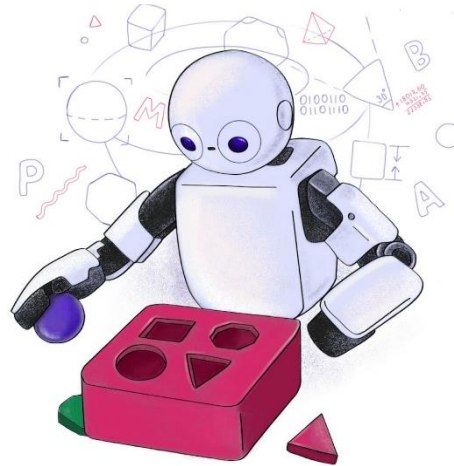


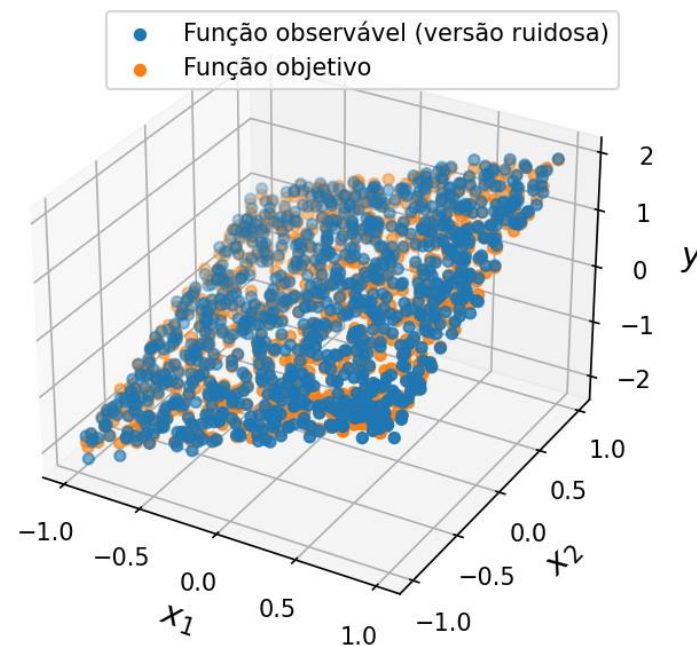
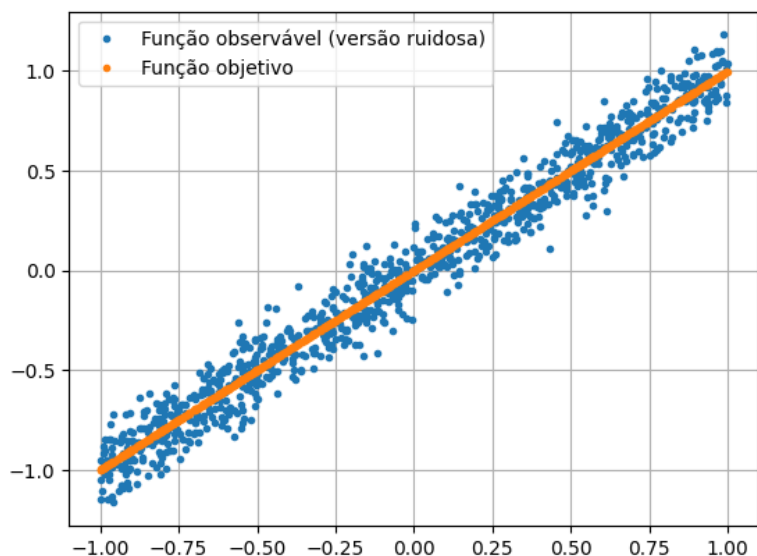
# TP555 - Inteligência Artificial e Machine Learning: *Regressão com Modelos Não- Lineares com Relação aos Atributos*



***Inatel***

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

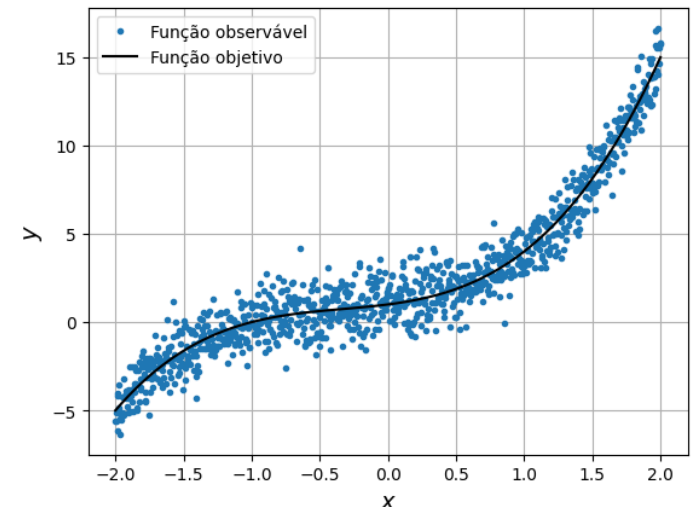
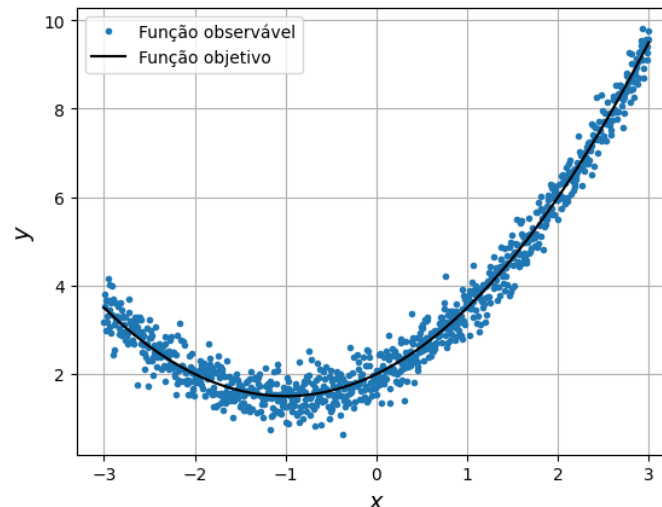
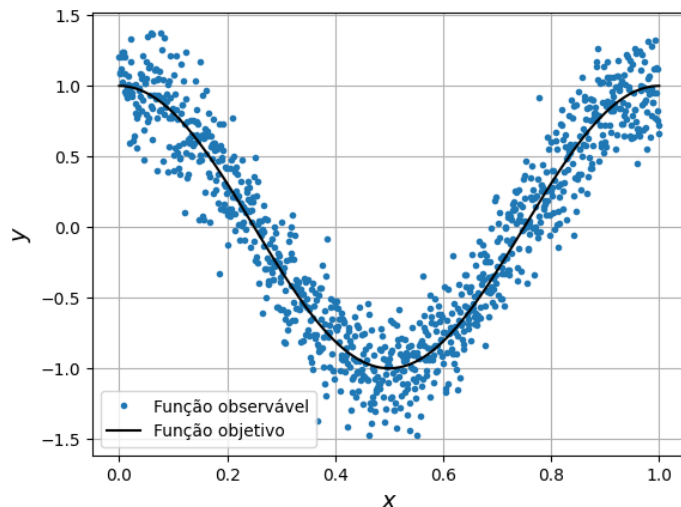
Até agora, usamos *funções hipóteses com formato de hiperplanos*, e.g., retas e planos, para aproximar *mapeamentos lineares* entre os atributos e o valor esperado, mas *e se os mapeamentos forem não lineares?*



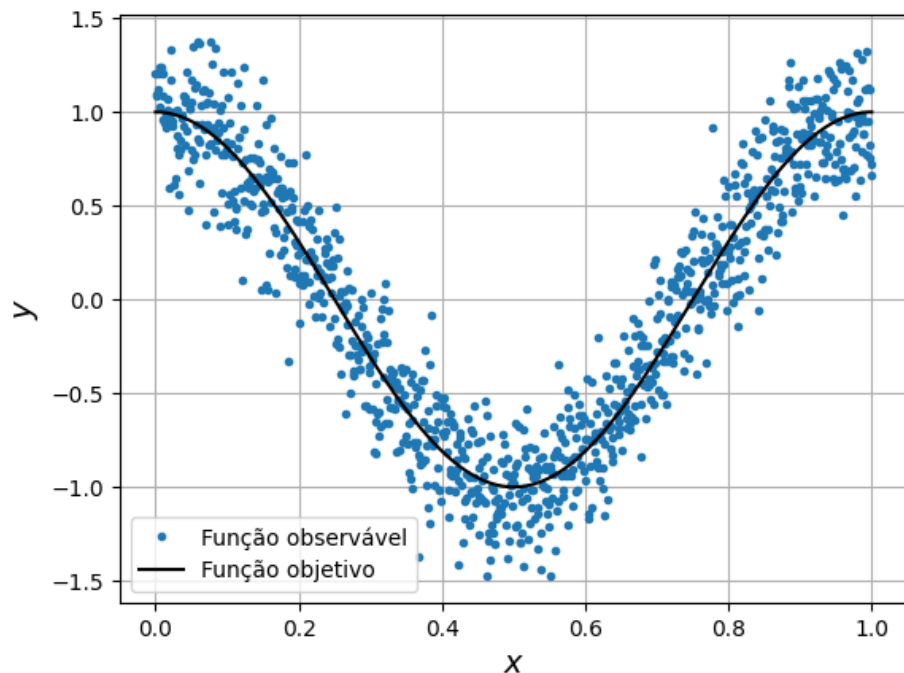
O que podemos fazer quando *hiperplanos não se ajustam bem aos dados*?

# 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



- Toda a teoria que vimos até agora para **regressão linear** também se aplica a **funções hipótese não-lineares** em **relação aos atributos**.
- **Modelos não-lineares** realizam um **mapeamento não-linear das entradas para a saída**, **mas possuem dependência linear com relação aos pesos**.

# 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:

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

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

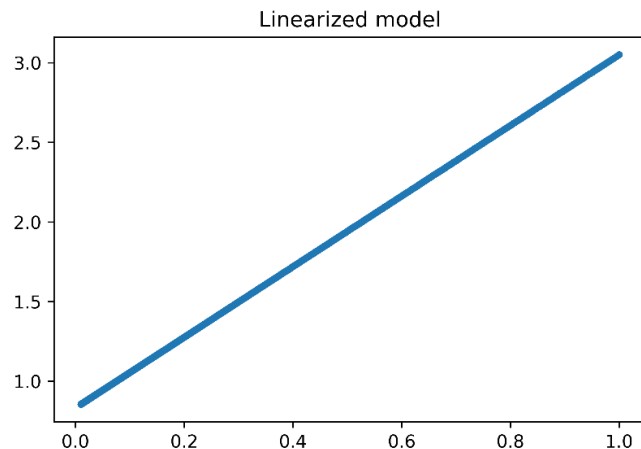
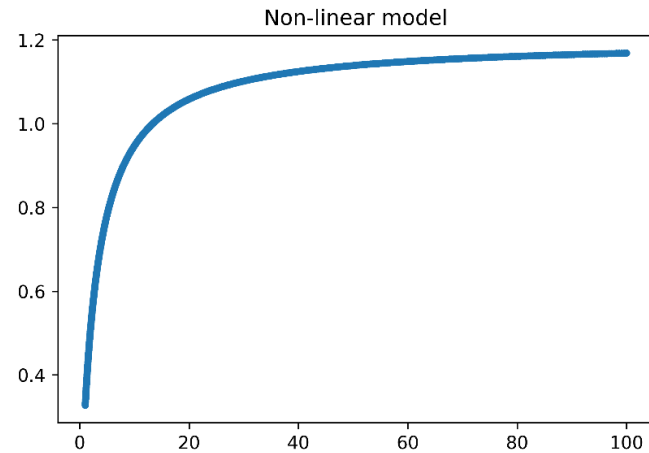
onde a segunda equação pode ser reescrita como

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

Modelos lineares são  
mais fáceis de  
interpretar.

que é linear em relação aos pesos transformados.

# Exemplo: Linearização



```
x1 = logspace(0,2,M)
```

```
# Non-linear model.
```

$$y = (a0 * x1) / (a1 + x1)$$

```
# Linearized model.
```

$$\begin{aligned} y\_lin = 1/y &= (1/a0) + (a1/a0) * (1/x1) \\ &= a0' + a1' * x1' \end{aligned}$$

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





# Regressão Polinomial

- Um caso particular é obtido quando as **funções-base** são compostas por **combinações dos atributos**, por exemplo

$$y(\mathbf{x}) = a_0 + a_1x_1 + a_2x_2 + a_3x_1x_3 + a_4x_1x_2x_3^2 + a_5x_1^3$$

- Para simplificar nossa análise, vamos considerar o modelo de **regressão polinomial em uma variável**

$$h(\mathbf{x}(i)) = \hat{y}(\mathbf{x}(i)) = a_0 + a_1x_1(i) + a_2x_1^2(i) + \dots + a_Mx_1^M(i),$$

onde o vetor de **funções-base** é dado por

$$\Phi(i) = [1, x_1(i), x_1^2(i), \dots, x_1^M(i)]^T.$$

- Por que polinômios?
  - “Qualquer função contínua no intervalo fechado  $[a, b]$  pode ser uniformemente aproximada tão bem quanto desejado por um polinômio”, **Teorema da aproximação de Weierstrass**.
- Entretanto, a presença de **outliers** nos dados de treinamento impacta o desempenho do modelo.

# Regressão Polinomial: Exemplo

- Geramos 30 exemplos do seguinte ***mapeamento verdadeiro***:

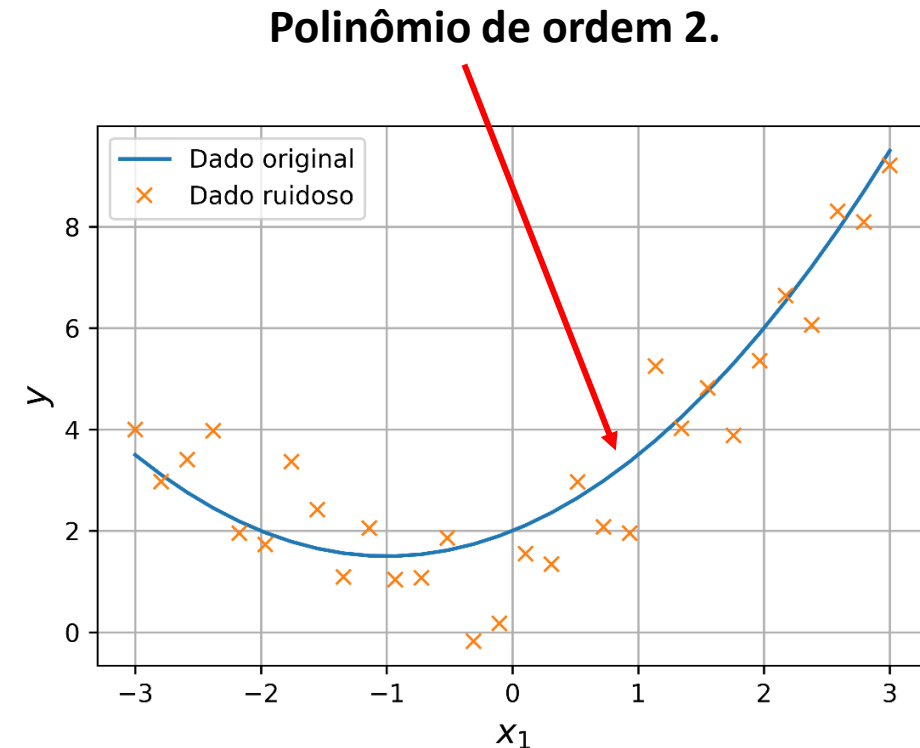
$$y(x_1(n)) = 2 + x_1(n) + 0.5x_1^2(n),$$

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  $w(n) \sim N(0, 1)$ .

