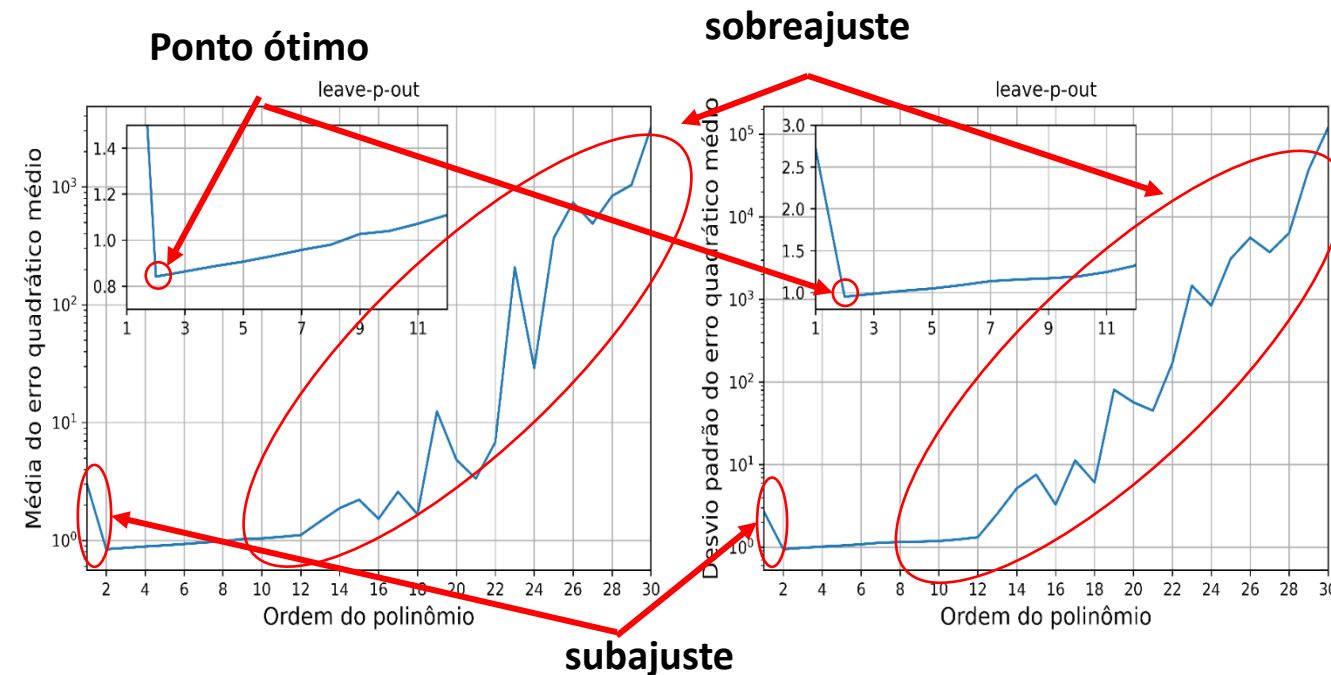
Ponto de equilíbrio ótimo  
(mudança de tendência)

região de  
sobreajuste



- Conforme o modelo se **sobreajusta** aos dados de treinamento, o **desvio padrão do erro de validação aumenta devido à redução de sua capacidade de generalização**.
  - Ou seja, o modelo aprendido se distancia muito do comportamento geral dos dados.
- Modelos **muito flexíveis** (mais do que o necessário) apresentam desvios padrão do erro de treinamento muito baixo e do erro de validação muito alto, indicando **sobreajuste**.
- Modelos pouco flexíveis (menos do que o necessário) têm ambos os desvios padrão dos erros altos, indicando **subajuste**.

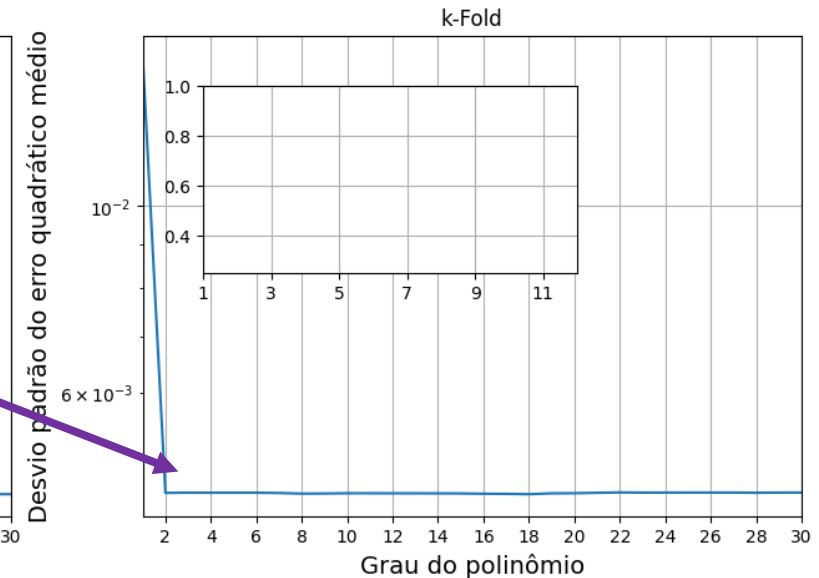
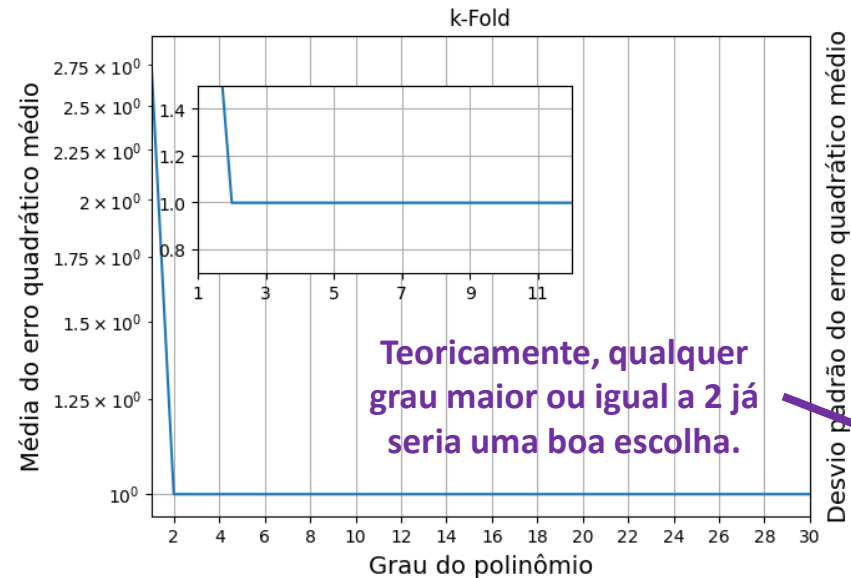
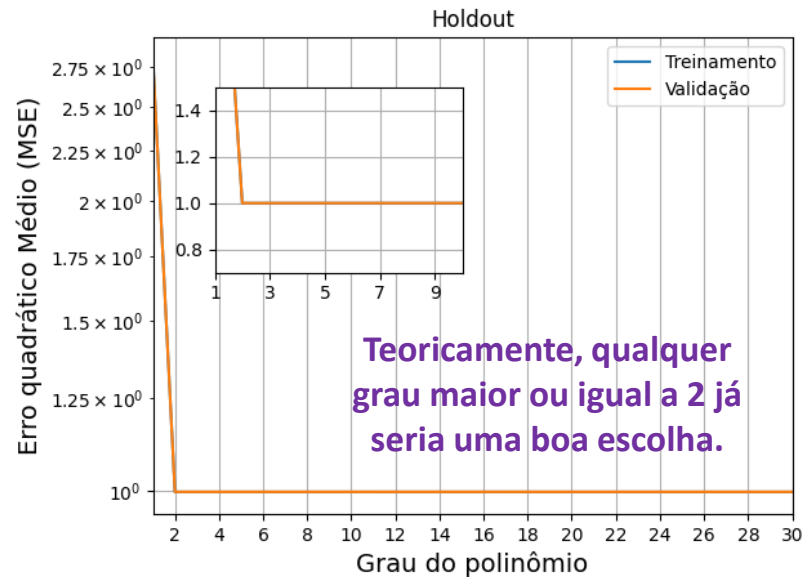
# Usando leave- $p$ -out para encontrar o grau do polinômio aproximador



- Com  $p = 2$  temos 4950 combinações possíveis com 98 exemplos para treinamento e 2 para validação.
- Os gráficos mostram a média e o desvio padrão do EQM de validação para as 4950 etapas de treinamento e validação para cada grau avaliado.
- A média e o desvio padrão do EQM diminuem, passando pelo ponto de equilíbrio, e depois aumentam com o grau do polinômio.
- Qual grau escolher?
  - Valor onde **ambos, média e desvio padrão do EQM, sejam mínimos** e que tenha **menor complexidade computacional**.

\* Tempo médio para validação cruzada leave- $p$ -out com  $N = 100$  e  $p = 2$  é de  $\approx 5$  [m].

# Qual grau escolher quando vários são possíveis?



- Observem as figuras.
- *Qual grau devemos escolher quando os erros (holdout) ou a média e o desvio padrão dos erros (k-fold) são mínimos e praticamente constantes para vários graus de polinômio?*
  - Isso ocorre quando o número de exemplos é muito maior do que a flexibilidade (i.e., grau) do modelo.

# Qual grau escolher quando vários são possíveis?

- A resposta é aplicar a ***navalha de Occam***.
- A ***navalha de Occam*** é um princípio lógico que sugere que, ***entre várias explicações igualmente plausíveis*** para um conjunto de observações, a ***mais simples deve ser preferida***.
  - Ou seja, deve-se ***preferir explicações mais simples às mais complexas***.
- Portanto, usando a ***navalha de Occam*** escolhemos a ***função hipótese polinomial com menor grau*** (i.e., menos complexa), ***mas que se ajusta bem ao comportamento geral dos dados***.
  - Ou seja, escolhemos o modelo mais simples em termos de quantidade de cálculos, mas que possua uma boa capacidade de generalização.