- Agora surge uma dúvida, ***e se não soubéssemos a ordem por trás do modelo gerador (i.e., seu formato), qual ordem deveríamos utilizar?***



# Regressão Polinomial: Exemplo com SciKit-Learn

```
# Import all the necessary libraries.
```

```
import numpy as np
```

```
from sklearn.preprocessing import PolynomialFeatures
```

```
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.linear_model import LinearRegression
```

```
from sklearn.pipeline import Pipeline
```

```
# Size of the example set.
```

```
M = 100
```

```
# Create target function.
```

```
x1 = np.linspace(-3, 3, M).reshape(M, 1)
```

```
y = 2 + x1 + 0.5*x1**2;
```

```
y_noisy = y + np.random.randn(M, 1)
```

```
# Instantiate a degree 2 polynomial.
```

```
poly_features = PolynomialFeatures(degree=2, include_bias=True)
```

```
# Instantiate a scaler.
```

```
std_scaler = StandardScaler()
```

```
# Instantiate a linear regressor.
```

```
lin_reg = LinearRegression()
```

```
# Create a pipeline of actions.
```

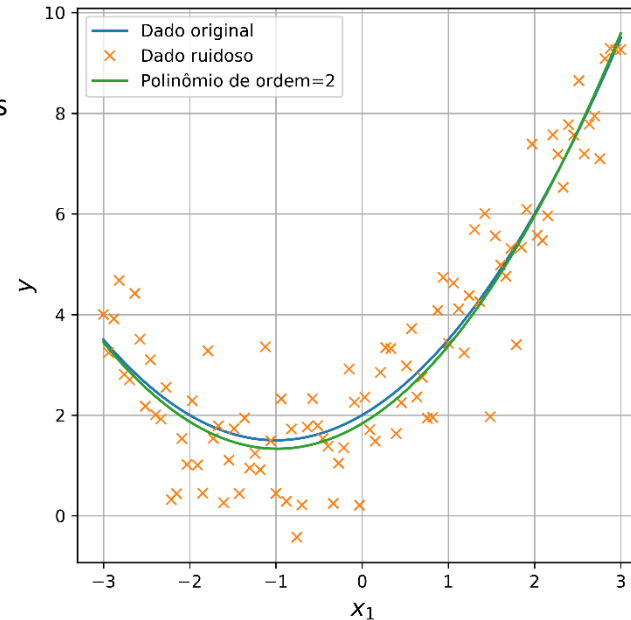
```
polynomial_regression = Pipeline([("poly_features", poly_features), ("std_scaler", std_scaler), ("lin_reg", lin_reg),])
```

```
# Perform polynomial regression.
```

```
polynomial_regression.fit(x1, y_noisy)
```

```
# Use the trained model for prediction of the training set.
```

```
y_train_predict = polynomial_regression.predict(x1)
```



grau do polinômio

## Observações:

- A classe **PolynomialFeatures** cria a matriz de atributos com as combinações polinomiais dos atributos.
- A classe **Pipeline** cria um objeto que sequencializa a aplicação de **transformadores** e **estimadores** (e.g., PolynomialFeatures e StandardScaler, LinearRegression) aos dados e ao final treina e realiza previsões.
- `include_bias=True` é usado para se incluir o vetor de **1s** dado que a função original possui um termo de bias.

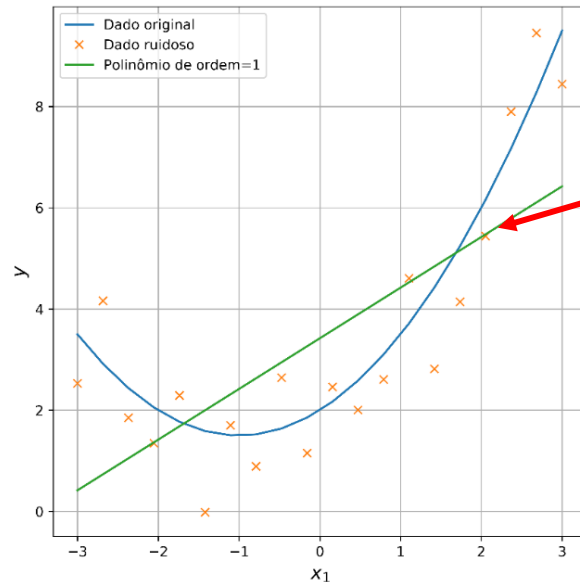
## Exemplos:

- [polynomial features from scikit.ipynb](#)
- [polynomial regression mse calc. ipynb](#)



# Regressão Polinomial: Qual ordem usar?

Polinômio de ordem 1.



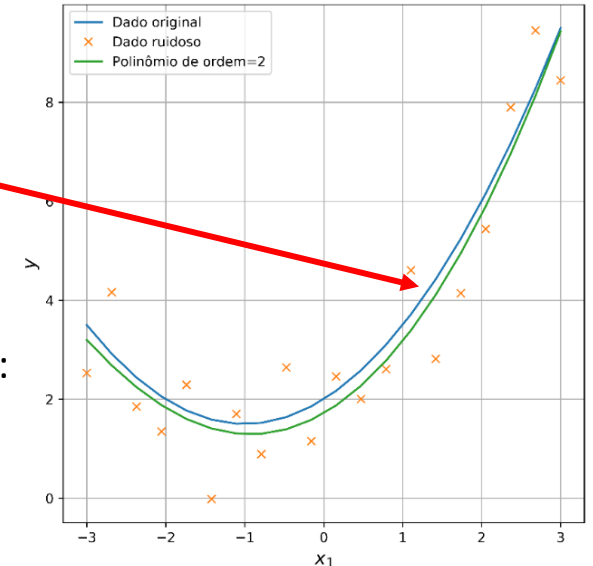
Reta não é flexível o suficiente para se contorcer e aproximar os dados.

**Flexibilidade e grau de generalização** muito baixos.

Encontra relação de compromisso entre **flexibilidade** e **generalização**: **flexibilidade e grau de generalização** médios.

[Exemplo: polynomial\\_regressionv1.ipynb](#)

Polinômio de ordem 2.

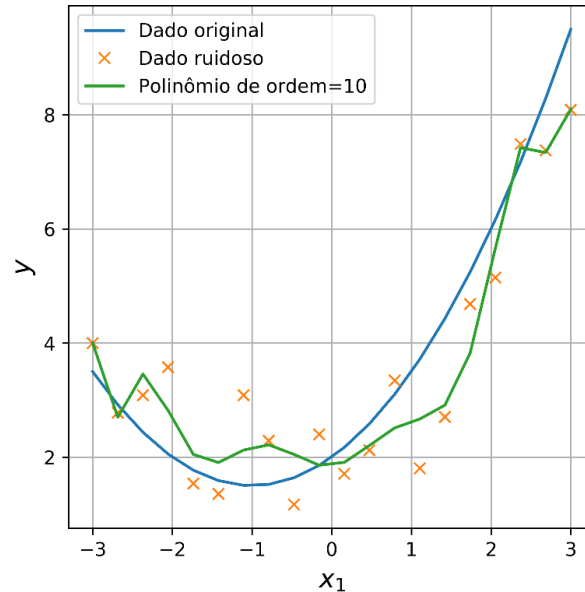


Ordem ótima pois é a mesma do modelo gerador.

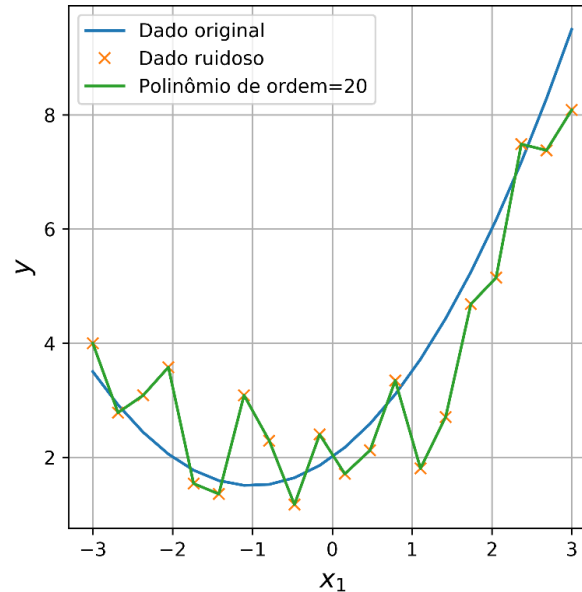
- Polinômio de ordem 1 (i.e., uma reta) não tem flexibilidade o suficiente para aproximar bem os dados.
- O modelo **erra muito tanto para predição dos exemplos de treinamento quanto para exemplos de validação** (ou seja, exemplos não vistos durante o treinamento).
- Efeito conhecido como **subajuste** ou **underfitting**: **flexibilidade e grau de generalização** muito baixos.
- Porém, como esperado, o polinômio de ordem 2 produz a melhor aproximação dos dados, errando pouco para exemplos de treinamento e validação.
  - Essa aproximação será melhor quanto maior for o conjunto de treinamento e/ou menor o ruído.

# Regressão Polinomial: Qual ordem usar?

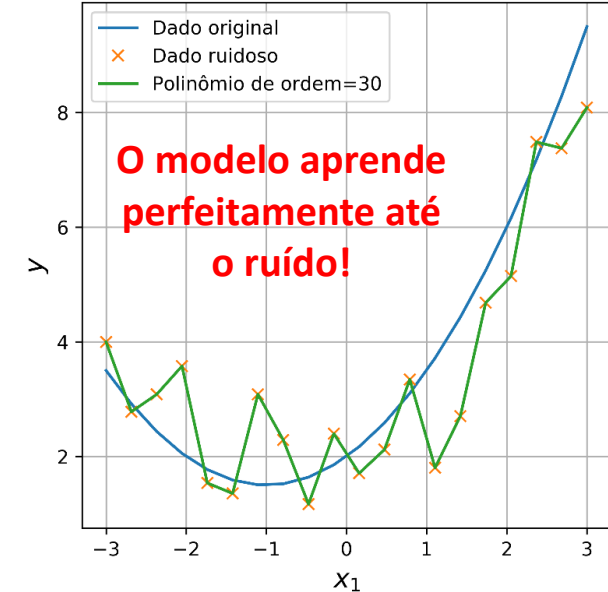
Polinômio de ordem 10.



Polinômio de ordem 20.



Polinômio de ordem 30.



- Polinômios com ordem  $> 2$  tendem a produzir **aproximações perfeitas** dos exemplos disponíveis, ou seja, o modelo acaba **memorizando** os **exemplos de treinamento**.
- Isso ocorre com modelos de ordem próxima à quantidade de exemplos.
- Porém, essa aproximação se distancia bastante do modelo gerador.
- Portanto, esses modelos apresentarão **erros significativamente maiores** quando forem apresentados a **exemplos de validação** (i.e., dados não vistos durante o treinamento).
- Efeito conhecido como **sobreajuste** ou **overfitting**: **flexibilidade** muito alta e **grau de generalização** muito baixo.

# Observações quanto à regressão polinomial

- O objetivo da regressão é o de encontrar a ***melhor aproximação possível*** do ***mapeamento verdadeiro*** dado o conjunto ***limitado*** de exemplos de treinamento.
- ***Melhor aproximação possível***: por mais que tenhamos muitos exemplos, eles dificilmente vão nos dar todos os ***possíveis mapeamentos***  $y = f(x)$ , pois em muitos casos a função  $f(x)$ , é contínua e, conseqüentemente, possui infinitos valores.
- Portanto, queremos um modelo que se ***aproxime o melhor possível*** de  $f(x)$  e, conseqüentemente, ***generalize bem***.
- Vimos que ***polinômios de ordem inferior ao do mapeamento verdadeiro não conseguem aproximar o mapeamento verdadeiro***, pois eles ***não têm complexidade/flexibilidade o suficiente*** para capturar a curvatura da função.
- Percebemos também que é ***possível encontrar uma aproximação ótima*** com relação aos ***exemplos de treinamento*** que, ao mesmo tempo, produz um mapeamento,  $\hat{y} = h(x)$ , que ***difere significativamente do mapeamento verdadeiro***.

# Observações quanto à regressão polinomial

- Ao ***aumentarmos a ordem do polinômio***, ***percebemos uma redução progressiva do erro de aproximação*** em relação aos exemplos de treinamento.
- Porém, isso ***não necessariamente significa que estamos***, de fato, ***construindo um modelo melhor***.
- Portanto, precisamos encontrar ***estratégias*** que
  - forneçam ***indicativos*** de como o modelo se comporta ao aproximar o mapeamento verdadeiro ***como um todo*** (e não somente nas amostras de treinamento),
  - ***auxiliem a seleção do modelo mais adequado*** a um determinado problema de aprendizado (e.g., regressão, classificação, agrupamento).
- 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, que aprende a generalizar a partir desse conjunto.
  - ***Validação***: usado para ajustar os hiperparâmetros do modelo (e.g., passo de aprendizagem, ordem).
  - ***Teste***: usado para avaliar o desempenho final (capacidade de generalização) do modelo.
- A ideia é que eles tenham amostras diferentes, mas que os ***conjuntos sejam representativos do fenômeno sendo modelado***.



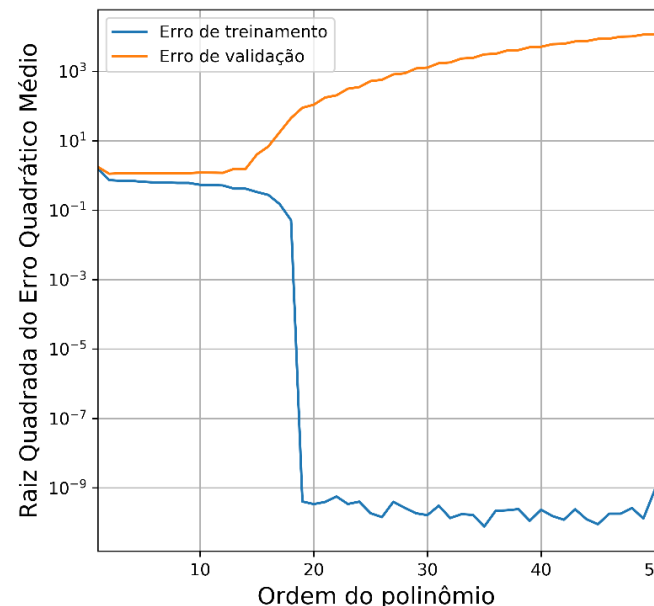
# Erro de treinamento versus erro de validação

- O **erro de treinamento** diminui com o aumento da ordem do polinômio, porém, isso não significa que estamos construindo um modelo melhor.
- Modelos mais complexos com erro de treinamento pequeno não necessariamente são modelos que melhor **generalizam** (ou seja, apresentam alto **erro de validação**).
- **Modelos ótimos** conseguem, **generalizar**, ou seja, mapear valores não vistos durante o treinamento em valores muito próximos dos verdadeiros.
- Em outras palavras, modelos que **generalizam bem** apresentam **erro de validação** próximo ao do **erro de treinamento**.

## Modelo gerador

$$y = 2 + x(i) + 0.5x^2(i) + w(i),$$

onde  $w(i) \sim N(0, 1)$  e  $-3 \leq x(i) \leq 3$ .



# Training set size.

N = 24

# Create training set.

```
x = np.linspace(-3, 3, N).reshape(N, 1)
```

```
y = 2 + x + 0.5*x**2;
```

```
y_noisy = y + np.random.randn(N, 1)
```

# Validation set size.

N = 6

# Create training set.

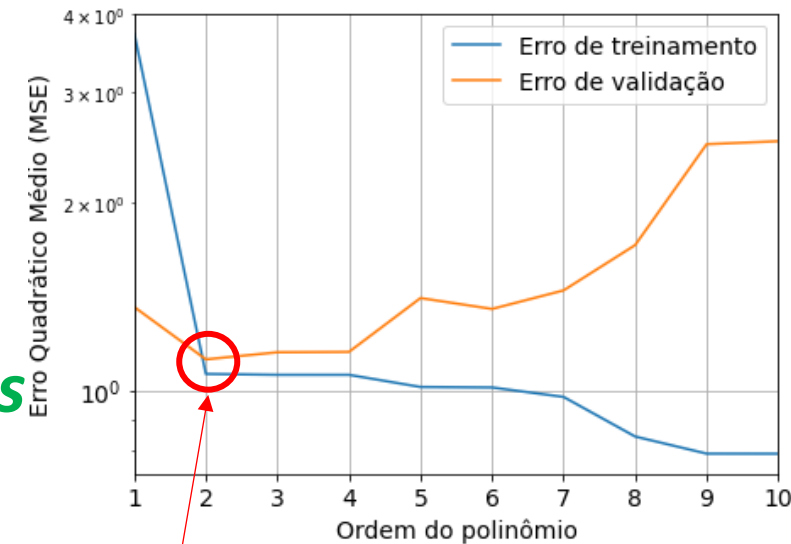
```
x = np.linspace(-3, 3, M).reshape(N, 1)
```

```
y = 2 + x + 0.5*x**2;
```

```
y_noisy = y + np.random.randn(N, 1)
```

# Capacidade e generalização de um modelo

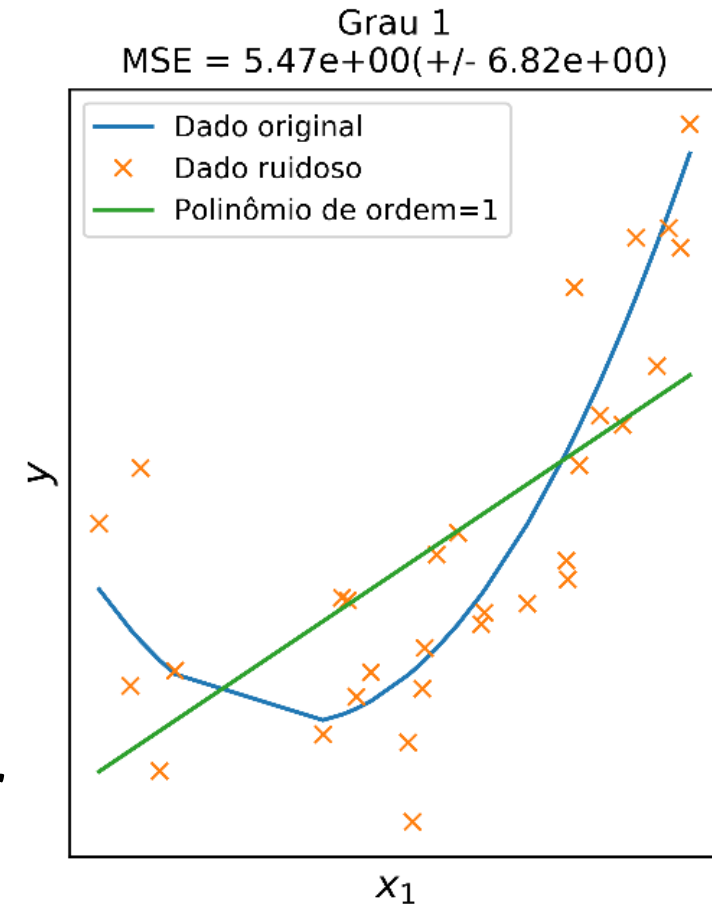
- A **capacidade (ou flexibilidade)** de um modelo diz respeito à sua **capacidade de aprender as regularidades ou características** dos dados do **conjunto de treinamento**.
  - Medida através do **erro de treinamento**.
- O **grau de generalização** de um modelo diz respeito a **qualidade da aproximação**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



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

# Subajuste (Underfitting)

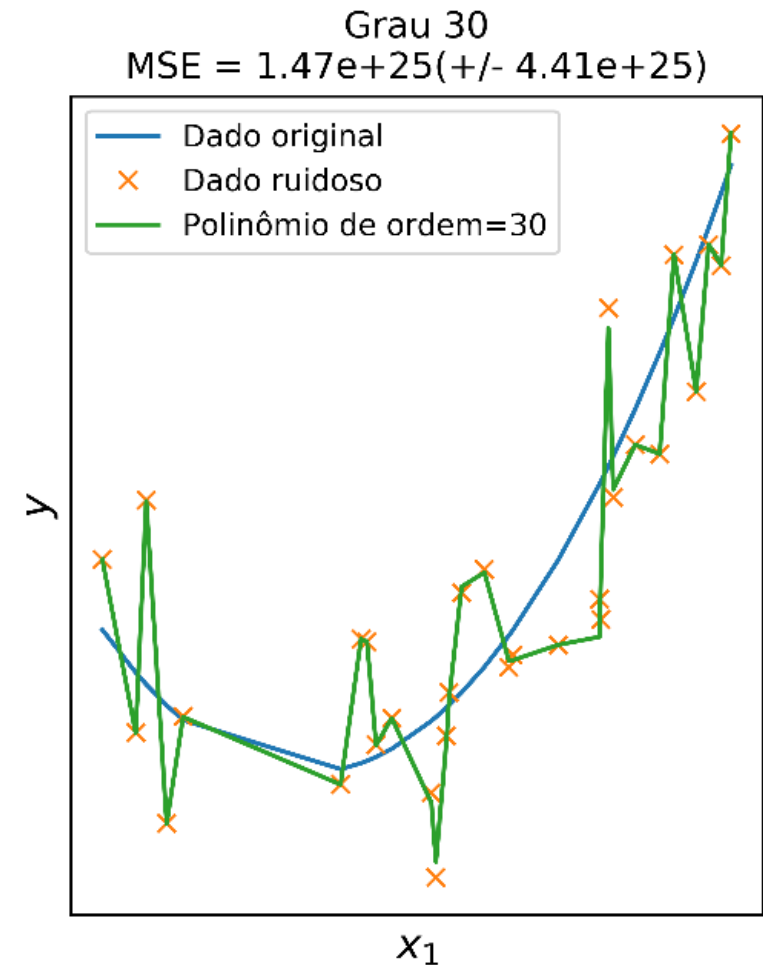
- **Subajuste**: situação em que o modelo não consegue se ajustar aos exemplos de treinamento, falhando em aproximar o **mapeamento verdadeiro**.
- Isto pode ocorrer devido ao **baixa complexidade** (i.e., grau de liberdade) do modelo ou por **problemas de convergência** durante o treinamento.
- **Ambos os erros** (i.e., treinamento e validação) **serão altos**, pois o **modelo falha completamente em capturar a curvatura do mapeamento verdadeiro**.
- Se o modelo está **subajustando**, mesmo que o número de exemplos aumente esta situação não vai desaparecer, é **necessário aumentar a complexidade do modelo**, ou seja, no caso da regressão polinomial, sua ordem.



O modelo com polinômio de grau 1 tem capacidade **muito baixa** e grau de generalização **muito baixo**

# Sobreajuste (Overfitting)

- **Sobreajuste**: é a situação em que o modelo, por ter capacidade muito alta, se ajusta tão bem aos exemplos do conjunto de treinamento que ele “*memoriza*” até o ruído presente nos mesmos.
  - O modelo apresenta **baixíssimo erro de treinamento**.
- Porém, o modelo produz erros significativos quando apresentado a dados inéditos.
  - O modelo apresenta **alto erro de validação (i.e., não generaliza bem)**.
- Se o modelo está **sobreajustando**, então é necessário **diminuir sua complexidade ou aumentar o conjunto de treinamento** até que o erro de validação se aproxime do erro de treinamento.



Um modelo com polinômio de grau 30 e treinado com 30 amostras tem **capacidade muito alta** e grau de generalização **muito baixo**.

# Erros envolvidos no processo de aproximação

- O processo de aproximação (i.e., predição) envolve dois tipos de erros:
  - Redutível (dividido em erros de variância, de viés, de computação e de amostragem);
  - Irredutível.
- **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) provavelmente apresenta *alta variância* e, portanto, se *sobreajusta* aos dados de treinamento.
  - 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 rótulos (i.e., valores esperados).
  - Ou seja, um ***alto valor de bias*** leva a ***altos erros tanto 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 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 ótimos (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.
- **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.
- **Erro de amostragem:** ocorre quando o conjunto de treinamento não é representativo da população de interesse. Isso pode levar a um modelo que não generaliza bem para dados inéditos, mesmo que o modelo tenha erros de viés e de variância baixos.

# Como avaliar e escolher a melhor hipótese?

- Após todas essas informações, algumas perguntas surgem a respeito do modelo e seu treinamento.
- Como escolhemos a ***complexidade da função hipótese (i.e., modelo)*** quando não conhecemos o ***mapeamento verdadeiro***?
  - Ou seja, o ***quão complexo (i.e., flexível)*** deve ser o modelo?
- Como podemos dizer que o modelo está ***ajustando demais ou insuficientemente*** ao conjunto de treinamento?
- O quão bem o modelo prediz valores de saídas para entradas não vistas durante o treinamento (ou seja, ***o quão bem ele generaliza***)?
- ***Resposta***: podemos fazer isso através de ***técnicas de validação cruzada***.

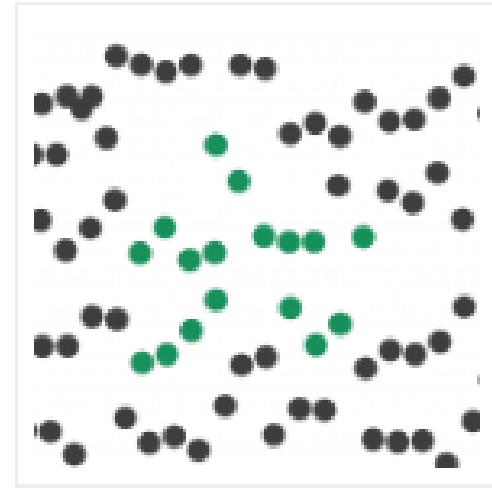


# Validação cruzada

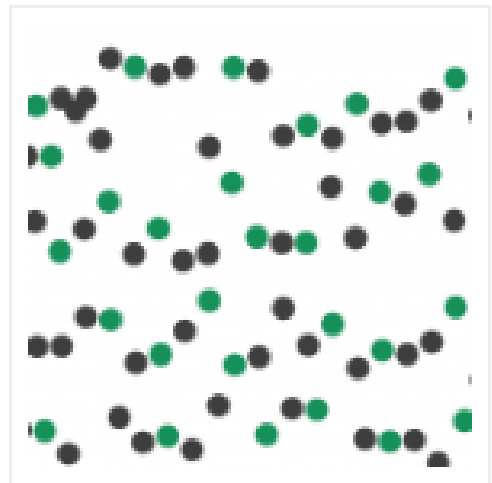
- Técnicas de **validação cruzada** podem ser usadas para se avaliar **quantitativamente** o **sobreajuste** e **subajuste** de um modelo, de forma a encontrar o **modelo ótimo** (i.e., ponto de compromisso entre os erros de viés e de variância).
  - **OBS.:** *validação cruzada é usada para encontrar os valores ótimos de hiperparâmetros de modelos de aprendizado de máquina* (e.g., passo de aprendizagem, ordem do modelo, coeficiente de momentum, número de camadas e nós de uma rede neural, etc.).
- Na **validação cruzada**, nós, geralmente, dividimos o conjunto total de exemplos em dois subconjuntos, o de **treinamento** e o de **validação** (algumas vezes chamado de conjunto de teste).
- O **objetivo** desta **divisão** é treinar o modelo com o conjunto de treinamento e avaliar sua **capacidade em prever** valores de saída para dados que não foram utilizados durante o treinamento (i.e., com o conjunto de validação).
  - Ou seja, avaliamos sua capacidade de **generalização** com o conjunto de validação.

# Estratégias para validação cruzada

- Existem diversas estratégias para validação cruzada, sendo as mais utilizadas: ***holdout***, ***k-fold*** e ***leave-p-out***.
  - Existem outras estratégias, mas elas podem ser vistas como variações de uma dessas três.
- Para aplicarmos estas estratégias, nós devemos nos assegurar que os ***dois conjuntos sejam suficientemente representativos do mapeamento que se pretende aproximar*** de modo que não ocorra o fenômeno conhecido como ***viés de seleção***.
  - O ***viés de seleção*** é o erro introduzido pela seleção de amostras para análise de um modelo de tal forma que os conjuntos de amostras não sejam representativos da população que se pretende analisar.



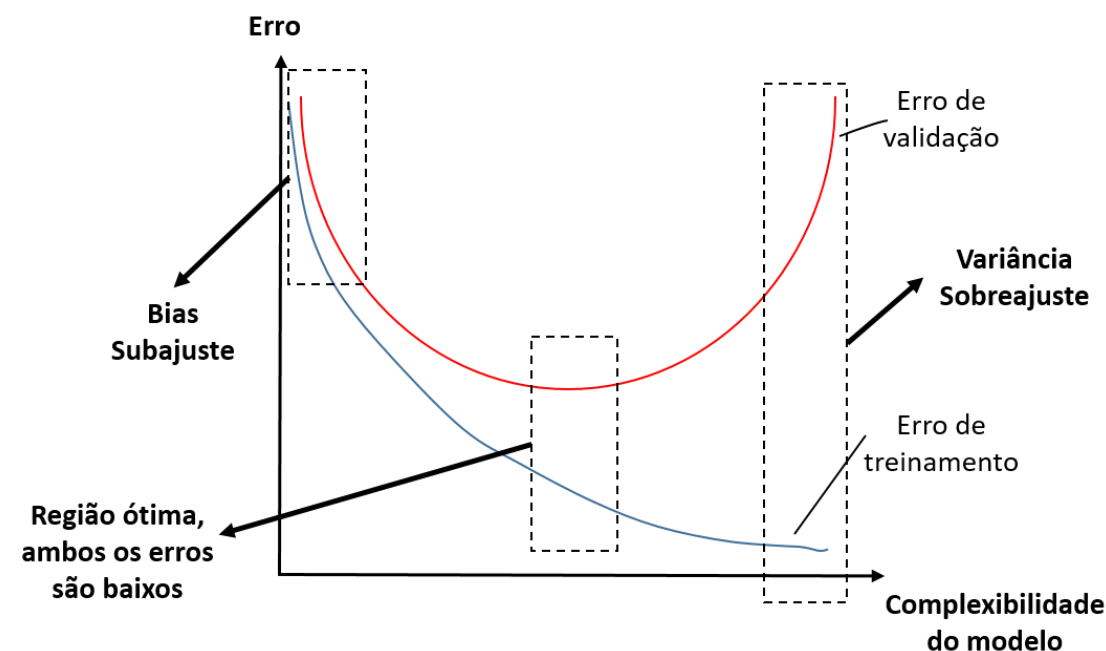
***Viés de seleção***: conjuntos não-representativos.



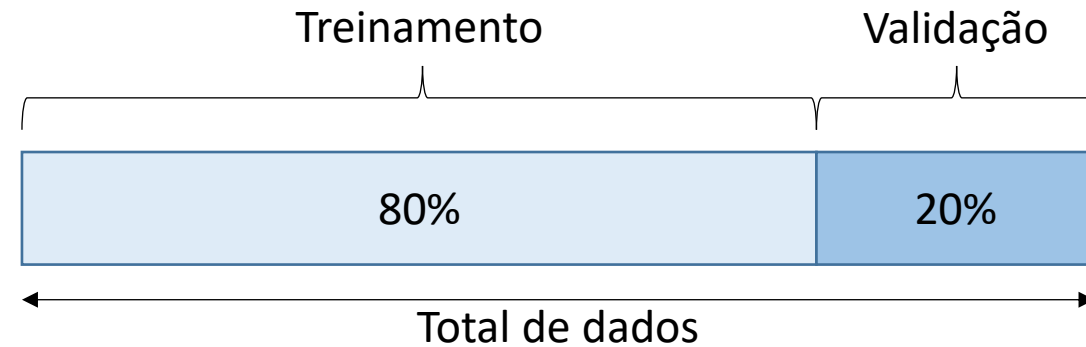
***Amostragem aleatória correta***: conjuntos representativos.

# Comportamento “*típico*” dos erros

- A figura mostra o comportamento “*típico*” dos **erros de treinamento e validação** durante o processo de validação cruzada:
  - Aqui o objetivo é determinar a **flexibilidade** ótima do modelo.
- **Subajuste**: os erros de treinamento e validação são elevados (**viés alto**).
- **Sobreajuste**: o erro de treinamento é baixo, mas o erro de validação é alto (**variância alta**).
- **Região ótima**: ambos os erros são baixos (**relação de compromisso entre viés e variância**).