# Para casa

- Façam o exercício 6 da lista #3.
- Leiam o material sobre curvas de aprendizado:
  - Usadas para avaliar:
    - Qual é o *melhor nível de complexidade de um modelo*.
    - O quanto o *modelo se beneficia de mais dados*.
    - A *otimização dos hiperparâmetros* de um modelo.
  - [https://github.com/zz4fap/tp555-ml/blob/main/slides/TP555\\_Curvas de Aprendizado.pdf](https://github.com/zz4fap/tp555-ml/blob/main/slides/TP555_Curvas_de_Aprendizado.pdf)
- Na sequência, façam o exercício 7 da lista #3.



# Regularização: penalizando a complexidade dos modelos

- Anteriormente, nós vimos como escolher o melhor modelo de regressão utilizando ***validação cruzada*** ou ***curvas de aprendizado***.
  - Em ambos os casos, escolhemos o modelo ***menos complexo, mas que generaliza bem***.
- Uma abordagem alternativa é ***minimizar conjuntamente o erro e a complexidade da função hipótese***.
- Essa abordagem combina ***medidas de erro e de complexidade em uma única função de erro***, possibilitando que encontremos uma ***função hipótese*** com
  - a complexidade/flexibilidade ideal,
  - os pesos que minimizam o erro.
- Existem duas técnicas que seguem essa abordagem:
  - **Regularização**: ***penaliza*** funções hipótese muito complexas, ou seja, muito flexíveis.
  - **Early-stopping**: encerra o treinamento de ***algoritmos iterativos*** (e.g., gradiente descendente) quando o erro de validação for o menor possível (chamado de ***regularização temporal***).
- O objetivo das duas técnicas é deixar o modelo ***mais regular*** (i.e., menos complexo) de tal forma que ele ***não se sobreajuste*** ao conjunto de treinamento.

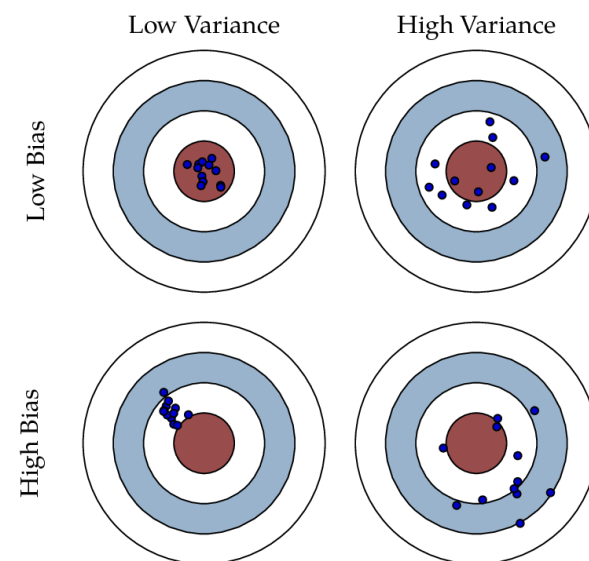


# Regularização: penalizando a complexidade dos modelos

- Um claro sinal de um modelo que se **sobreajustou aos dados de treinamento**, ou seja, um modelo **flexível demais**, são **pesos com magnitudes extremamente altas**.
- Portanto, a ideia por trás da **regularização** é **restringir o aumento da magnitude dos pesos** de forma a **reduzir o risco de sobreajuste**.
  - Para tanto, incorpora-se ao **processo de treinamento** **restrições** proporcionais a alguma **norma** do **vetor de pesos**.
- **Técnicas de regularização reduzem o risco de sobreajuste** do modelo ao conjunto de treinamento, **umentando sua capacidade de generalização**.
  - Quanto menos graus de liberdade o modelo tiver, mais difícil será para ele se **sobreajustar** aos dados de treinamento.
- A **regularização** força o algoritmo de aprendizado não apenas a capturar o **comportamento do modelo gerador**, mas também a **manter os pesos do modelo os menores possíveis**.

# Regularização: penalizando a complexidade dos modelos

- Por **restringir a magnitude dos pesos**, essas técnicas também são conhecidas como técnicas de **shrinkage** (i.e., redução ou encolhimento).
- Essas técnicas permitem diminuir a **variância** do modelo ao custo de introduzir algum **viés**.
  - Encontrar uma **relação de compromisso** entre **viés e variância** permite que **minimizemos o erro total do modelo**.
- **OBS.:** Deve-se aplicar o **escalonamento de atributos** quando se utiliza regularização.
  - A regularização pode ser influenciada pela escala dos atributos e, se eles não estiverem escalonados, alguns atributos podem acabar tendo mais peso do que outros, o que pode prejudicar o desempenho do modelo.
  - Escalonando, os pesos são penalizados de forma justa, pois garantimos que o modelo aprenda a importância relativa de cada atributo.
- As principais técnicas de regularização são: *Rigde*, LASSO e *elastic-net*.



# Ridge regression

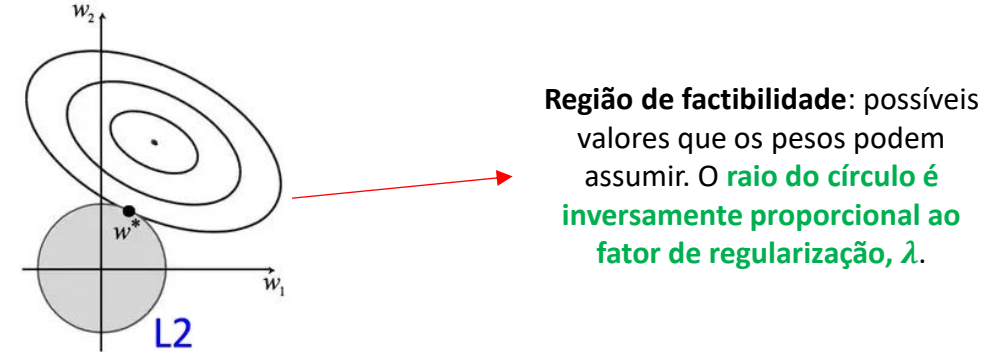
- Ao invés de minimizarmos apenas o **erro quadrático médio**, como fizemos antes, introduzimos um **termo de penalização (ou regularização)** proporcional à **norma Euclidiana** (ou seja, a norma L2) do vetor de pesos:

$$\min_{\mathbf{a} \in \mathbb{R}^{K+1 \times 1}} (\|\mathbf{y} - \Phi \mathbf{a}\|^2 + \lambda \|\mathbf{a}\|_2^2), \quad \text{Se } \|\mathbf{a}\|_2^2 \text{ aumenta, } \lambda \text{ deve diminuir para que o erro seja minimizado.}$$

onde  $\lambda \geq 0$  é o **fator de regularização**,  $\Phi$  é a matriz de atributos,  $\mathbf{a}$  é o vetor de pesos e  $\|\mathbf{a}\|_2^2 = \sum_{i=1}^K a_i^2$  (**OBS.**: somatório inicia em 1 e não em 0).

- O aumento na **flexibilidade** de um modelo é representado pelo aumento da magnitude de seus pesos e, se quisermos minimizar a função de erro, essas magnitudes precisam ser restringidas.
- **OBS.**: o peso  $a_0$  não é considerado no cálculo da **norma L2**, pois a **complexidade** se deve apenas à ordem do modelo.
  - $a_0$  apenas dita o **deslocamento** em relação ao eixo das ordenadas e não influencia na complexidade da **função hipótese**, pois não é multiplicado por nenhum atributo.

# Ridge regression



- Podemos re-escrever o problema de regularização anterior como um **problema de otimização com restrição** da seguinte forma

$$\begin{aligned} \min_{\mathbf{a} \in \mathbb{R}^{K+1 \times 1}} & \|\mathbf{y} - \Phi \mathbf{a}\|^2 \\ \text{s. a. } & \|\mathbf{a}\|_2^2 \leq c, \end{aligned}$$

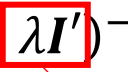
onde  $c$  restringe a magnitude dos pesos (**raio da região de factibilidade**) e é inversamente proporcional à  $\lambda$ .

- Observem que
  - Conforme o valor de  $c$  diminui, menor poderá ser a magnitude dos pesos, até que no limite, quando  $c \rightarrow 0$ , então  $a_i \rightarrow 0, \forall i$ .
  - Por outro lado, conforme  $c$  aumenta, maior poderá ser a magnitude dos pesos, até que no limite, quando  $c \rightarrow \infty$ , então  $a_i$  pode assumir qualquer valor.
  - Portanto, o parâmetro  $c$  (e, consequentemente,  $\lambda$ ) **controla o compromisso entre a redução do erro de aproximação e a limitação da magnitude dos pesos**.

# Ridge Regression

- A equação de **erro regularizado**,  $\|y - \Phi a\|^2 + \lambda \|a\|_2^2$ , **continua sendo quadrática com relação aos pesos**, e, portanto, a **superfície de erro continua sendo convexa**.
- Desta forma, podemos encontrar uma solução de **forma fechada** seguindo o mesmo procedimento que usamos para encontrar a **equação normal**

$$a = (\Phi^T \Phi + \lambda I')^{-1} \Phi^T y, \text{ onde } I' = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

 termo de regularização  
=  
derivada do termo de regularização

*matriz identidade* com o primeiro elemento igual a 0, pois não queremos regularizar o peso de bias.

- **Observações**

- mesmo que a matriz  $\Phi$  não possua **posto completo** (i.e., matriz singular), a inversa na equação acima sempre existirá por conta da adição da **derivada do termo de regularização** à diagonal principal da matriz quadrada  $\Phi^T \Phi$ .
- como a **norma L2** é **diferenciável**, a regressão *Ridge* também pode ser resolvida iterativamente através do **algoritmo do gradiente descendente**.
- o **termo de regularização** deve ser **adicionado apenas à função de erro durante o treinamento**. Depois que o modelo é treinado, a avaliação do desempenho do modelo não utiliza a regularização.