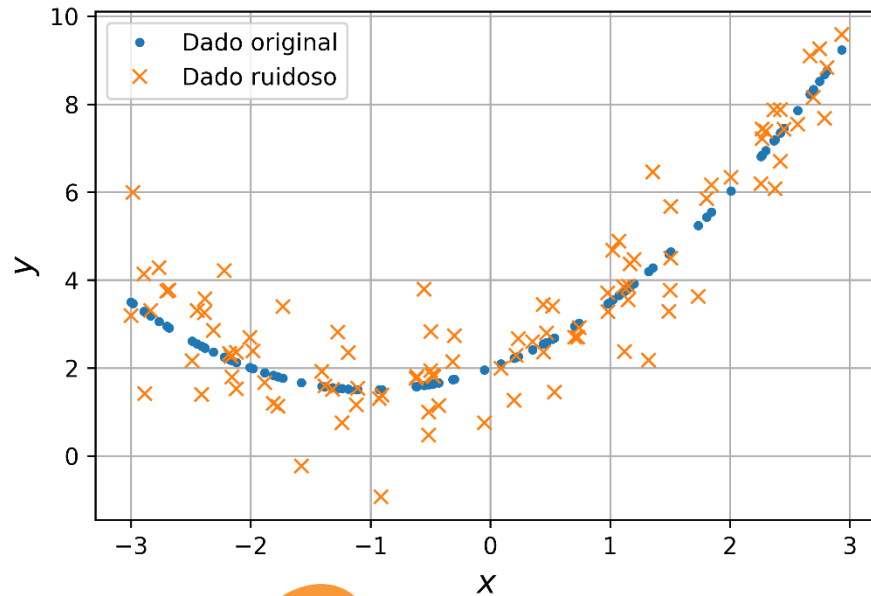


# Holdout



- É a estratégia de validação cruzada ***mais simples*** entre as três e não acarreta em aumento da complexidade computacional, pois tem-se apenas ***um único par de conjuntos de treinamento e validação***.
- Nessa abordagem, divide-se ***aleatoriamente*** o conjunto de dados em ***p*** (%) para treinamento e ***(100 - p)*** (%) para validação.
- Normalmente divide-se o conjunto de dados em 70/80% para treinamento e 20/30% para validação.
- **Desvantagem**
  - Pode sofrer com o problema do ***viés de seleção***, o que, consequentemente, acarreta em ***alta variância***, pois a validação cruzada pode depender muito de quais amostras vão para o conjunto de treinamento e quais vão para o conjunto de validação.
  - Os resultados vão depender da divisão aleatória feita para os conjuntos de treinamento e validação.

# Holdout: Exemplo



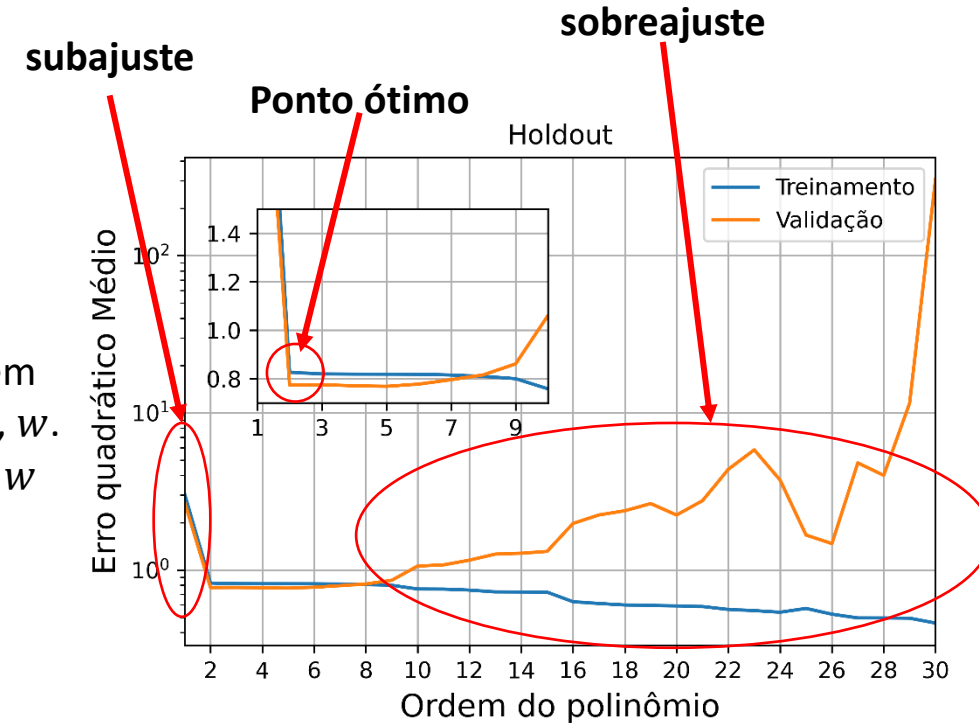
Função observável é um polinômio de segunda ordem com ruído Gaussiano branco,  $w$ .  
 $y_{noisy} = 2 + x + 0.5x^2 + w$



# Split the whole set into random training and validation set.

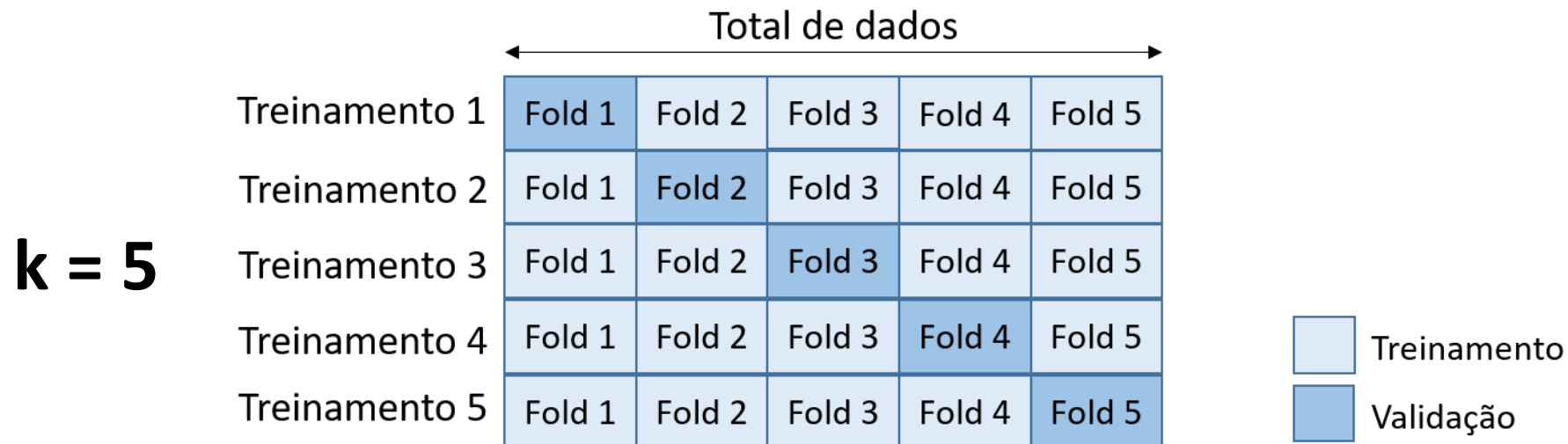
```
x_train, x_val, y_train, y_val = train_test_split(x, y_noisy, test_size=0.3, random_state=10)
```

- 70% para conjunto de treinamento e 30% para conjunto de validação.
- Tempo médio para execução com  $N = 100$  é de aproximadamente 160 ms.
- Erro de treinamento **diminui** conforme a ordem do polinômio aumenta.
- Erro de validação **aumenta** conforme a ordem do polinômio aumenta.
- Qual ordem escolher?
  - O ponto onde **ambos** os erros sejam mínimos (balanço entre flexibilidade e grau de generalização).



# k-Fold

- Estratégia mais elaborada que a do Holdout.
- Consiste em dividir o conjunto total de dados em  $k$  *folds* (partições) de tamanhos iguais (se possível) e realizar  $k$  treinamentos distintos, onde cada um dos  $k$  treinamentos considera  $k-1$  *folds* para treinamento e  $1$  *fold* para validação.

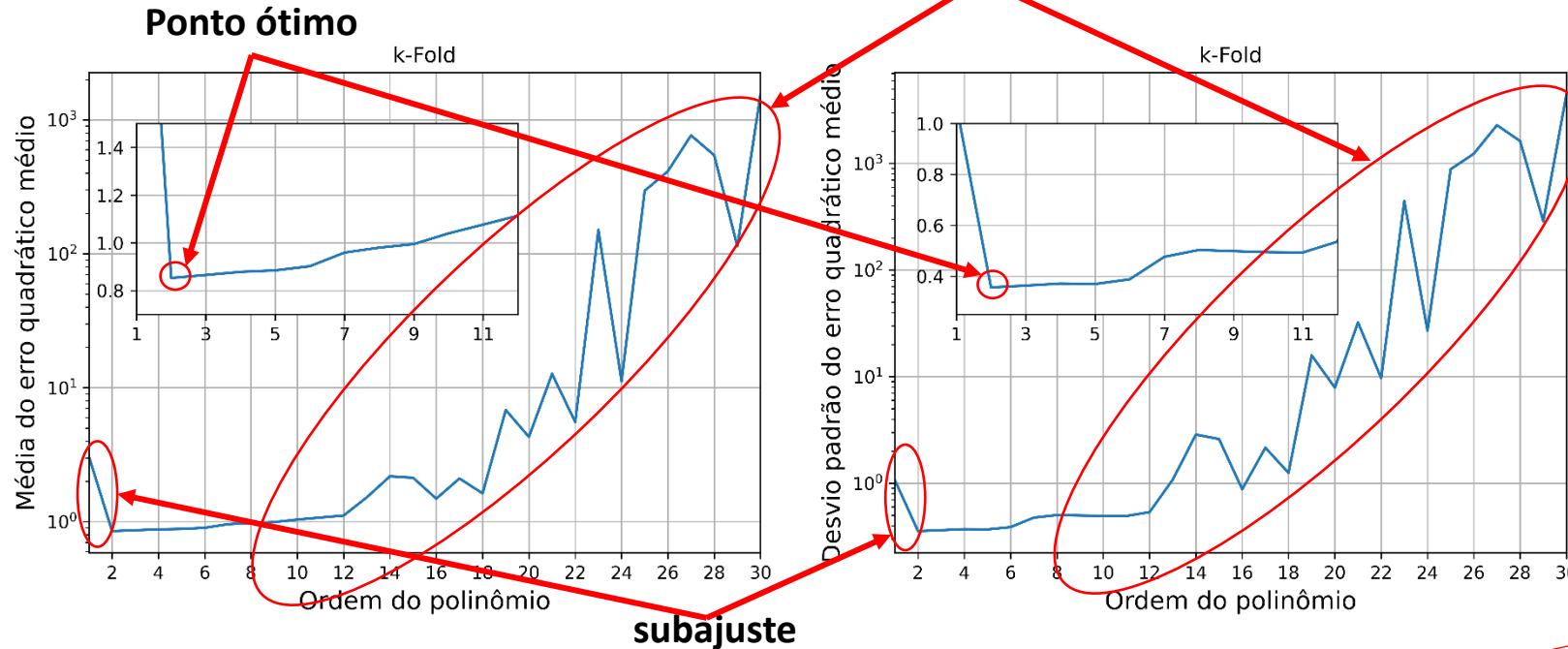


- Cada exemplo entra em um conjunto de validação exatamente **1** vez e em um conjunto de treinamento **k-1** vezes.
- O desempenho do modelo é dado pela **média dos erros de validação** calculados para cada um dos  $k$  *folds* de validação.

# k-Fold

- Ajuda a reduzir o problema do **viés de seleção** em relação ao **holdout**, pois não usamos apenas **um par de conjuntos de treinamento e validação**.
- Ou seja, o k-Fold garante que **todos os exemplos do conjunto de dados original apareçam nos conjuntos de treinamento e validação**, o que, consequentemente, reduz a **variância** do modelo.
  - A **variância** do modelo resultante é reduzida à medida que **k** aumenta.
  - Normalmente, utiliza-se **k = 5** ou **k = 10**.
- Porém, tenham em mente que o valor de **k deve ser escolhido de forma que cada conjunto de treinamento e validação seja grande o suficiente para ser estatisticamente representativo** do conjunto de dados original.
- O k-Fold é bastante **útil quando se tem conjuntos de dados pequenos a moderados**.
- **Desvantagem**
  - O treinamento deve ser executado novamente do zero **k** vezes, o que significa que leva-se **k** vezes mais tempo para se fazer a avaliação do modelo (treinamento + validação).

# k-Fold: Exemplo



```
kfold = KFold(n_splits=10, shuffle=True, random_state=100)
scores = cross_val_score(regressor, x, y_noisy, scoring='neg_mean_squared_error', cv=kfold)
```

- Usa-se a mesma função observável do exemplo anterior.
- Tempo médio para execução com  $N = 100$  exemplos é de aproximadamente 1.9 s.
- Gráficos mostram a média e desvio padrão do MSE para as 10 etapas de treinamento/validação.
- Média e desvio padrão do MSE aumentam com a ordem do polinômio.
- Qual ordem escolher?
  - O ponto onde **ambos**, média e desvio padrão do MSE, sejam mínimos.

Conforme o modelo se **sobreajusta** aos dados de treinamento, sua variância aumenta, devido a redução de seu grau de generalização.

Em teoria, a variância deve ser muito baixa para modelos com alto grau de generalização.

- **k = 10 folds**: 10 iterações com 9 grupos para treinamento e 1 para teste.
- **shuffle=True**: os exemplos são “embaralhados” antes de dividi-los em k folds.
- **cross\_val\_score** recebe as instâncias de um modelo de regressão e de um de validação cruzada.
- **Scoring**: quanto mais positivo, melhor.

[Exemplo: validacao\\_cruzada.ipynb](#)



# Leave-p-out

- **Valida** um modelo usando **todas as combinações possíveis** de  **$p$**  exemplos como **conjunto de validação** e os  **$N-p$**  exemplos restantes como conjunto de treinamento.
- Para um conjunto de dados com  $N$  amostras, essa estratégia produz

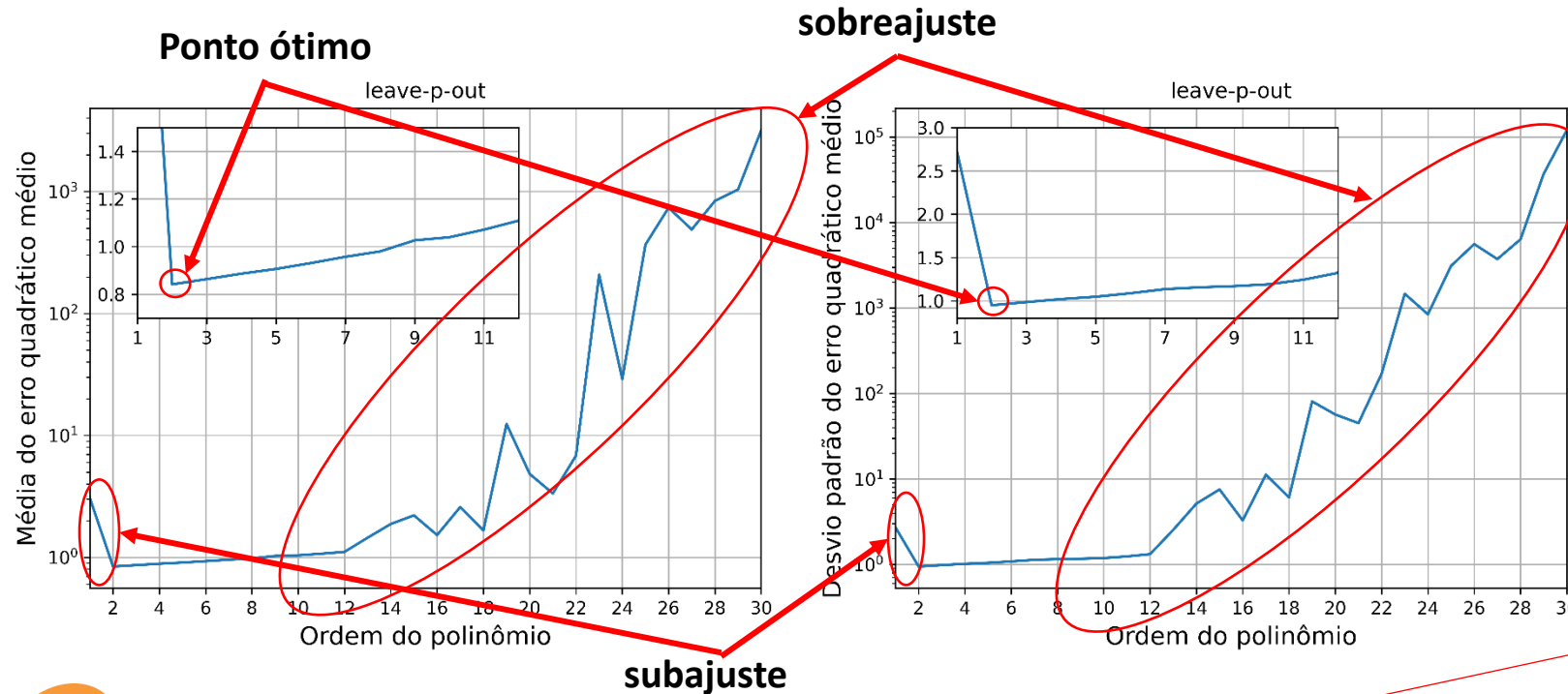
$$\binom{N}{p} = \frac{N!}{p!(N-p)!},$$

A equação mostra quantos subconjuntos distintos de  $p$  exemplos podemos criar a partir de  $N$  exemplos.

pares de conjuntos treinamento/teste, portanto, a complexidade computacional desta estratégia aumenta drasticamente com o aumento de  **$p$** . Exemplos para  $N = 100$ :

- $p = 1 \rightarrow 100$  combinações
- $p = 2 \rightarrow 4.950$  combinações
- $p = 5 \rightarrow 75.287.520$  combinações
- Fornece **estimativas de erro e desvio padrão mais precisas** do que as abordagens anteriores, pois tem-se **mais etapas de treinamento/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,  **$p = 1$** .

# Leave-p-out: Exemplo



`lpocv = LeavePOut(p=2)`

`scores = cross_val_score(regressor, x, y_noisy, scoring='neg_mean_squared_error', cv=lpocv)`

- Para ordem igual a 1, a média e desvio padrão são elevados: **subajuste**.
- Conforme a ordem aumenta, ambos diminuem, atingindo o **ponto ótimo** quando igual a 2.
- Porém, conforme a ordem continua a aumentar, ambos aumentam, indicando **sobreajuste**.

- **p = 2**: 4950 combinações possíveis com 98 exemplos para treinamento e 2 para validação.
- **cross\_val\_score** recebe as instâncias de um modelo de regressão e de validação cruzada.
- **Scoring**: quanto mais positivo, melhor.

- Usa-se a mesma função observável do exemplo anterior.
- Tempo médio para execução com  $N = 100$  é de aproximadamente 810 [s] (+ de 13 [m]).
- Gráficos mostram a média e desvio padrão do MSE para as 4950 etapas de treinamento/validação.
- Média e desvio padrão do MSE aumentam com a ordem do polinômio.
- Qual ordem escolher?
  - O ponto onde **ambos**, média e desvio padrão do MSE, sejam mínimos.

[Exemplo: validacao\\_cruzada.ipynb](#)

# Qual estratégia utilizar?

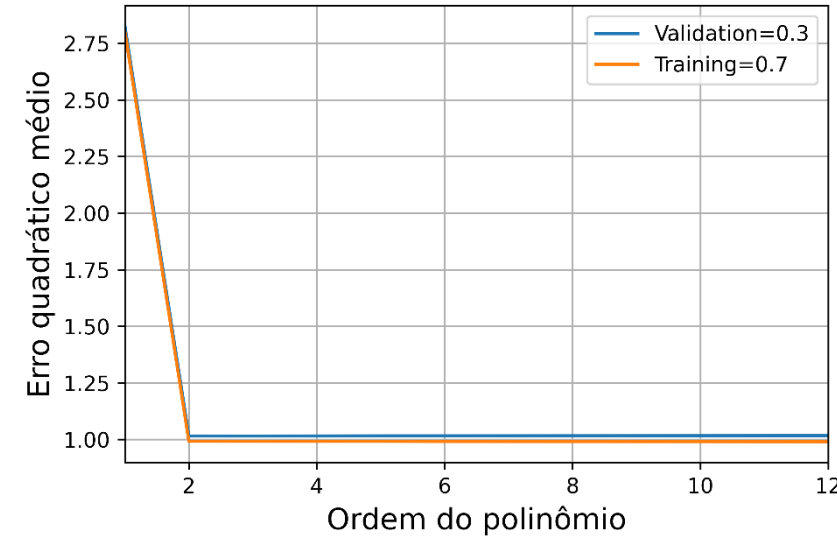
- A abordagem ***leave-p-out*** é a que ***dá indicações mais claras*** de qual ordem usar, ***pois usa um maior número de pares treinamento/validação***, aumentando a confiabilidade da média e do desvio padrão do MSE.
  - Porém, ela é bastante custosa em relação ao tempo necessário para se executá-lo, mesmo com uma base de 100 amostras leva-se mais de 13 minutos!
  - 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*** já daria 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.

[Exemplo: validacao cruzada base maior.ipynb](#)

[Exemplo: holdout com bases de tamanhos diferentes.ipynb](#)

# Qual ordem escolher para o modelo?

- Qual ordem escolher *se os erros de treinamento e validação são pequenos e praticamente constantes para várias ordens de polinômio?*
- Uma resposta é usar o **princípio da navalha de Occam**.
- A **navalha de Occam** é um **princípio lógico** que postula que entre múltiplas explicações adequadas e possíveis para o mesmo conjunto de fatos, deve-se optar pela mais simples delas.
  - Ou seja, deve-se preferir explicações mais simples às mais complicadas/complexas.
- Portanto, usando a **navalha de Occam** escolhemos a função hipótese **menos complexa (i.e., mais simples), mas que se ajusta bem aos dados**.

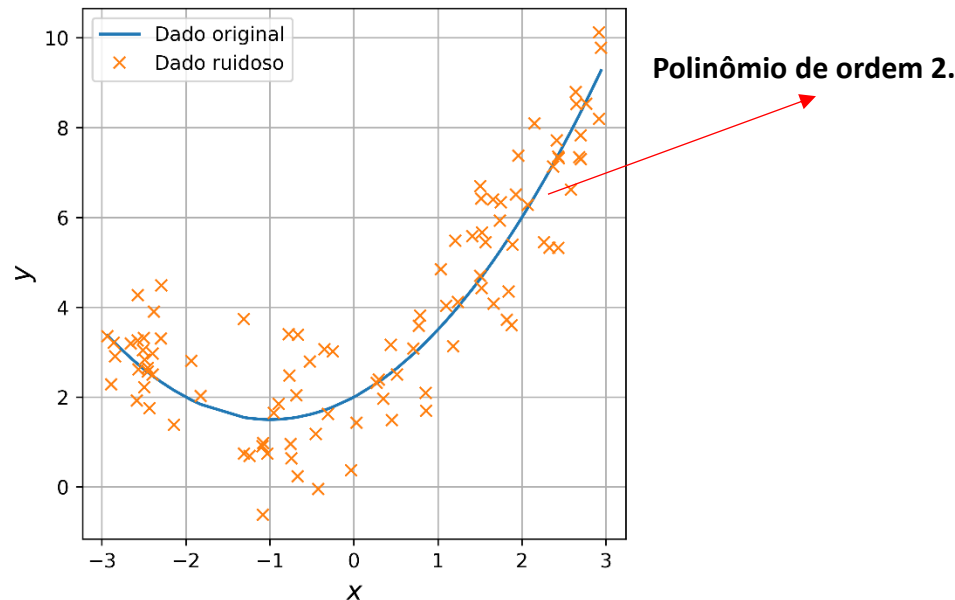


- Mesma função observável dos exemplos anteriores.
- Base de dados com **10000 exemplos**.
- Holdout com 30% para validação.
- Vejam que teoricamente, qualquer ordem maior ou igual a 2 já seria uma boa escolha.
- **Qual ordem escolher?**

# Curvas de Aprendizado

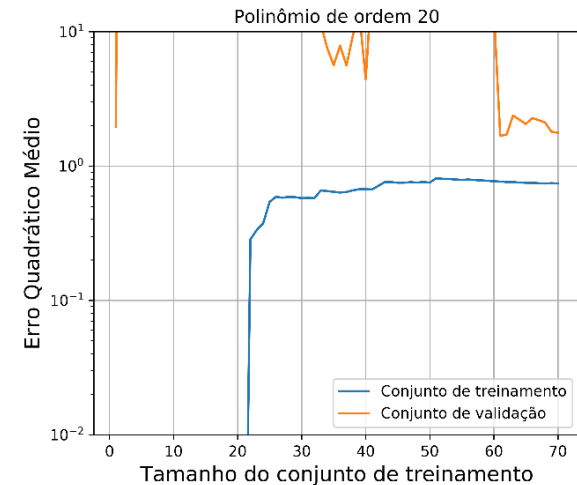
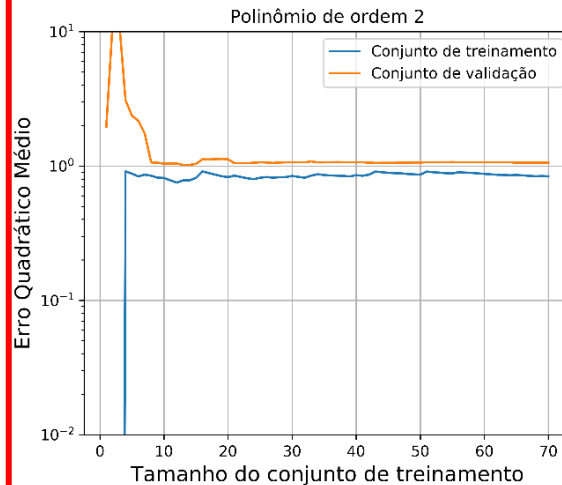
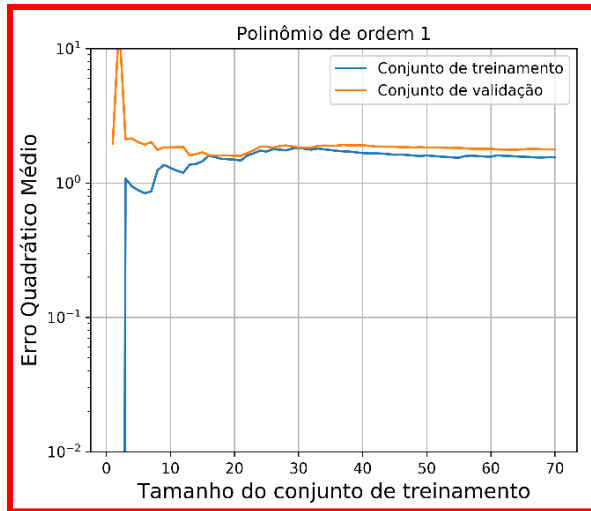
- **Curvas de aprendizado:** são gráficos que **comparam o desempenho do modelo nos conjuntos de treinamento e de validação em função do tamanho do conjunto de treinamento**.
  - **OBS.:** O tamanho do conjunto de validação permanece constante.
- São ferramentas úteis para **avaliar o desempenho do modelo de aprendizado de máquina à medida que mais dados são fornecidos para o treinamento**.
- Essa comparação é normalmente usada para avaliar:
  - Qual é o **melhor nível de complexidade de um modelo** (no caso de polinômios, sua ordem). Em outras palavras, são usadas para determinar o ponto de equilíbrio entre os erros de variância (sobreajuste) e de viés (subajuste).
  - O quanto o **modelo se beneficia de mais dados** (por exemplo, se temos "dados suficientes" ou se o desempenho do modelo ficará melhor se aumentarmos a base de dados).
  - Podemos também usar as curvas para **otimização dos hiperparâmetros** do modelo.

# Curvas de Aprendizado: Exemplo



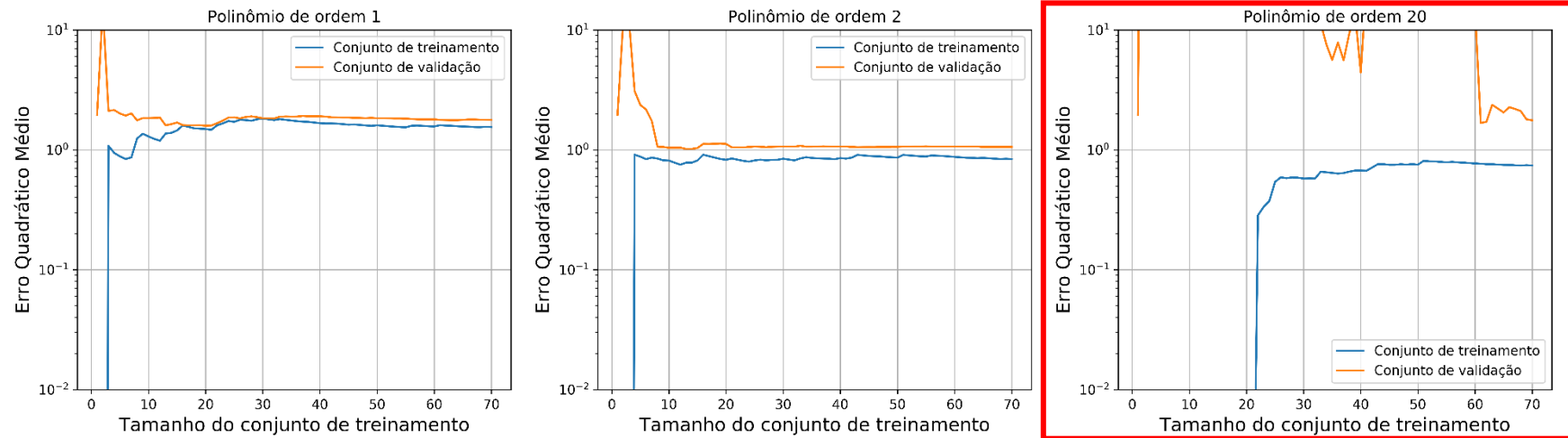
- Caso não conhecêssemos o ***mapeamento verdadeiro*** dos dados ruidosos da figura acima, qual ordem de polinômio melhor aproximaria o ***mapeamento verdadeiro***?
  - Além da validação cruzada, podemos usar as chamadas curvas de aprendizado para encontrar a ordem do polinômio aproximador.

# Curvas de Aprendizado: *Polinômio de ordem 1*



- Com 1 ou 2 exemplos no conjunto de treinamento, o modelo se ajusta perfeitamente (reta), porém, conforme o número de exemplos aumenta, é impossível para ele se ajustar (dados não-lineares e ruído).
- ***Erro nos exemplos de treinamento aumenta até atingir um platô e não diminui mesmo com o aumento do conjunto de treinamento***, pois o modelo não tem flexibilidade.
- ***Erro de validação é alto quando o modelo é treinado com poucos exemplos, porém, diminui conforme o conjunto de treinamento aumenta***, terminando em um platô próximo do erro de treinamento.
- Essas curvas são típicas de um modelo que está ***subajustando*** (erros altos e próximos).
- **O que fazer?** Aumentar a ordem do modelo.

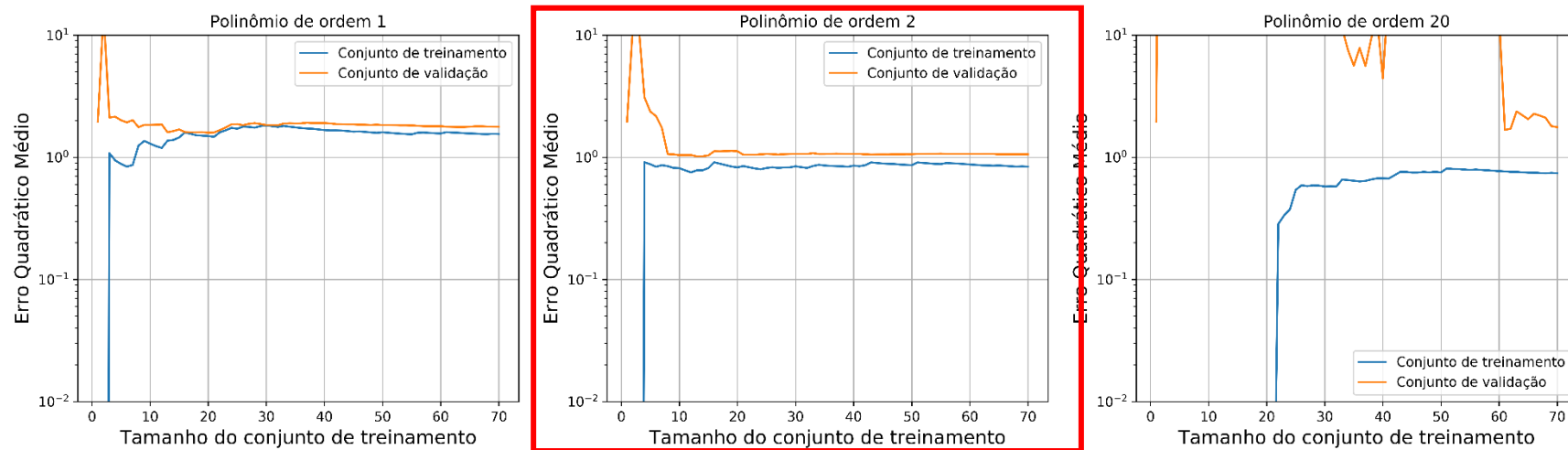
# Curvas de Aprendizado: *Polinômio de ordem 20*



- O erro de treinamento é menor do que com o modelo com polinômio de ordem 1.
- Porém, há uma diferença considerável entre as curvas do erro de treinamento e validação.
- Isso significa que o modelo tem um desempenho melhor no conjunto de treinamento do que no conjunto de validação, indicando que ele está **sobreaajustando**.
- A performance do modelo melhora caso o conjunto de treinamento aumente.
  - Com um conjunto maior, a tendência é que ambos os erros convirjam para o MSE mínimo.
- **O que fazer?** Diminuir a ordem do modelo ou aumentar o conjunto de treinamento.