# Ridge regression com gradiente descendente

$$J_e(\mathbf{a}) = \frac{1}{N} \sum_{i=0}^{N-1} (y(i) - \hat{y}(i))^2 = \frac{1}{N} \sum_{i=0}^{N-1} (y(i) - h(\mathbf{x}(i), \mathbf{a}))^2$$

$$J_e(\mathbf{a}) = \frac{1}{N} \|\mathbf{y} - \Phi \mathbf{a}\|^2 + \lambda \|\mathbf{a}\|_2^2$$

$$\frac{\partial J_e(\mathbf{a})}{\partial a_k} = -\frac{2}{N} \sum_{i=0}^{N-1} [y(i) - \hat{y}(i)] x_k(i) + 2\lambda a_k, \quad k = 1, \dots, K$$

$$\frac{\partial J_e(\mathbf{a})}{\partial a_0} = -\frac{2}{N} \sum_{i=0}^{N-1} [y(i) - \hat{y}(i)] x_0(i), \quad \leftarrow \text{Gradiente com relação ao peso de bias}$$

$$\frac{\partial J_e(\mathbf{a})}{\partial \mathbf{a}} = -\frac{2}{N} \Phi^T (\mathbf{y} - \Phi \mathbf{a}) + 2\lambda \mathbf{I}' \mathbf{a}, \quad \leftarrow \text{Equação geral do vetor gradiente em formato matricial.}$$

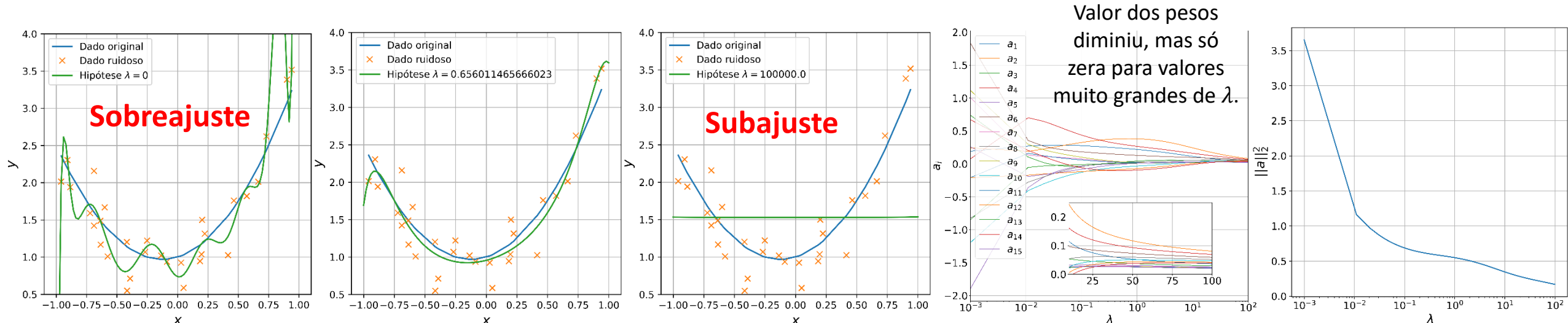
onde  $\mathbf{I}'$  é uma **matriz identidade** de tamanho  $K + 1$ , onde o primeiro elemento é feito igual a 0, pois não queremos regularizar o peso de bias.

- A **equação de atualização dos pesos** é dada por

$$\mathbf{a} = \mathbf{a} + 2\alpha \left[ \frac{1}{N} \Phi^T (\mathbf{y} - \Phi \mathbf{a}) - \lambda \mathbf{I}' \mathbf{a} \right].$$

derivada do termo de regularização

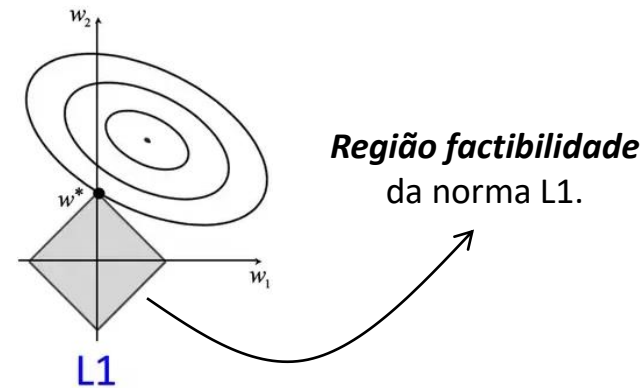
# Ridge Regression: Exemplo



- Função hipótese polinomial de grau 15.
- Modelo treinado com 30 amostras geradas a partir de  $y_{\text{noisy}} = 1 + 0.5x + 2x^2 + w$ , onde  $x \sim U(-1,1)$  e  $w \sim N(0,1)$ .
- Com  $\lambda = 0$ , regressão de Ridge se torna uma regressão polinomial sem regularização.
- Conforme  $\lambda$  aumenta, o modelo não se “contorce” tanto e passa a se ajustar aos dados de treinamento.
- Se  $\lambda$  continuar aumentando, todos os pesos acabarão muito próximos de zero e o resultado será uma linha reta que passa pela **média dos dados de treinamento (i.e., o valor do peso de bias)**.
- **O aumento de  $\lambda$  leva a hipóteses menos complexas.** Isso reduz a variância do modelo, mas aumenta seu bias. Ou seja, ele tende a **subajustar**.
- Conforme  $\lambda$  aumenta, os pesos e a norma L2 do vetor de pesos diminuem.
- Utiliza-se técnicas de **validação cruzada** para encontrar o valor ideal de  $\lambda$ .



# LASSO regression



- A **regressão LASSO** (do inglês *Least Absolute Shrinkage and Selection Operator*) adiciona à função de erro um **termo de penalização** proporcional à **norma L1** do vetor de pesos.

$$\min_{\mathbf{a} \in \mathbb{R}^{K+1 \times 1}} (\|\mathbf{y} - \Phi \mathbf{a}\|^2 + \lambda \|\mathbf{a}\|_1),$$

onde  $\|\mathbf{a}\|_1 = \sum_{i=1}^K |a_i|$  e  $\lambda \geq 0$  é o **fator de regularização**.

- Podemos re-escrever o **problema de regularização** acima como um **problema de otimização com restrição** da seguinte forma

$$\begin{aligned} \min_{\mathbf{a} \in \mathbb{R}^{K+1 \times 1}} \quad & \|\mathbf{y} - \Phi \mathbf{a}\|^2 \\ \text{s. a.} \quad & \|\mathbf{a}\|_1 \leq c, \end{aligned}$$

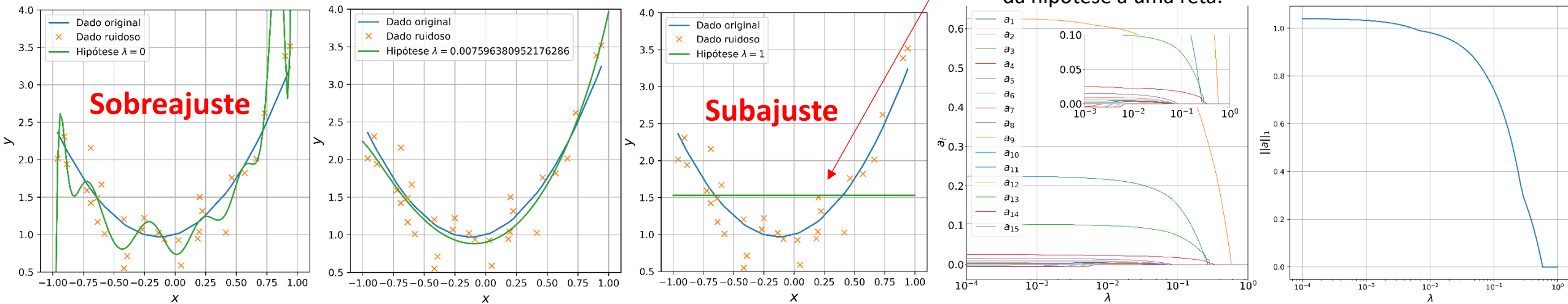
$c$  restringe a **área do quadrado** e é igual a distância da origem até qualquer um dos vértices.

onde  $c$  restringe a magnitude dos pesos e é **inversamente proporcional à  $\lambda$** .

**OBS.:** Assim como na regressão de Ridge,  $a_0$  também não faz parte do cálculo da norma.



# LASSO Regression

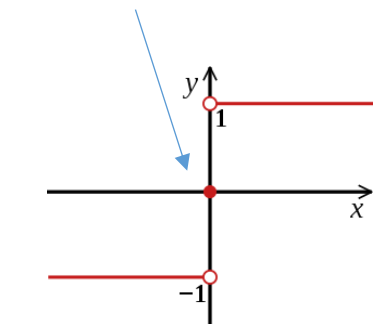


- Mesmas funções geradora e hipótese do exemplo anterior.
- Valores pequenos de  $\lambda$  fazem LASSO se comportar como regressão tradicional e valores muito grandes fazem os pesos serem anulados.
- A regularização com **norma L1** tem como **vantagem** a produção de **modelos esparsos**.
  - Ou seja, vários elementos do vetor de pesos acabam sendo **anulados**, indicando que os atributos correspondentes são irrelevantes para o processo de regressão.
- Isso sugere a ocorrência implícita de um processo de **seleção automática de atributos**, que leva a **modelos** mais **regulares**, ou seja, **menos complexos**.
- **Desvantagem:** como a **norma L1** não possui derivada no ponto  $a_i = 0, \forall i$ , o problema da minimização **não possui solução em forma fechada, mas pode ser implementado com o GD**.
- Utiliza-se técnicas de validação cruzada para encontrar o valor ideal de  $\lambda$ .

# LASSO regression com gradiente descendente

$$J_e(\mathbf{a}) = \frac{1}{N} \|\mathbf{y} - \Phi \mathbf{a}\|^2 + \lambda \|\mathbf{a}\|_1 = \frac{1}{N} \sum_{i=0}^{N-1} (y(i) - \hat{y}(i))^2 + \lambda \sum_{k=1}^K |a_k|$$

indeterminação



$$\frac{\partial |x|}{\partial x} = \text{sign}(x), \text{ para } x \neq 0.$$

$$\frac{\partial J_e(\mathbf{a})}{\partial a_k} = -\frac{2}{N} \sum_{i=0}^{N-1} [y(i) - \hat{y}(i)] x_k(i) + \lambda \text{sign}(a_k), \quad k = 1, \dots, K$$

$$\frac{\partial J_e(\mathbf{a})}{\partial a_0} = -\frac{2}{N} \sum_{i=0}^{N-1} [y(i) - \hat{y}(i)] x_0(i),$$

Gradiente com relação ao peso de bias

$$\frac{\partial J_e(\mathbf{a})}{\partial \mathbf{a}} = -\frac{2}{N} \Phi^T (\mathbf{y} - \Phi \mathbf{a}) + \lambda \mathbf{I}' \text{sign}(\mathbf{a}),$$

Equação geral do vetor gradiente em formato matricial.

onde  $\mathbf{I}'$  é uma **matriz identidade** de tamanho  $K + 1$ , onde o primeiro elemento é feito igual a 0 e a **função sign** ou **signum** é uma função matemática ímpar que extrai o sinal de um número real.