# Curvas de Aprendizado: *Polinômio de ordem 2*



- A diferença entre os erros diminui com o aumento do conjunto de treinamento se tornando pequena.
- Tanto o erro de treinamento quanto o de validação são menores do que com o modelo com polinômio de ordem 1 (i.e., reta).
- Isso é a indicação de um modelo que está se ajustando bem aos dados de treinamento (i.e., flexibilidade) e é capaz de generalizar bem para os dados de validação.
- Aumentar o conjunto de treinamento faz com que a diferença entre as duas curvas se torne ainda menor.
- **O que fazer?** Escolher esta ordem de polinômio.

# Regularização: penalizando a complexidade dos modelos

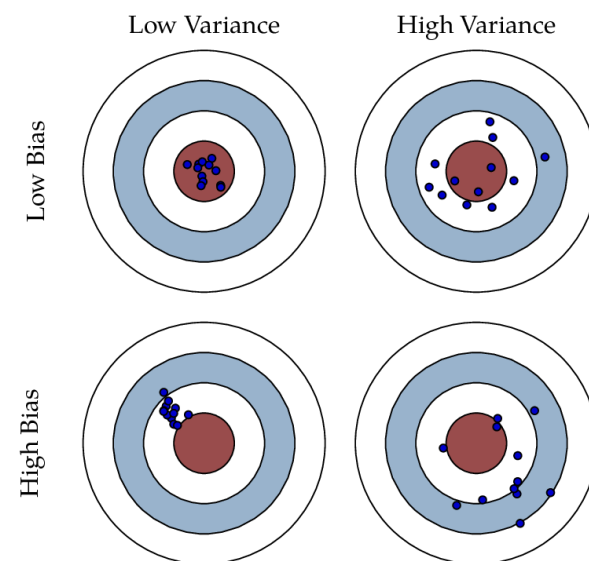
- Anteriormente, nós vimos como escolher o melhor modelo de regressão utilizando ***validação cruzada*** ou ***curvas de aprendizado***.
  - Em ambos os casos, escolhemos o modelo ***menos complexo, mas que generaliza bem***.
- Uma abordagem alternativa é ***minimizar conjuntamente o erro e a complexidade 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

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


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.

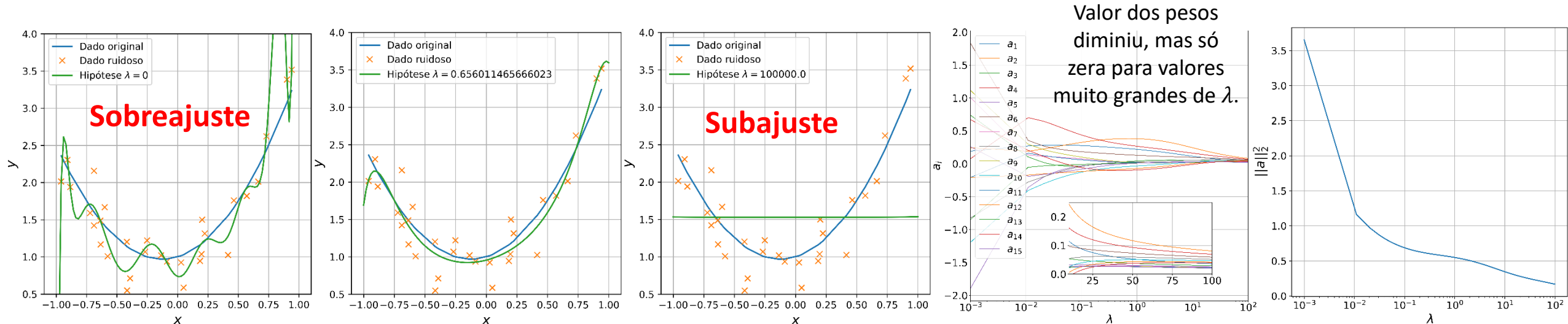
- A **equação de atualização dos pesos** é dada por

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

derivada do termo de regularização

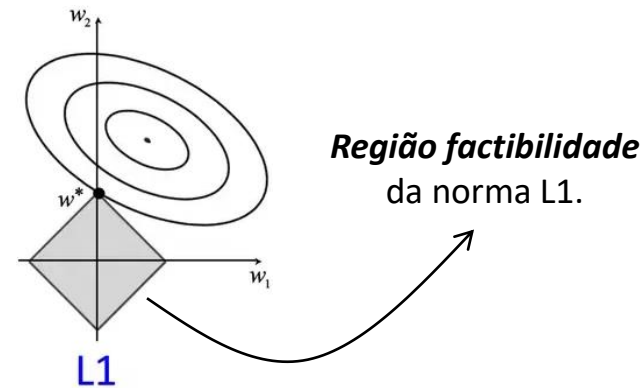


# Ridge Regression: Exemplo



- 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

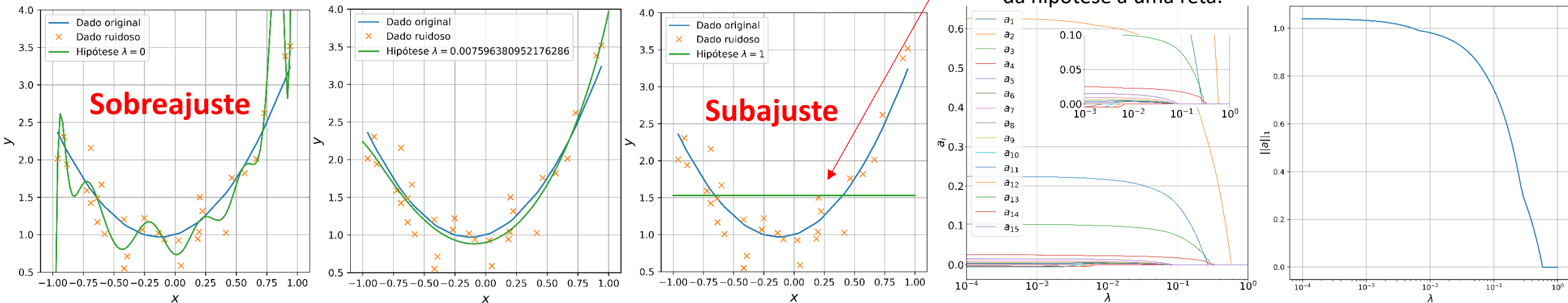
$$\begin{aligned} \min_{\mathbf{a} \in \mathbb{R}^{K+1 \times 1}} \quad & \|\mathbf{y} - \Phi \mathbf{a}\|^2 \\ \text{s. a.} \quad & \|\mathbf{a}\|_1 \leq c, \end{aligned}$$

$c$  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

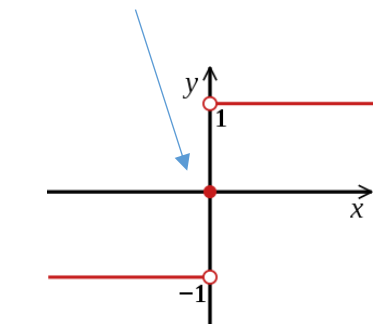


- Mesmas funções geradora e hipótese do exemplo anterior.
- Valores pequenos de  $\lambda$  fazem LASSO se comportar como regressão tradicional e valores muito grandes fazem os pesos serem anulados.
- A regularização com **norma L1** tem como **vantagem** a produção de **modelos esparsos**.
  - Ou seja, 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.

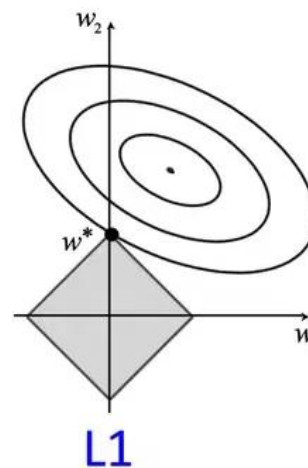
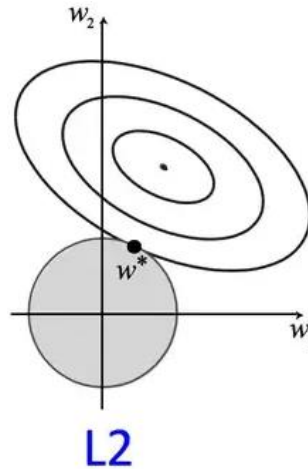
- A **equação de atualização dos pesos** é dada por

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

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**.

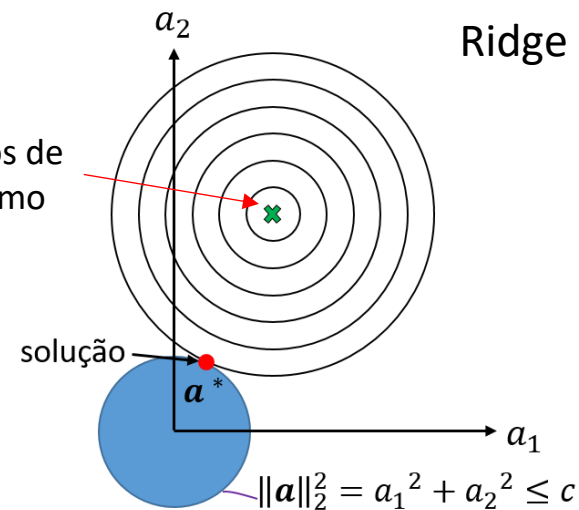
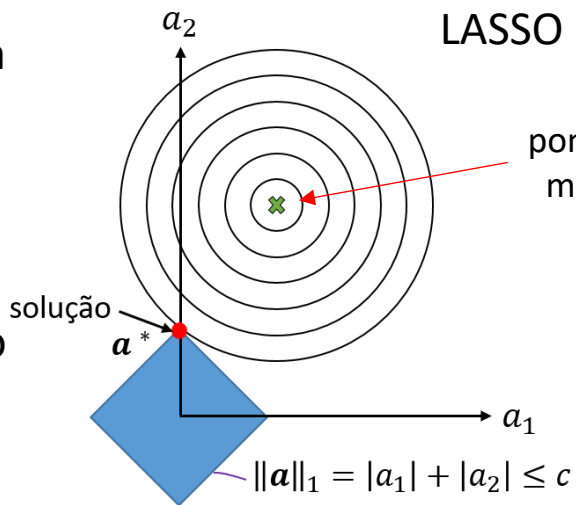


# Vantagem da regressão LASSO sobre Ridge

- 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.

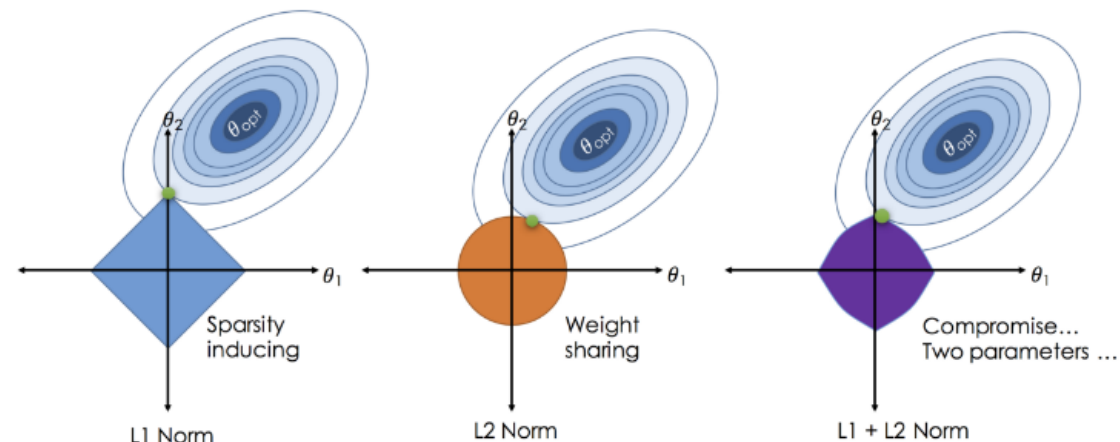
- A figura mostra as **curvas de nível** da **função de erro** de um problema de regressão linear com dois pesos ( $a_1$  e  $a_2$ ) e as regiões do **espaço de hipóteses** onde as restrições L1 e L2 são válidas.
- A solução para ambos os métodos corresponde ao ponto, dentro da **região de factibilidade 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.

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

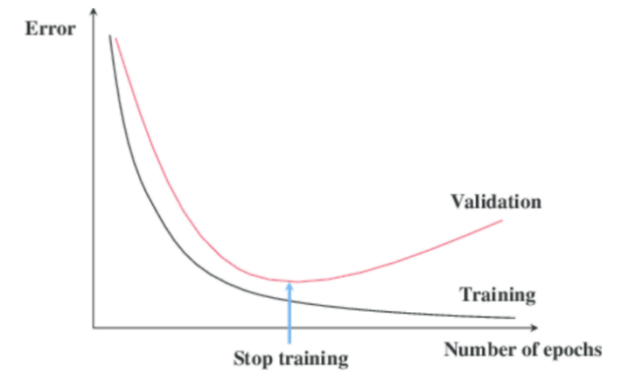
onde  $\kappa \in [0, 1]$  é o **fator de elasticidade** entre as duas normas, ou seja, estabelece uma **relação de compromisso entre as duas normas**.

- 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

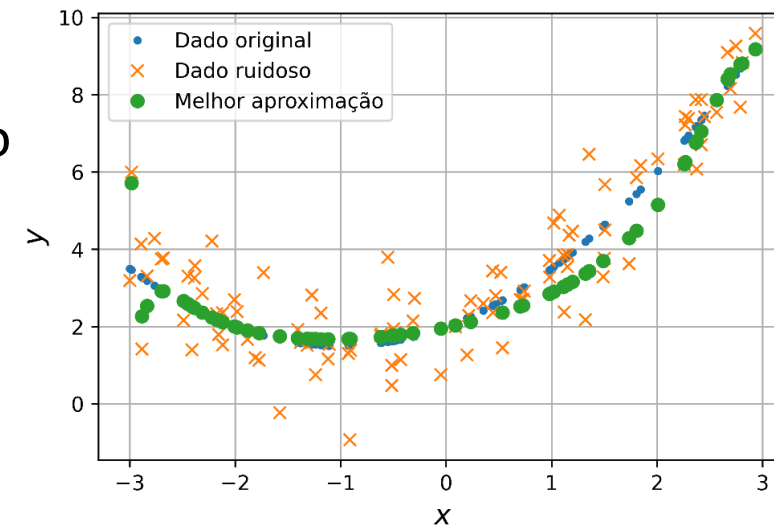
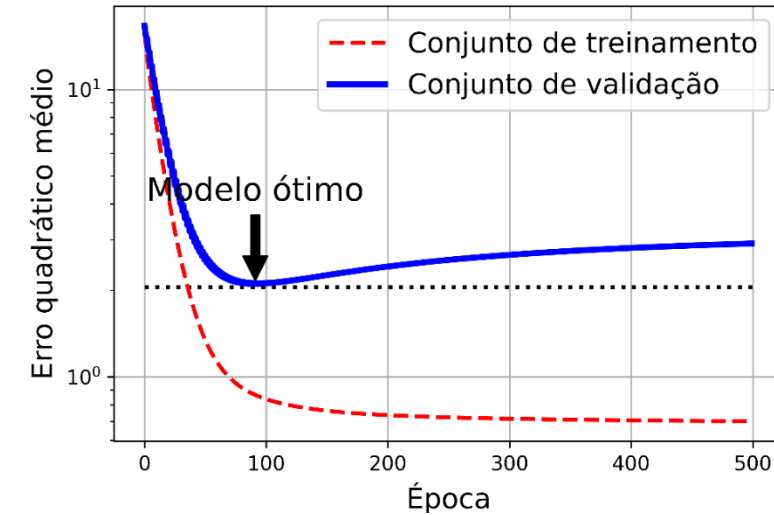
- A figura mostra os erros de treinamento e validação de um modelo de regressão polinomial com grau igual a 90 treinado usando-se o GDE com apenas 70 amostras.

- A função observável é dada por

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

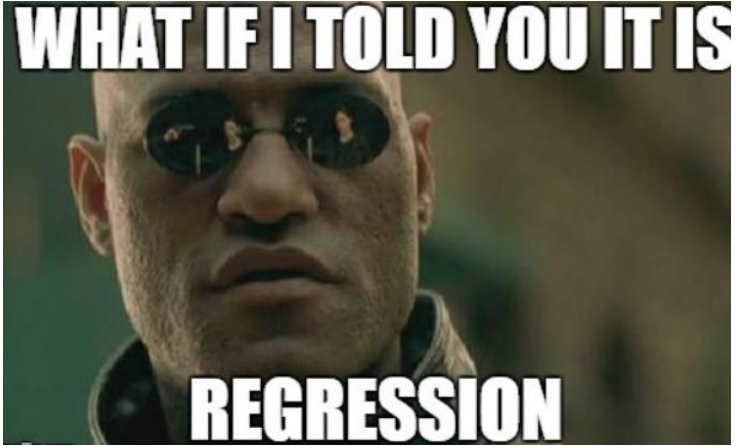
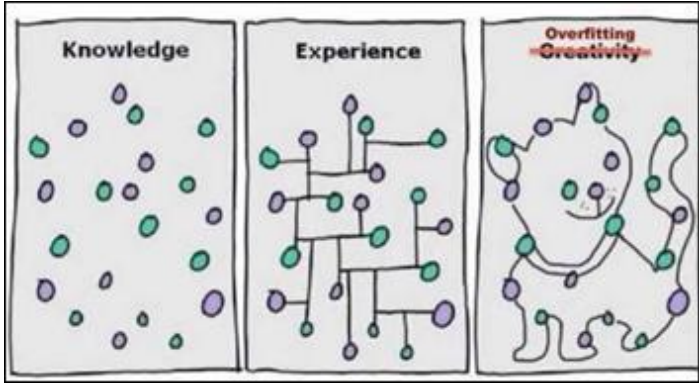
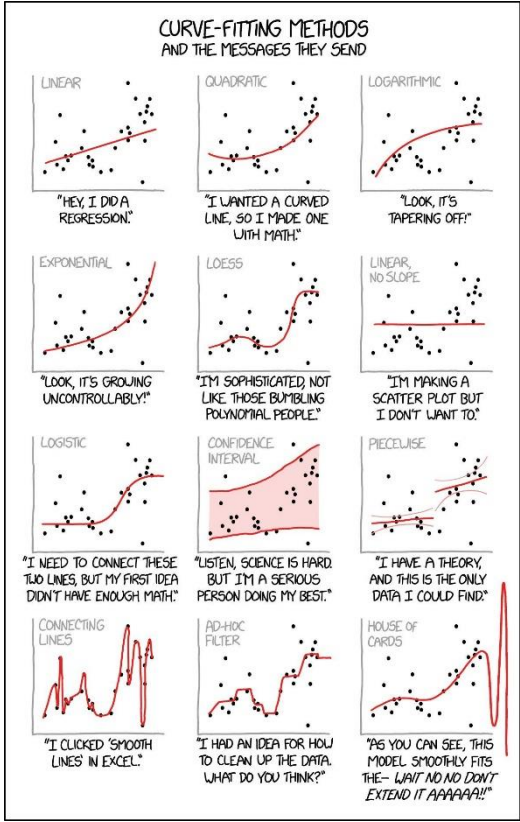
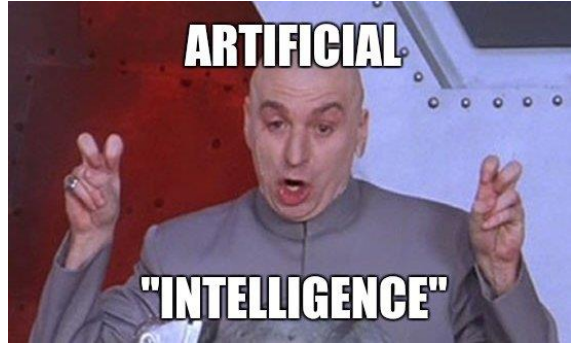
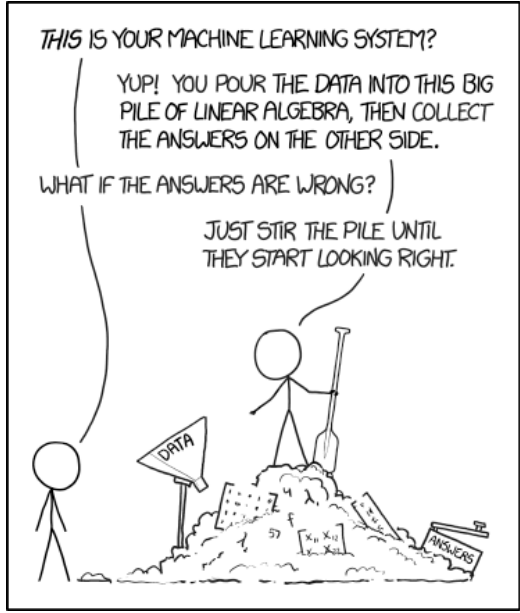
onde  $x \sim U(-3,3)$  e  $w \sim N(0,1)$ . → Função verdadeira

- **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**.



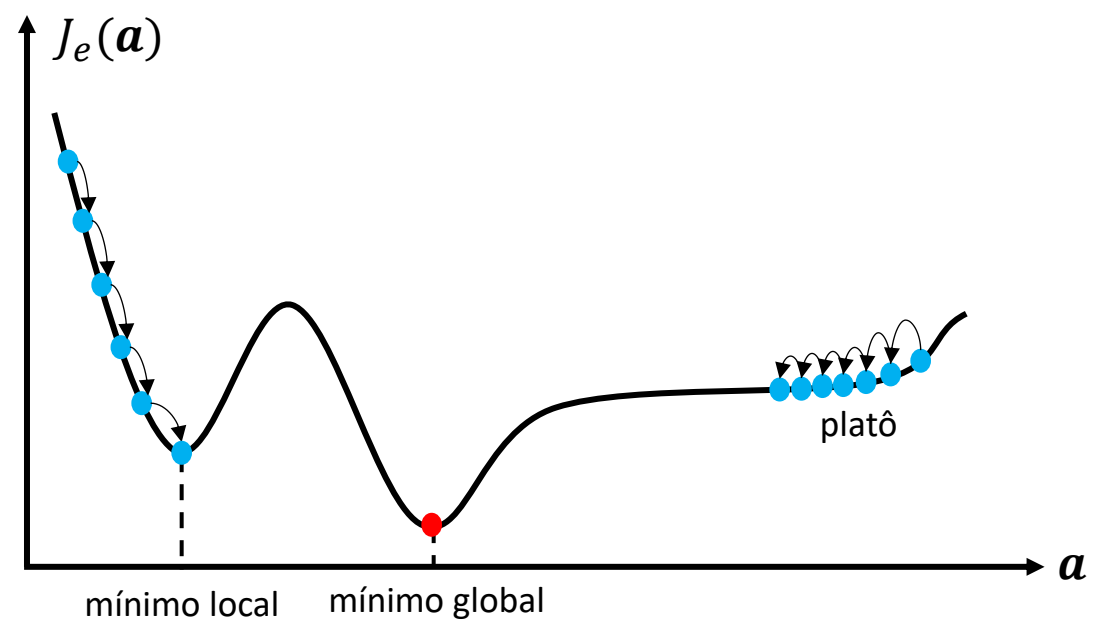
Obrigado!

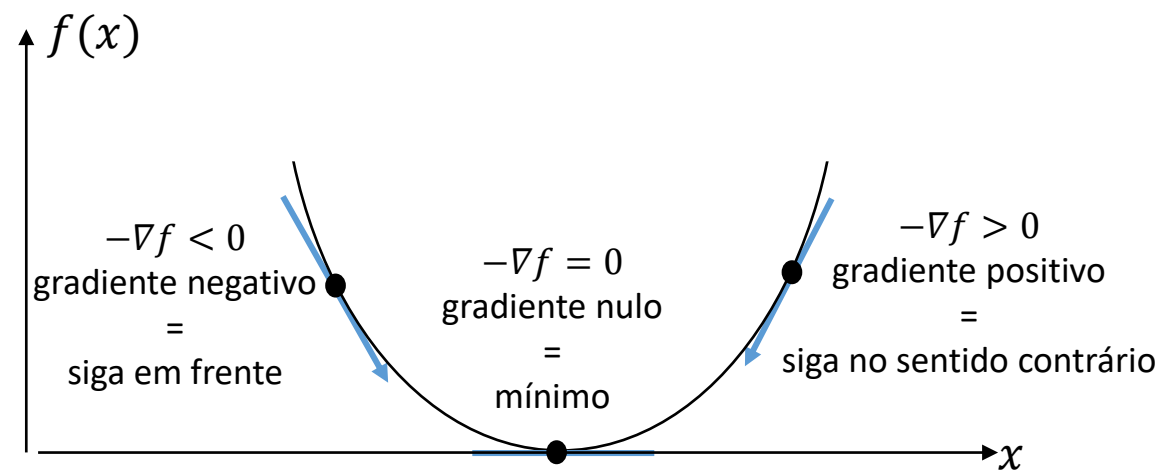
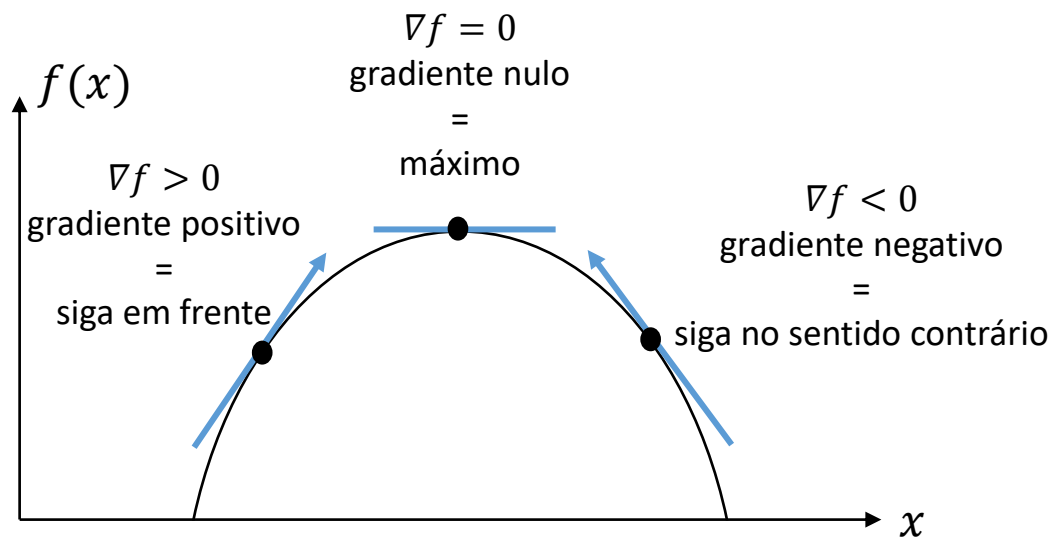