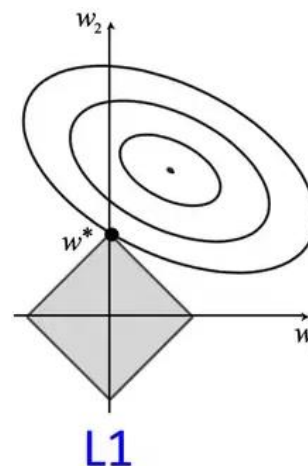
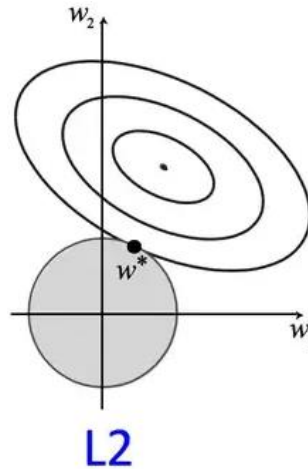
- A **equação de atualização dos pesos** é dada por

$$\mathbf{a} = \mathbf{a} + \alpha \left[ \frac{2}{N} \Phi^T (\mathbf{y} - \Phi \mathbf{a}) - \frac{\lambda}{2} \mathbf{I}' \text{sign}(\mathbf{a}) \right].$$

Implementações práticas consideram que  $\text{sign}(0) = 0$ .

# Vantagem da regressão LASSO sobre Ridge

- A vantagem da regressão LASSO sobre a Ridge está na **forma de quadrado da região de factibilidade** criada pela penalização com a norma L1.
- Como veremos a seguir, **região de factibilidade em forma de diamante** leva à **eliminação de alguns dos pesos** (i.e., os pesos são zerados).
  - A regressão LASSO tende a produzir modelos esparsos.
- Os pesos que são zerados são aqueles **correspondentes aos atributos que são menos relevantes para a predição do modelo** (ou seja, que não contribuem para a predição).
- Isso pode ser útil para **reduzir a complexidade do modelo** e **melhorar sua capacidade de generalização** e **deixá-lo mais interpretável**.
- Além disso, a regressão LASSO também pode ser usada para **seleção de recursos**, onde os **atributos mais relevantes são selecionados automaticamente e os outros descartados**.

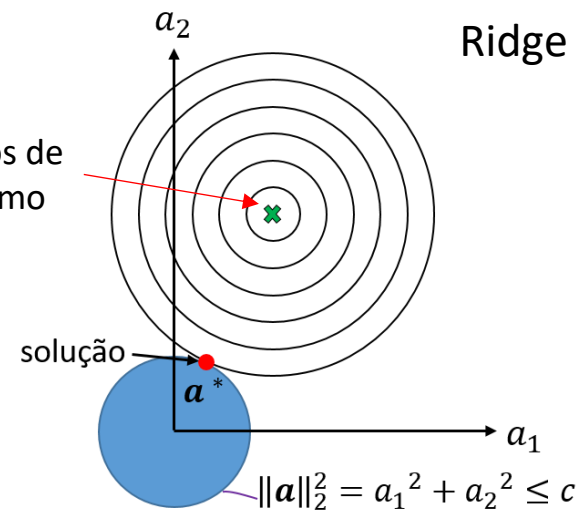
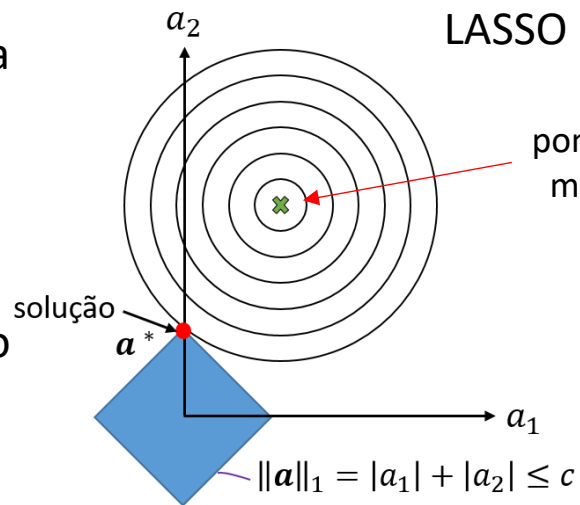


# Vantagem da regressão LASSO sobre Ridge

- A **regressão Ridge** não apresenta esta característica, pois ela **tende a manter os pesos não nulos** (i.e., os pesos nunca são totalmente anulados), produzindo **modelos com pesos pequenos e distribuídos ao longo dos atributos**.
- Podemos também entender a diferença entre as regressões Ridge e LASSO ao compreender que:
  - A norma L2 **penaliza (i.e., encolhe) mais fortemente pesos com magnitudes grandes e penaliza menos fortemente pesos com magnitudes pequenas** (devido ao quadrado na norma L2).
  - Já a norma L1 **penaliza mais agressivamente todos os pesos**, o que significa que a **norma L1 tende a reduzir os pesos para zero de forma mais uniforme** do que a norma L2 (devido a usar apenas o valor absoluto dos pesos).

# Por que a regressão LASSO produz modelos esparsos?

- O quadrado azul representa o conjunto de pontos  $\mathbf{a}$  no espaço de pesos bidimensional que tenham norma L1 menor do que  $c$ .
- A solução deve estar dentro do quadrado, o mais próximo do mínimo.



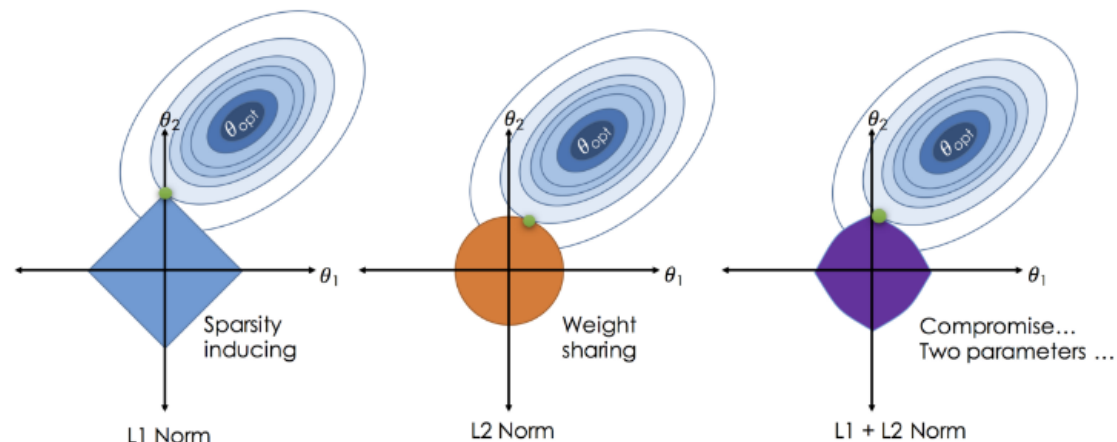
- O círculo azul representa o conjunto de pontos  $\mathbf{a}$  no espaço de pesos bidimensional que tenham norma L2 menor do que  $c$ .
- A solução deve estar dentro do círculo, o mais próximo do mínimo.

- A figura mostra as **curvas de nível** da **função de erro** de um problema de regressão linear com dois pesos ( $a_1$  e  $a_2$ ) e as regiões do **espaço de hipóteses** onde as restrições L1 e L2 são válidas.
- A solução para ambos os métodos corresponde ao ponto, dentro da **região de factibilidade mais próximo do ponto de mínimo** da função de erro.
- É fácil ver que para uma posição arbitrária do mínimo, será comum que um **vértice** (ou cantos) do quadrado seja o ponto mais próximo do ponto de mínimo da função de erro.
- Os **vértices** na **região de factibilidade** da restrição L1 aumentam as chances de alguns pesos assumirem o valor zero, pois são eles que possuem valor igual a zero em alguma das dimensões (i.e., pesos).

# Limitações

- Por não fazer ***seleção de atributos***, a regressão Ridge resulta em um modelo:
  - ***Com baixa interpretabilidade***: não se consegue determinar quais atributos são e não são importantes para a predição.
  - ***Complexo***: por ***manter todos os pesos no modelo, necessita de uma maior quantidade de cálculos*** para realizar predições, se tornando computacionalmente intensiva quando se lida com um grande número de atributos.
- A regressão LASSO:
  - ***Pode não selecionar o melhor atributo quando há forte correlação positiva*** entre dois ou mais atributos.
  - Se o ***fator de regularização não for ideal***, pode levar a um ***modelo muito simples*** que não inclui todos os atributos importantes (devido à seleção de atributos).
  - ***Numericamente instável*** para valores pequenos do ***fator de regularização, principalmente para casos subdeterminados, i.e.,  $K > N$*** .
- Nesses casos, a regressão Elastic-Net é mais indicada, pois minimiza as limitações das duas regressões.

# Elastic-Net



O hiperparâmetro  $\kappa$  dita a relação de compromisso entre as duas regularizações.

- **Elastic-net** é uma regularização que combina as regressões Ridge e LASSO de forma a **resolver as limitações de ambas**.
- Realiza **seleção automática de atributos** e **regularização** simultaneamente.
- Nada mais é do que uma **combinação linear** entre as penalizações baseadas nas normas L1 e L2 do vetor de pesos.

$$\min_{\mathbf{a} \in \mathbb{R}^{K+1 \times 1}} (\|\mathbf{y} - \Phi \mathbf{a}\|^2 + \lambda [\kappa \|\mathbf{a}\|_1 + (1 - \kappa) \|\mathbf{a}\|_2^2]),$$

onde  $\kappa \in [0, 1]$  é o **fator de elasticidade** entre as duas normas, ou seja, estabelece uma **relação de compromisso entre as duas normas**.

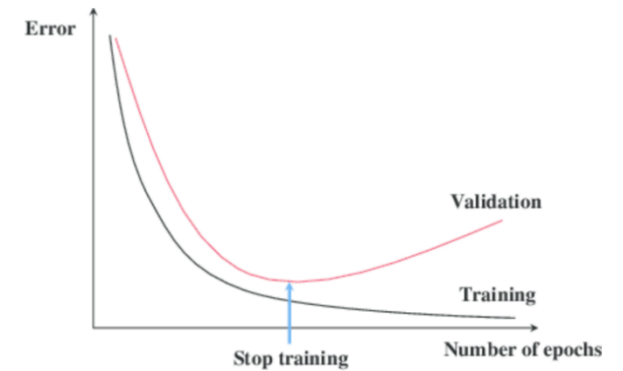
- Quando  $\kappa = 0$ , é equivalente à regressão Ridge, e quando  $\kappa = 1$ , ela é equivalente a regressão LASSO.
- A seleção dos hiperparâmetros  $\kappa$  e  $\lambda$  pode ser feita por meio de **validação cruzada**.

# Quando utilizar cada tipo de regressão?

- **Regressão Ridge:** é um bom começo. No entanto, se você suspeitar que apenas alguns atributos são realmente úteis, você deve preferir LASSO ou Elastic-Net.
- **Regressão LASSO:** boa para *seleção automática de atributos*.
  - No entanto, pode se comportar erráticamente se o *número de atributos,  $K$ , for maior que o número de exemplos de treinamento,  $N$* . Nesse caso, ele *encontra no máximo  $N$  pesos diferentes de zero*, mesmo que os  $K$  atributos sejam relevantes.
  - Se houverem *atributos fortemente correlacionados entre si*, ela *seleciona um deles aleatoriamente e ignora os demais*, o que não é bom para a interpretação do modelo.
  - Nestes casos, deve-se usar a regressão Elastic-Net.
- **Regressão Elastic-Net:** é mais versátil que as anteriores, pois o fator de elasticidade,  $\kappa$ , pode ser ajustado de forma a encontrar uma relação de compromisso entre as normas L1 e L2.
  - Quando há vários atributos correlacionados entre si, a regressão LASSO provavelmente escolherá um deles aleatoriamente, enquanto o Elastic-Net provavelmente *escolherá todos*, facilitando a *interpretação do modelo*.
  - Uma proporção de 50% (i.e.,  $\kappa = 0.5$ ) entre as penalizações L1 e L2 é uma boa escolha inicial para esse parâmetro.



# Early-stopping: Parada antecipada



- O algoritmo do **gradiente descendente** tende a aprender modelos cada vez mais **complexos** à medida que o número de épocas aumenta.
  - Ou seja, ele se **sobreajusta** ao conjunto de treinamento ao longo do tempo.
- Uma forma de se **regularizar** algoritmos de **aprendizado iterativo**, como o **gradiente descendente**, é interromper seu treinamento assim que o **erro de validação** comece a crescer sistematicamente.
- Essa abordagem é chamada de **early-stopping** e pode ser vista como uma **regularização temporal**.
- Assim como as outras abordagens, ela tem o objetivo de evitar o **sobreajuste** de um modelo.
- Ao se **regularizar no tempo**, a complexidade do modelo pode ser controlada, melhorando sua **generalização**.
- Mas como saber quando interromper o treinamento?
  - Ou seja, qual é o critério de parada?

# Early-stopping: critério de parada

- Existem duas estratégias para se definir o critério de parada:
  - Interromper o treinamento quando o valor do **erro de validação** aumentar por  **$P$**  épocas sucessivas, sendo o parâmetro  **$P$**  chamado de ***paciência***.
    - **Problema:** como o erro de validação pode oscilar bastante (e.g., SGD) e apresentar um comportamento pouco previsível, nem sempre é fácil se desenvolver detectores automáticos de mínimos e encerrar o treinamento.
  - Outra estratégia é permitir que o treinamento prossiga por um determinado número de épocas, mas sempre armazenando os pesos associados ao ***menor erro de validação***. Ao final do treinamento, os ***pesos associados ao menor erro de validação são considerados*** para realizar previsões com o modelo.

# Early-stopping: exemplo

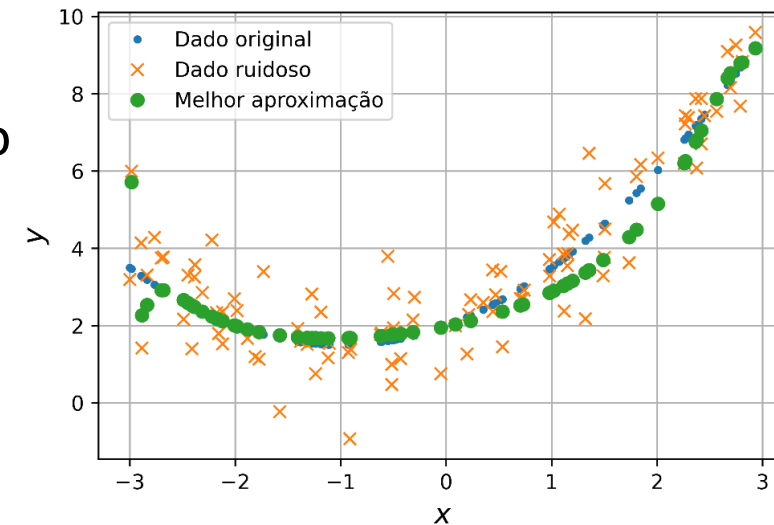
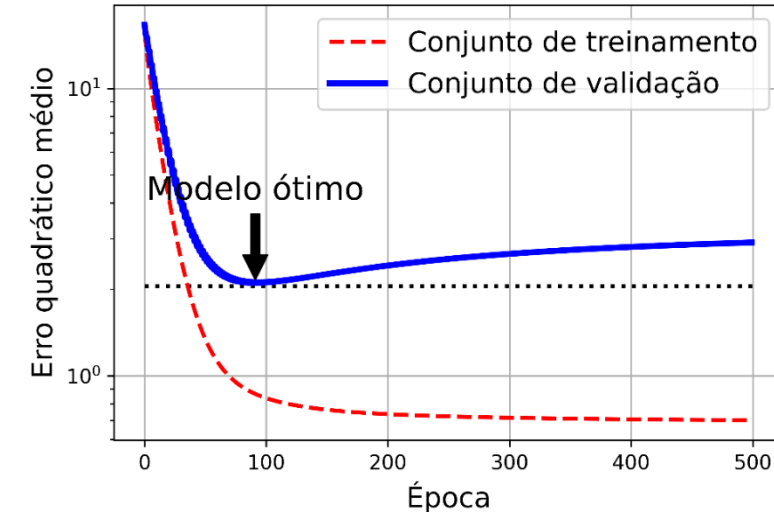
- A figura mostra os erros de treinamento e validação de um modelo de regressão polinomial com grau igual a 90 treinado usando-se o GDE com apenas 70 amostras.

- A função observável é dada por

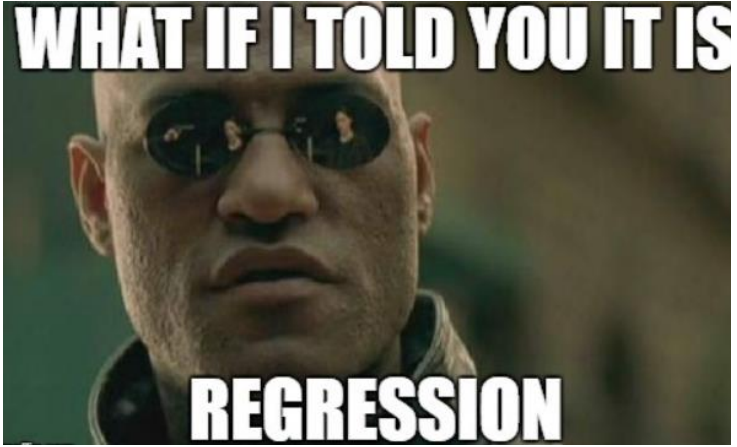
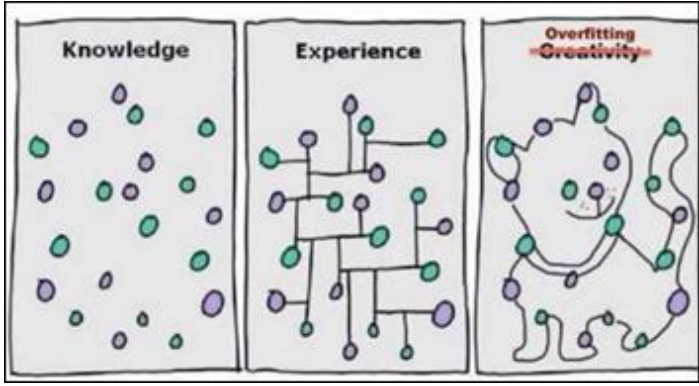
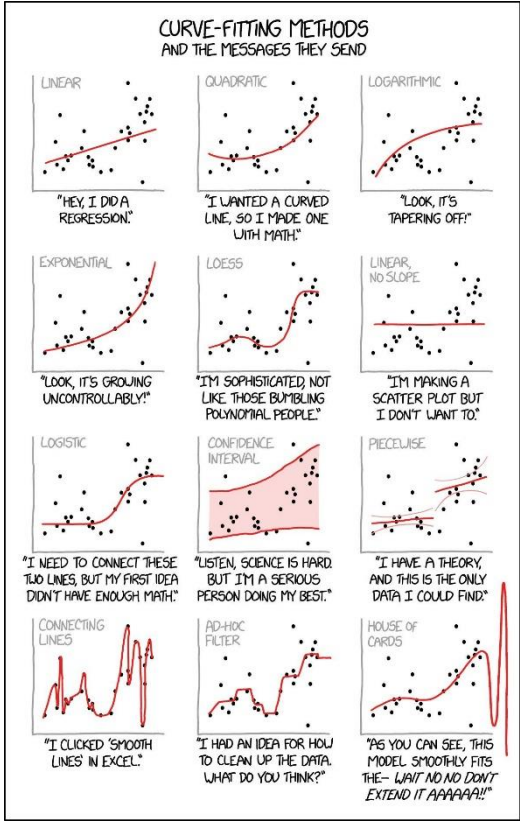
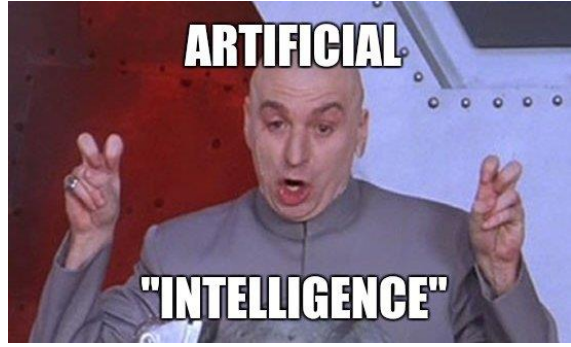
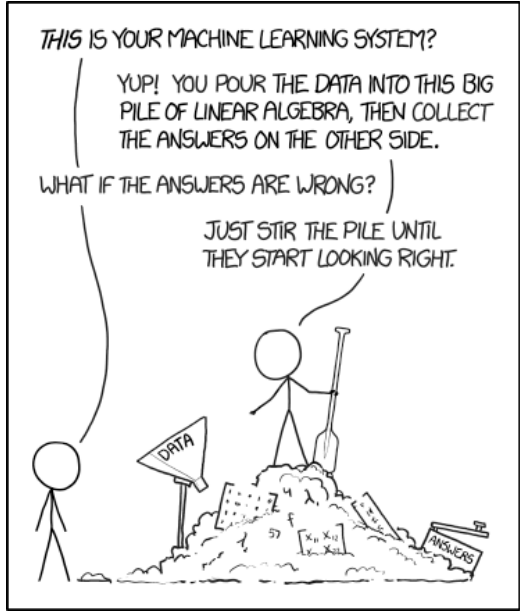
$$y_{\text{noisy}} = 2 + x + 0.5x^2 + w,$$