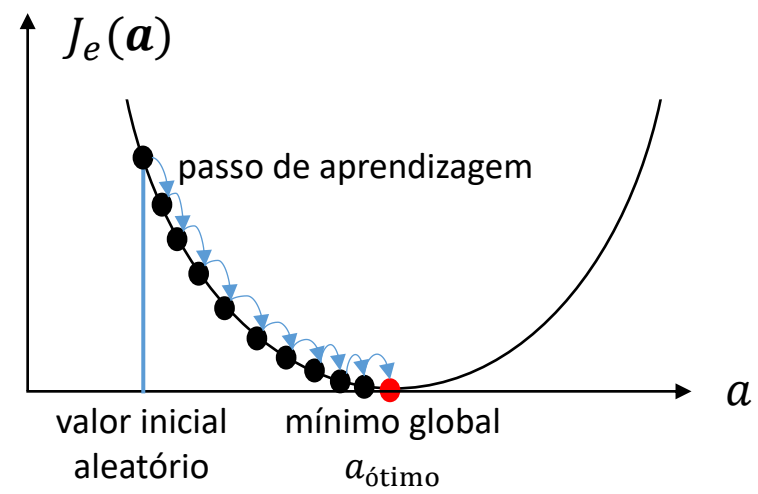


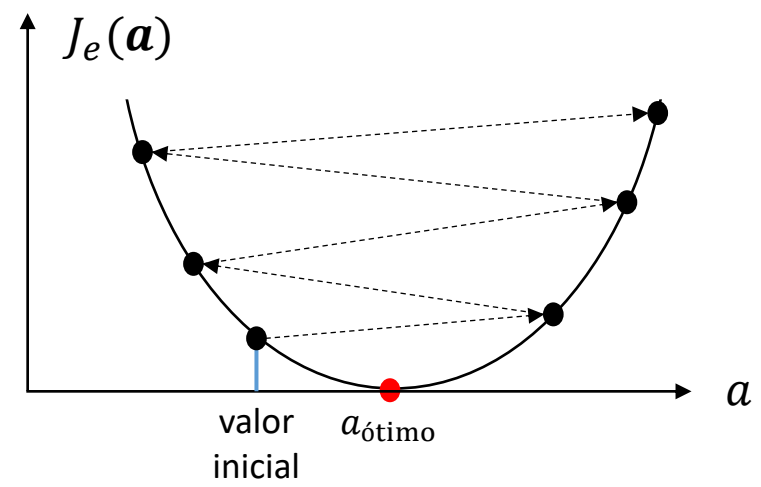
FIGURAS

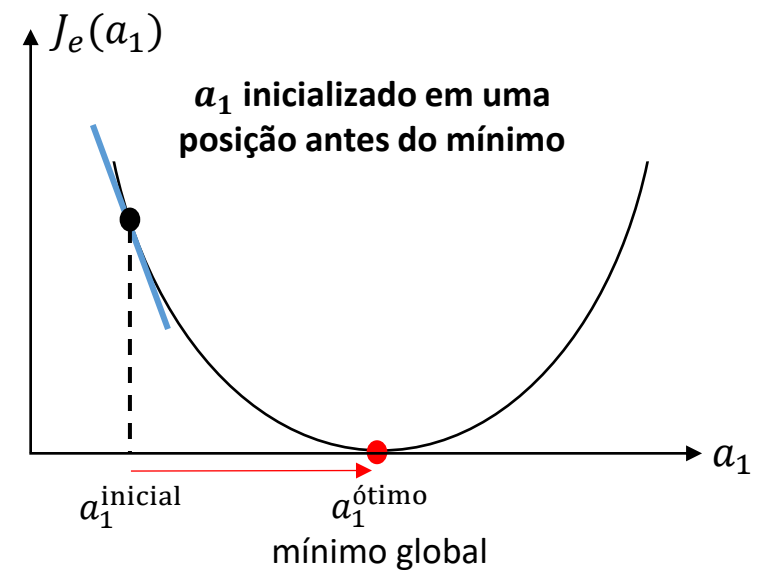




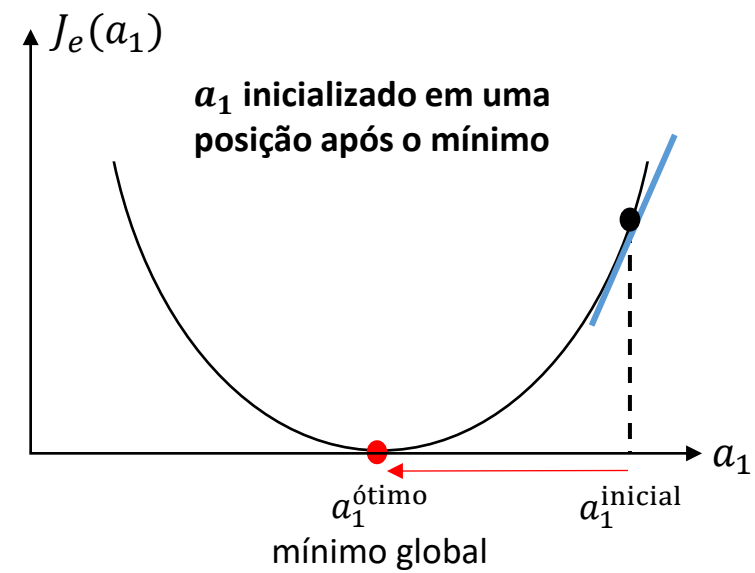




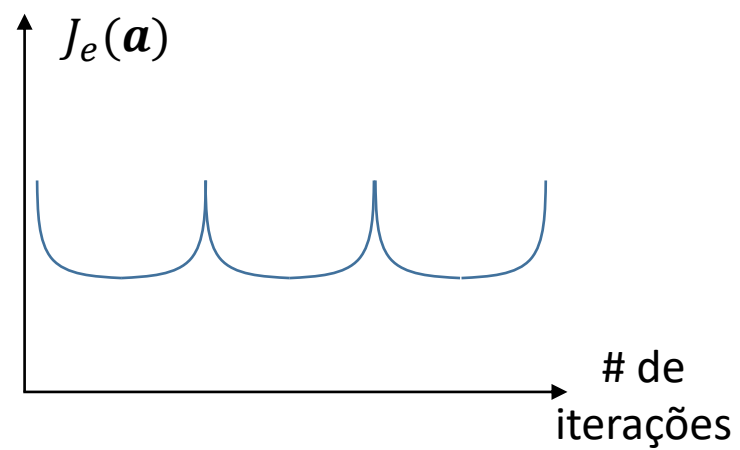
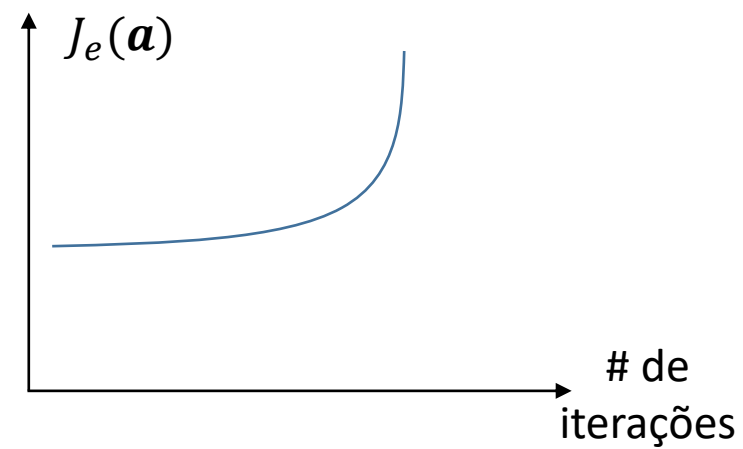
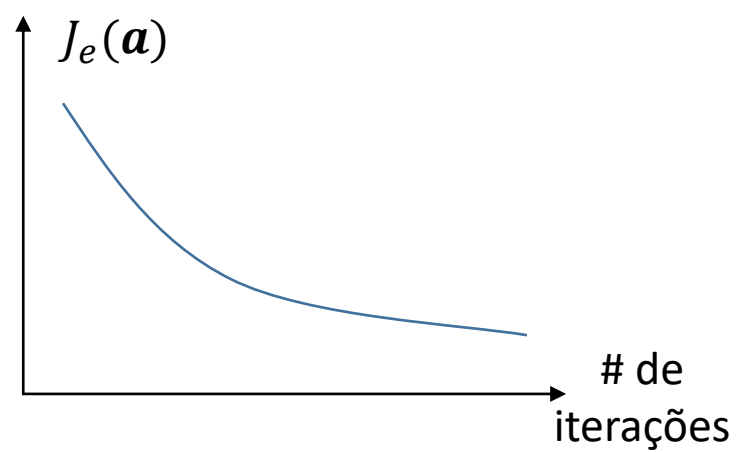
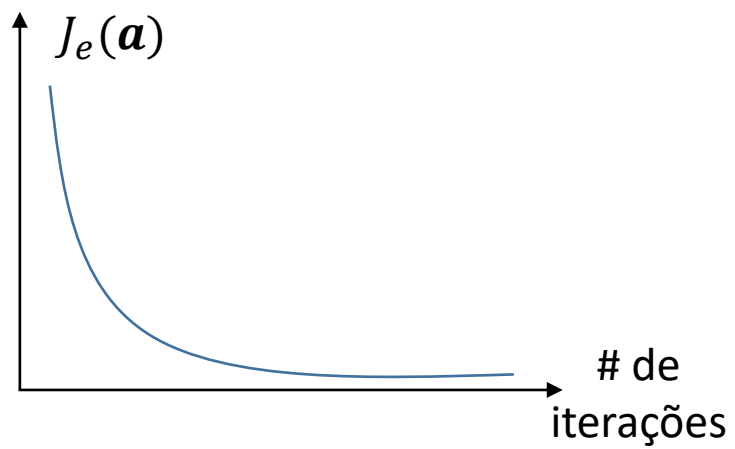


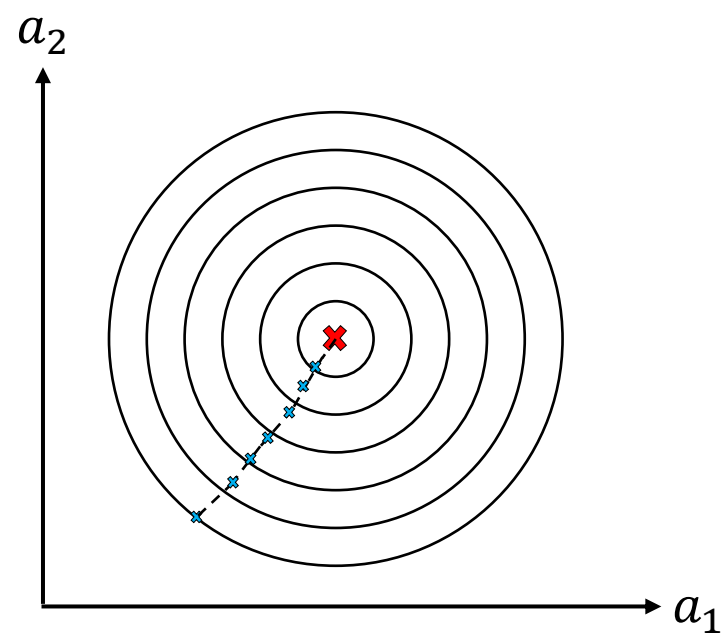
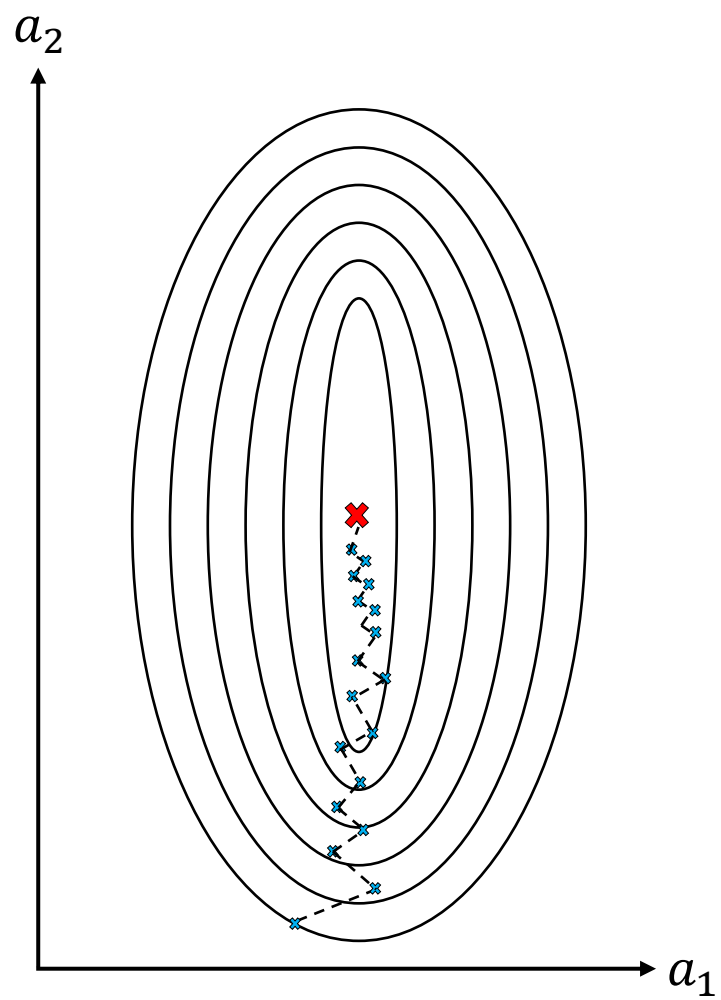


**gradiente negativo:**  $a_1 = a_1^{\text{inicial}} + \alpha \nabla J_e(a_1)$   
 $a_1$  aumenta e se aproxima do mínimo

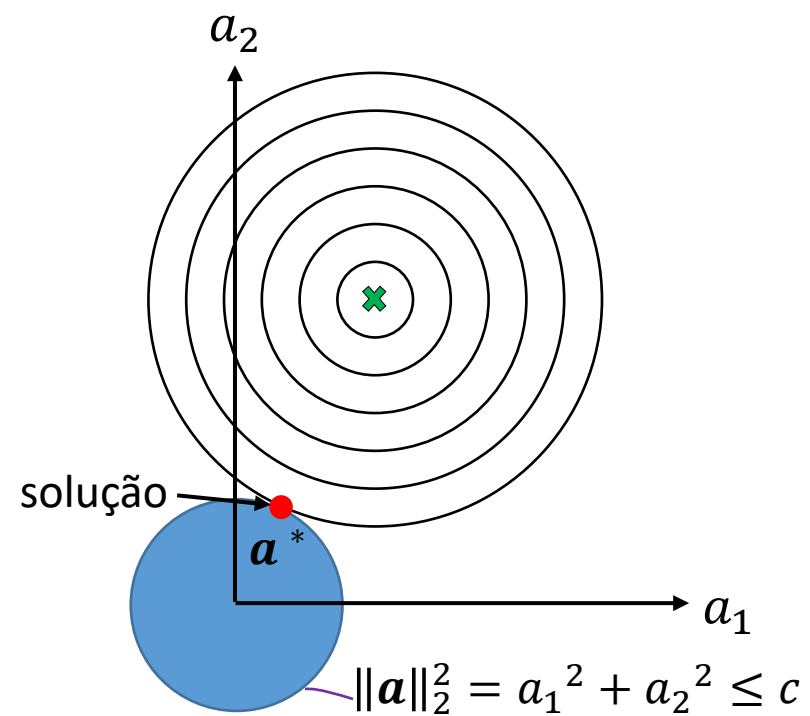
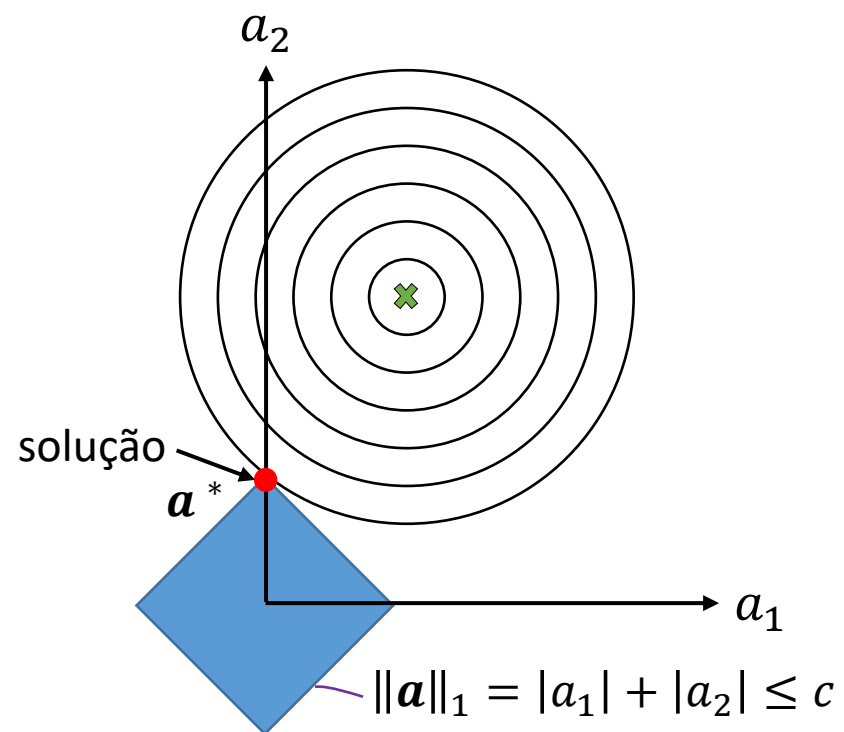


**gradiente positivo:**  $a_1 = a_1^{\text{inicial}} - \alpha \nabla J_e(a_1)$   
 $a_1$  diminuiu e se aproxima do mínimo







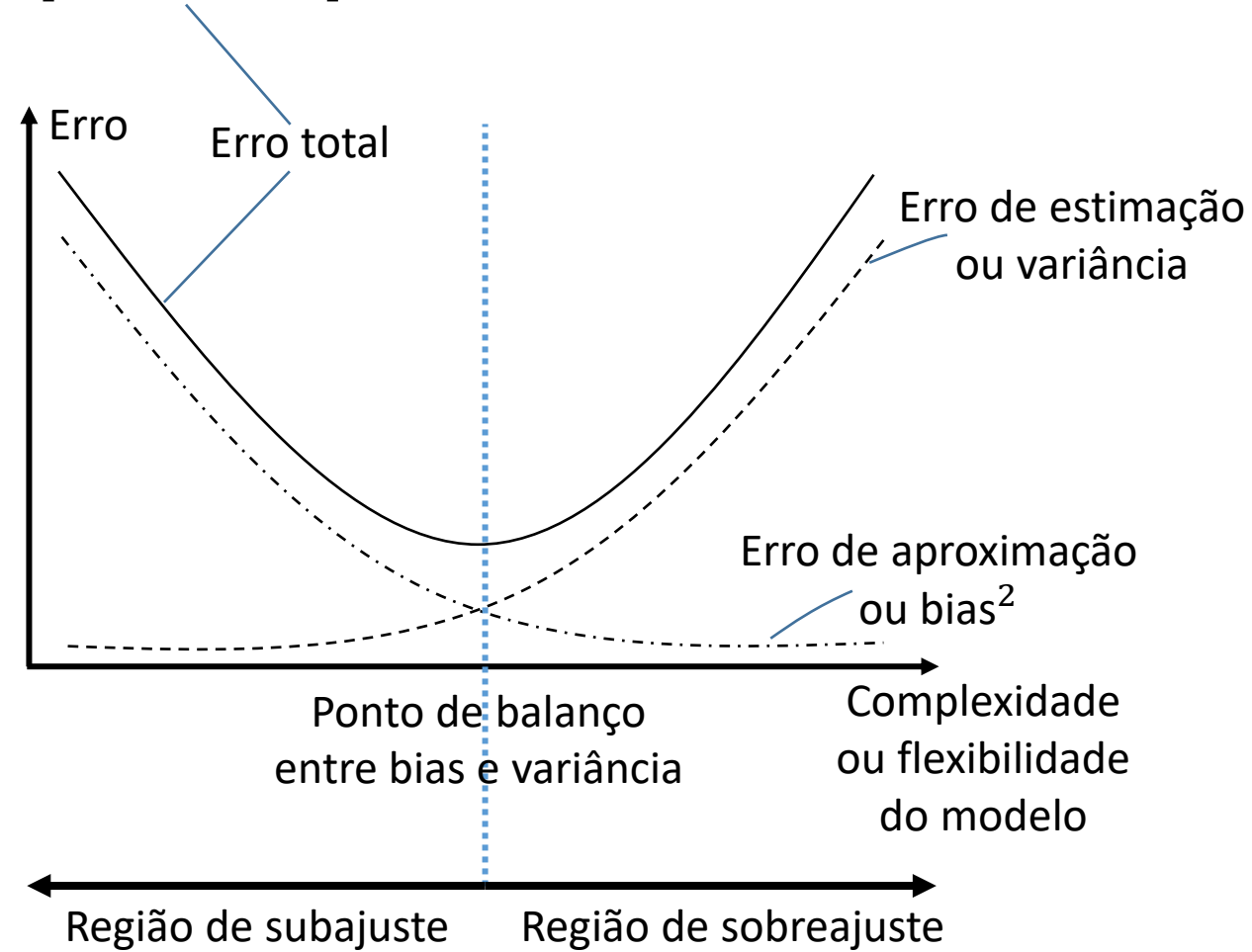


	Total de dados				
Treinamento 1	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Treinamento 2	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Treinamento 3	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Treinamento 4	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Treinamento 5	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5

Treinamento

Validação

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



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