onde  $x \sim U(-3,3)$  e  $w \sim N(0,1)$ . → Função verdadeira

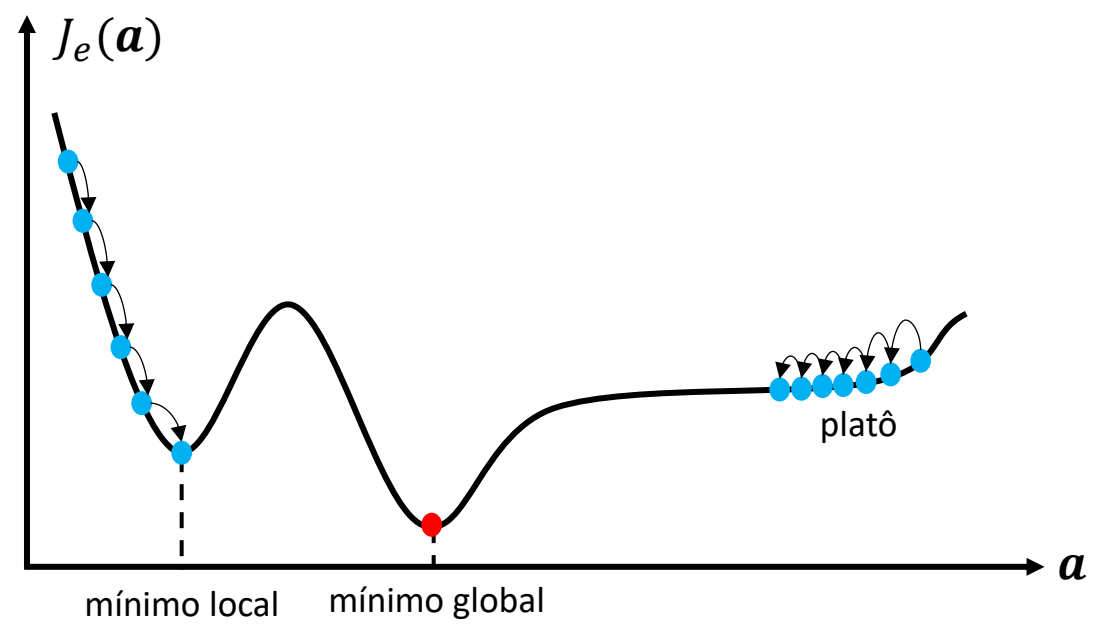
- **A ordem do modelo é muito maior do que a ordem da função verdadeira** (alta flexibilidade), além de ser maior do que o número de amostras de treinamento (sobreajuste).
- À medida que as épocas passam, o algoritmo aprende e seu erro de treinamento diminui, juntamente com o erro de validação.
- No entanto, após aproximadamente 100 épocas, o erro de validação para de diminuir e começa a crescer.
- Isso indica que o modelo começou a **sobreajustar** aos dados de treinamento.
- Com a parada antecipada, usa-se os pesos que resultaram no menor erro de validação ao longo de todo o treinamento, garantindo que o modelo apresente uma boa **generalização**.

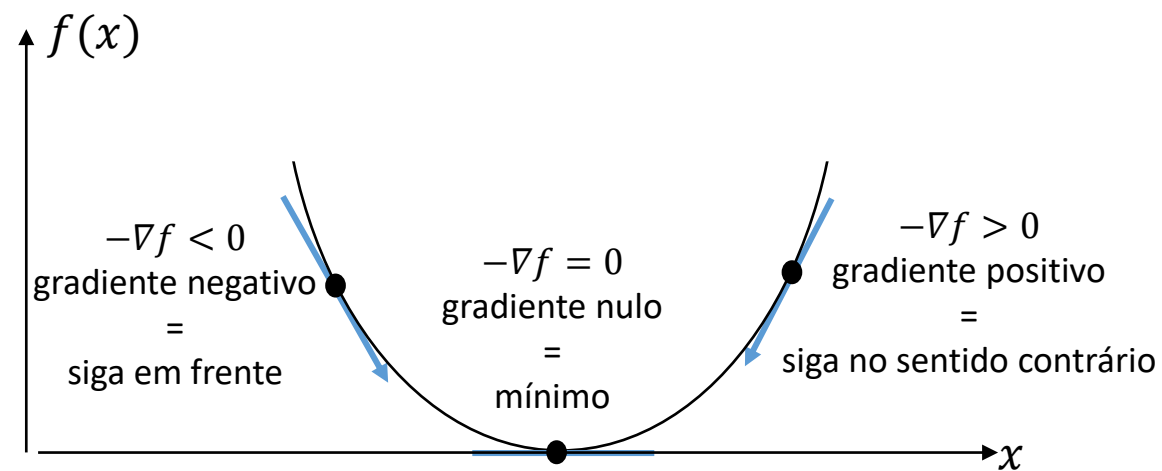
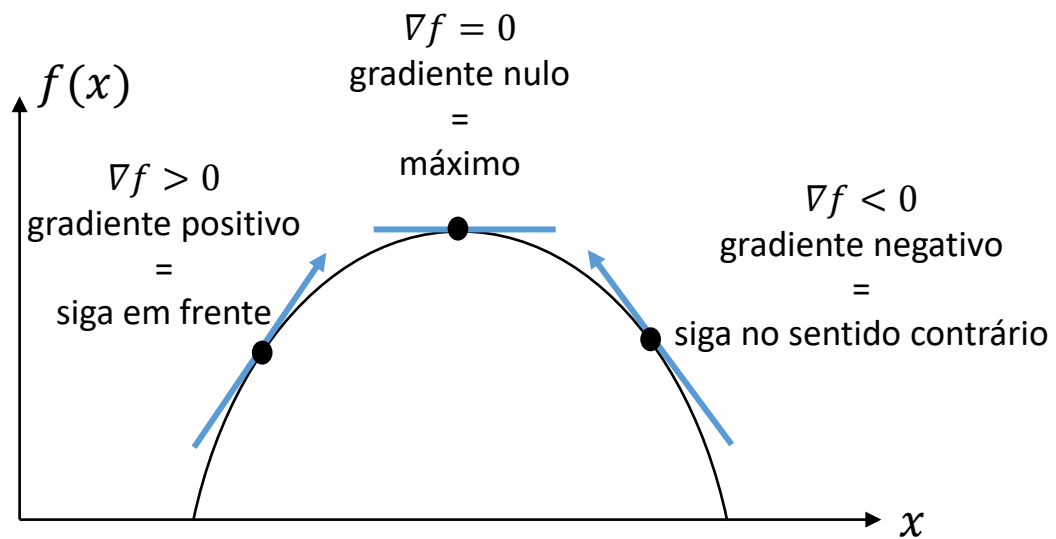


Obrigado!

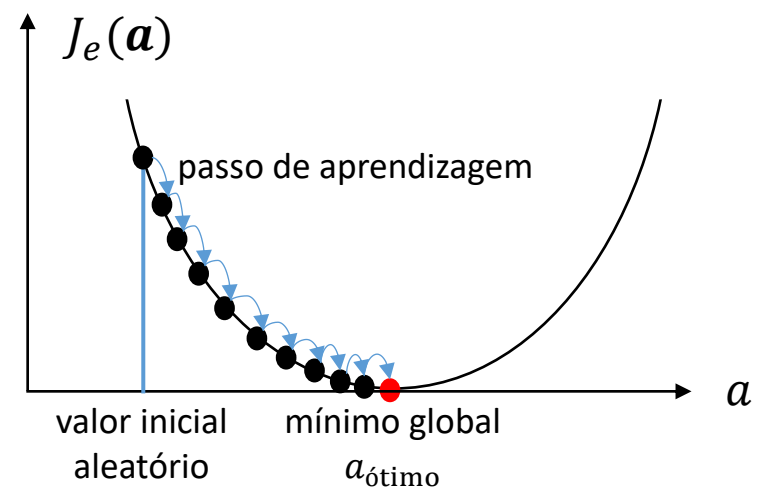


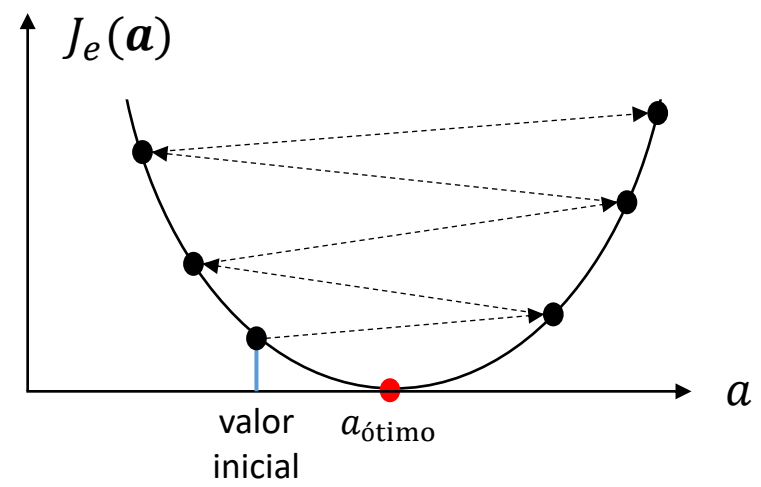
FIGURAS

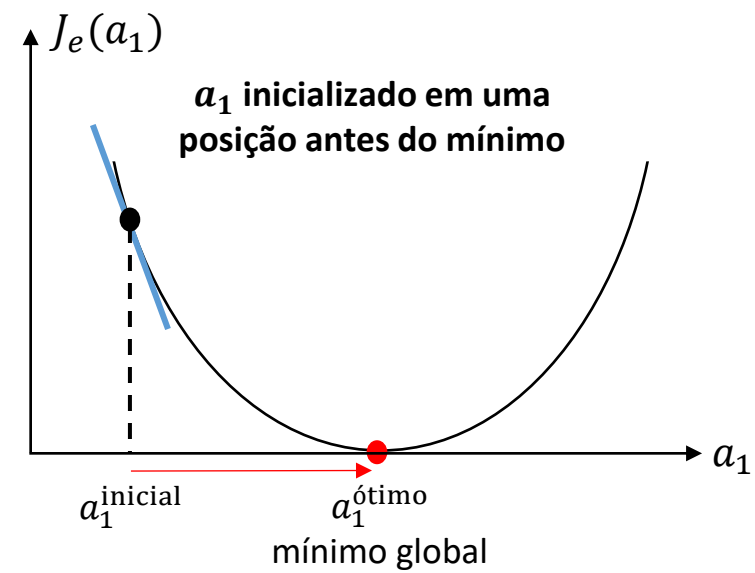




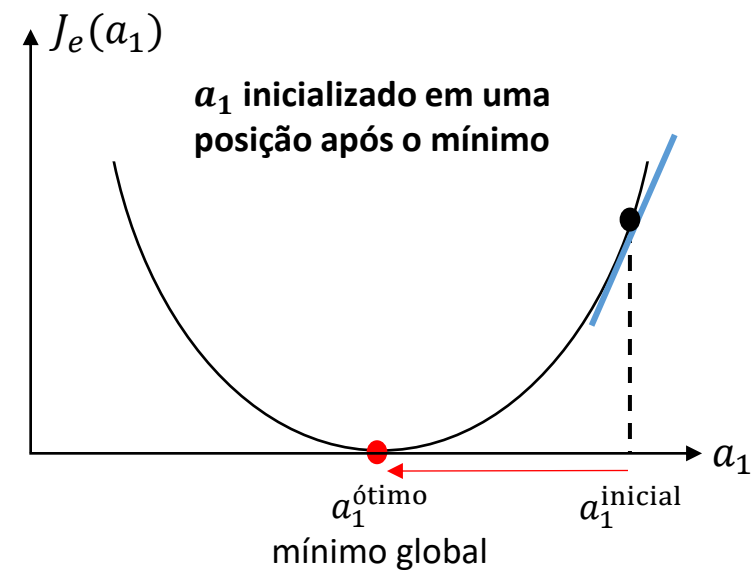




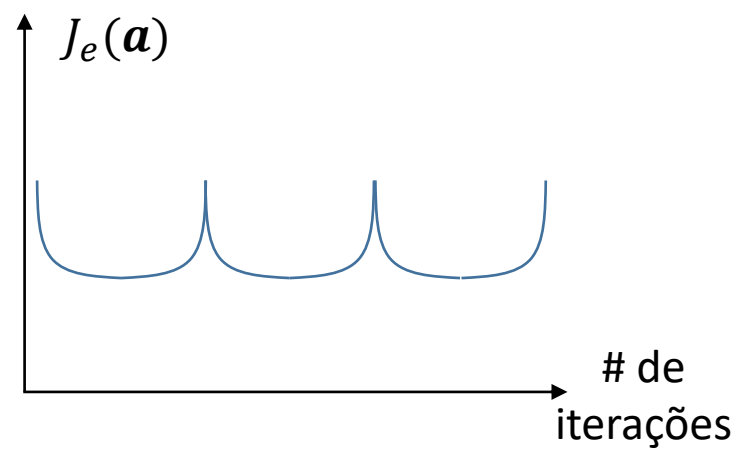
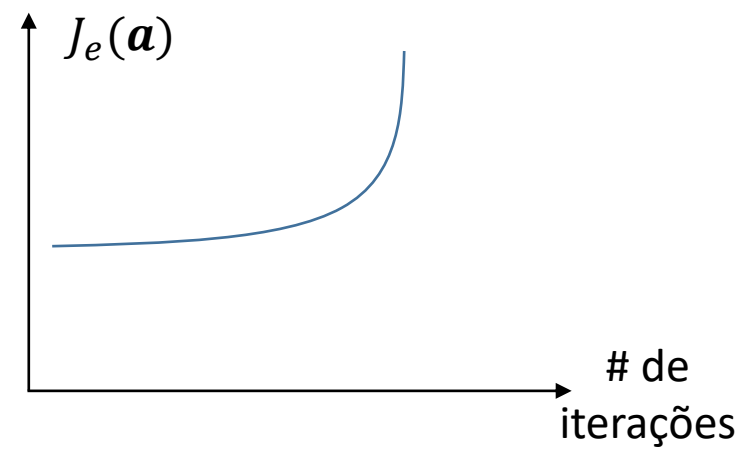
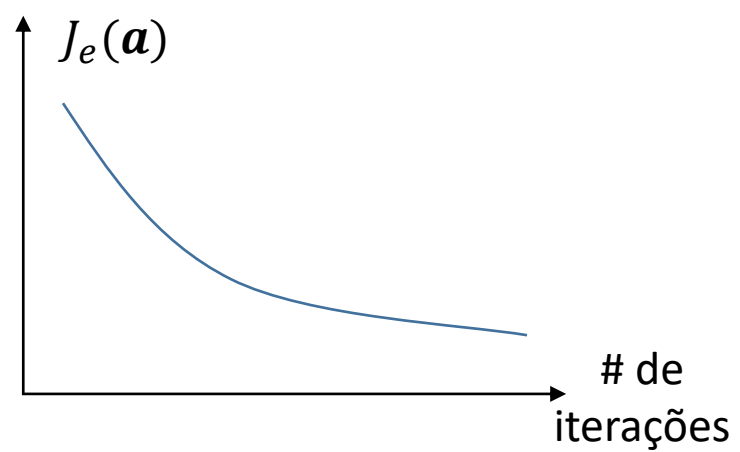
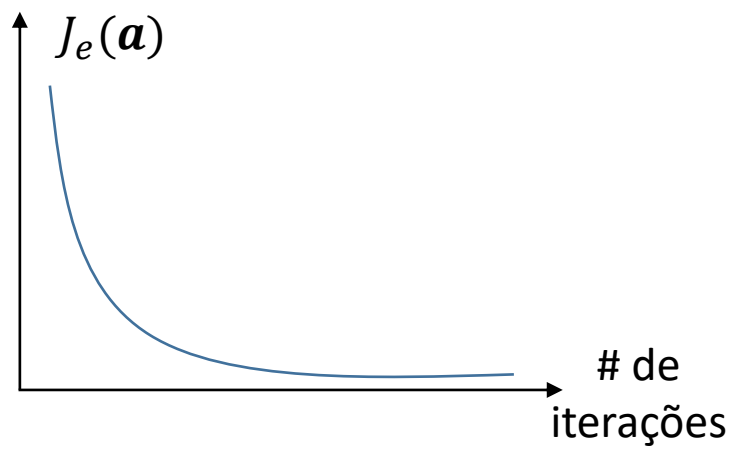


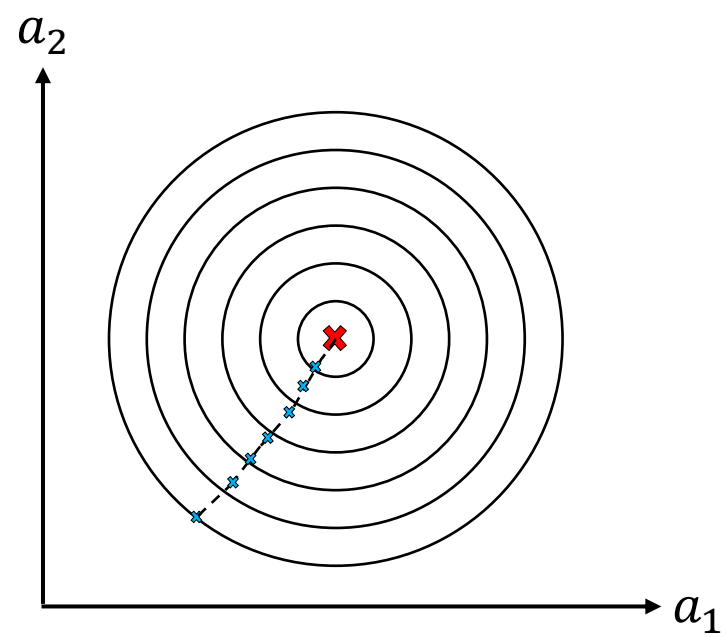
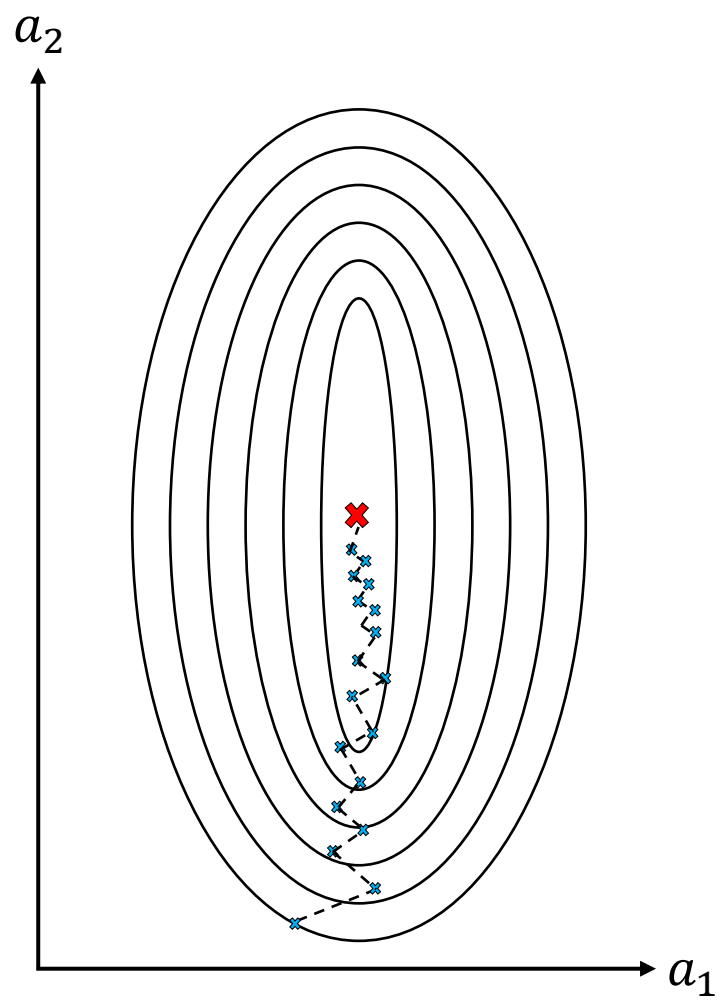


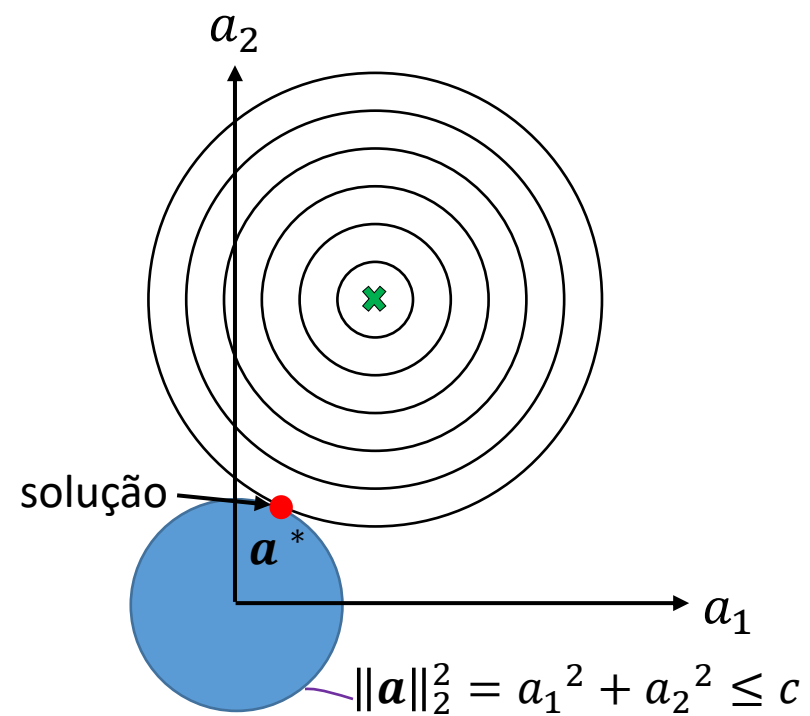
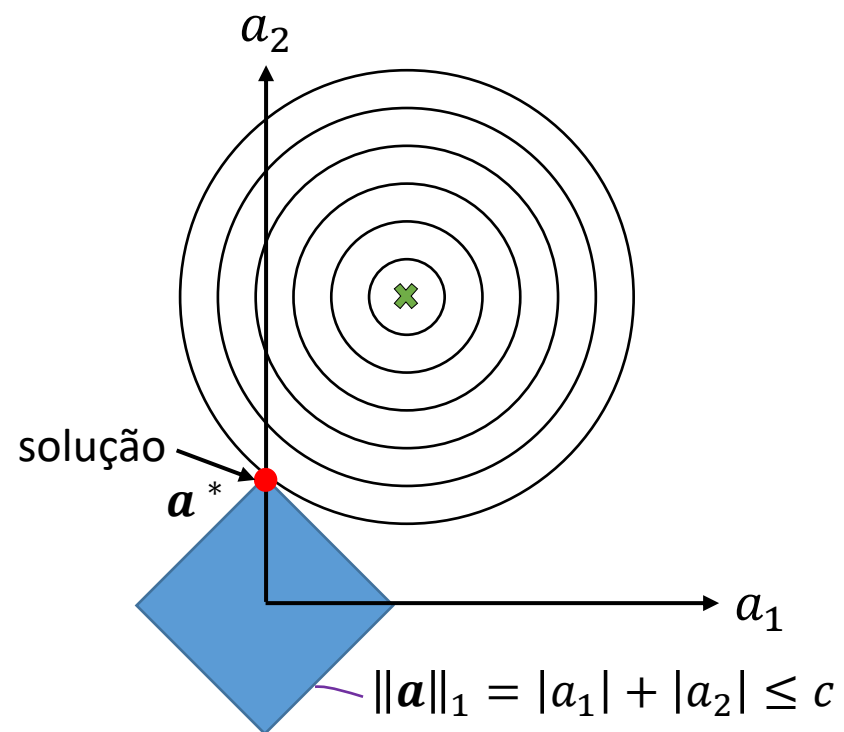
**gradiente negativo:**  $a_1 = a_1^{\text{inicial}} + \alpha \nabla J_e(a_1)$   
 $a_1$  aumenta e se aproxima do mínimo



**gradiente positivo:**  $a_1 = a_1^{\text{inicial}} - \alpha \nabla J_e(a_1)$   
 $a_1$  diminuiu e se aproxima do mínimo







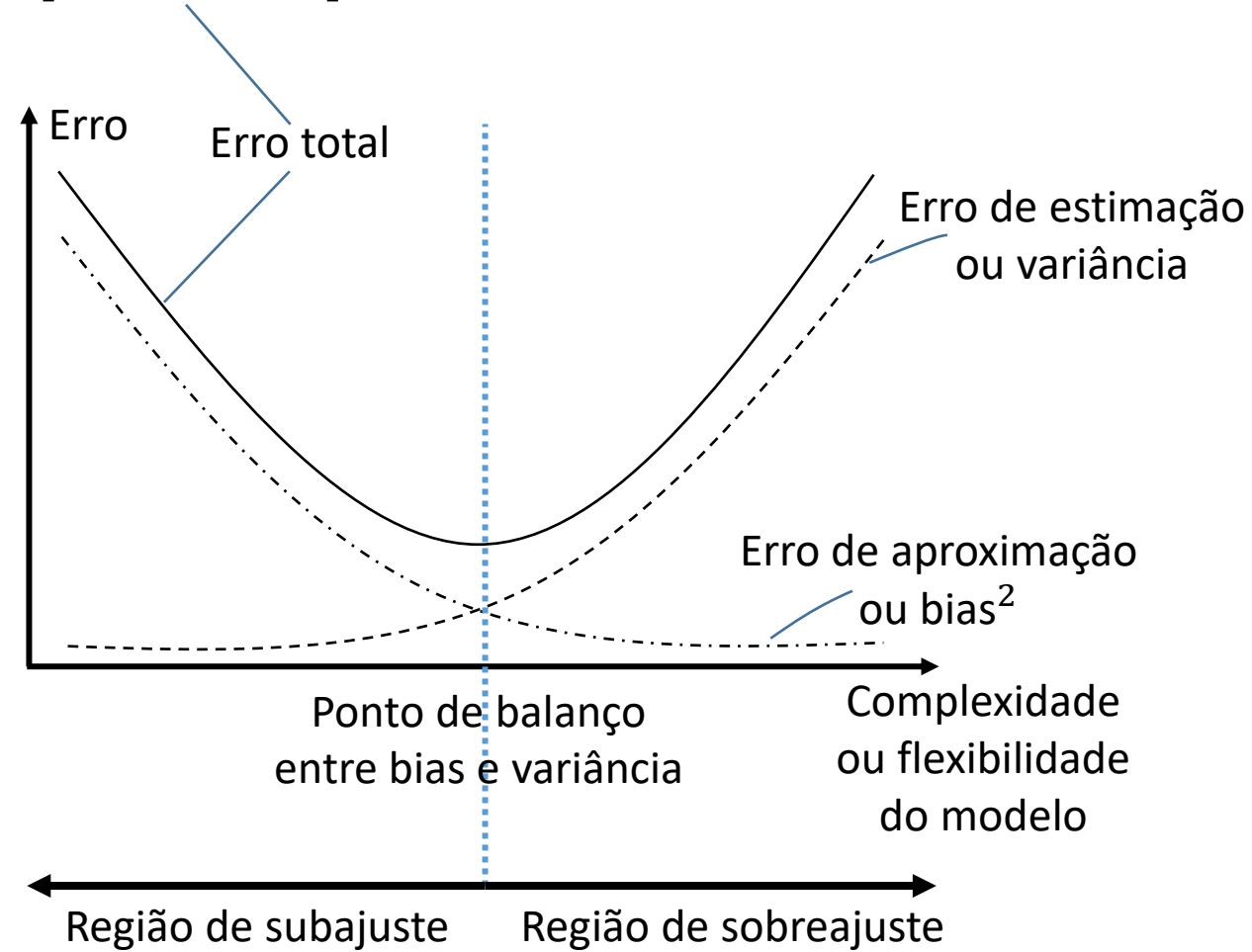
	Total de dados				
Treinamento 1	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Treinamento 2	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Treinamento 3	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Treinamento 4	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Treinamento 5	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5

Treinamento

Validação



$$E \left[ (y_{noisy} - \hat{y})^2 \right] = \text{bias}^2 + \text{variância} + \text{erro irreduzível}$$



$$E \left[ (y_{noisy} - \hat{y})^2 \right] = \text{bias}^2 + \text{variância} + \text{erro irreduzível}$$

