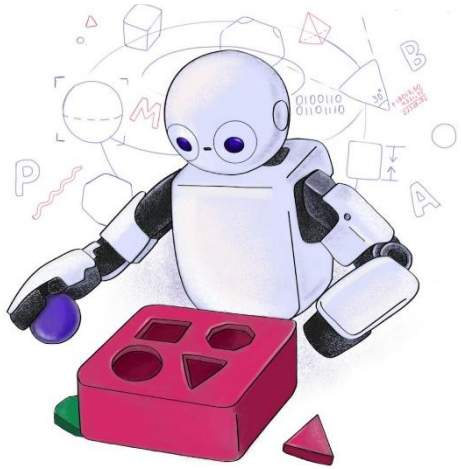
TP555 - Inteligência Artificial e Machine Learning:

Regressão para Modelos Não-Lineares

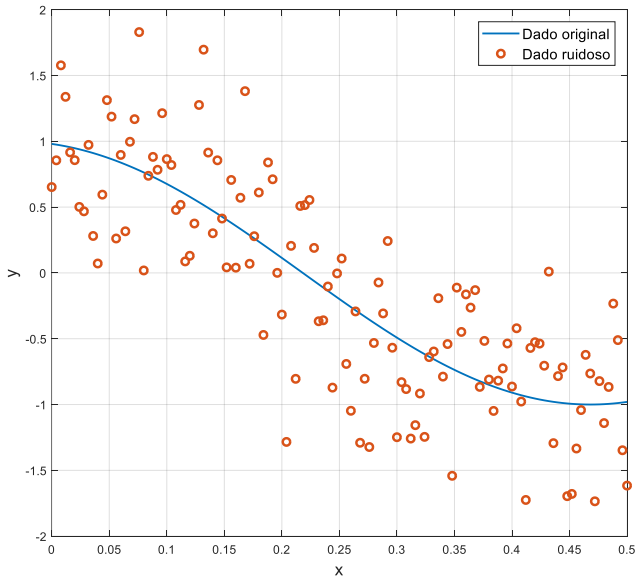


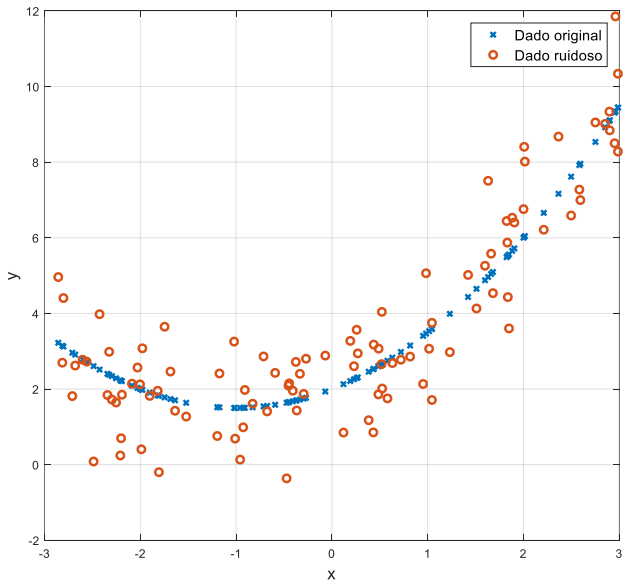
Inatel

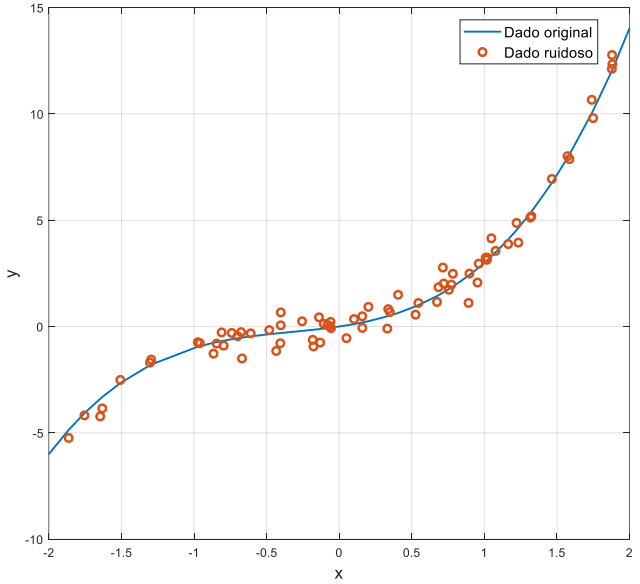
Felipe Augusto Pereira de Figueiredo
felipe.figueiredo@inatel.br

Motivação

- E se os dados tiverem um formato mais complexo do que uma simples linha reta?
- Como vocês fariam para encontrar um modelo que aproxime as funções abaixo?
- Uma reta claramente não seria uma boa escolha.







Extensão para modelos não-lineares nos atributos

- Toda a teoria que vimos até agora para ***regressão linear*** também se aplica quando o modelo de aproximação, ou seja, a ***função hipótese***, é ***não-linear*** em relação aos atributos.
- ***Modelos não-lineares*** são modelos que realizam um mapeamento ***não-linear*** das entradas (i.e., atributos) para a saída, mas que possuem ***dependência linear*** com relação aos pesos.
- ***Modelos não-lineares*** constroem uma aproximação por meio da combinação linear de ***funções-base*** não-lineares, da forma

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

  onde $b_m(\boldsymbol{x})$: $\mathbb{R}^K \rightarrow \mathbb{R}$, 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 $\Phi(i)$ como o vetor de ***funções-base***

$$\Phi(i) = \left[1, b_1\left(\boldsymbol{x}(i)\right), \ldots, b_M\left(\boldsymbol{x}(i)\right)\right]^T.$$

Linearização

- A maioria dos processos e modelos físicos não são lineares (e desta vez em relação aos pesos também), o que dificulta, na maioria das vezes, sua interpretação.
- No entanto, existem alguns ***modelos não-lineares*** que na verdade são chamados de ***intrinsecamente lineares*** porque podem ser tornados lineares em relação aos pesos através de uma simples transformação. 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, por exemplo, 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.

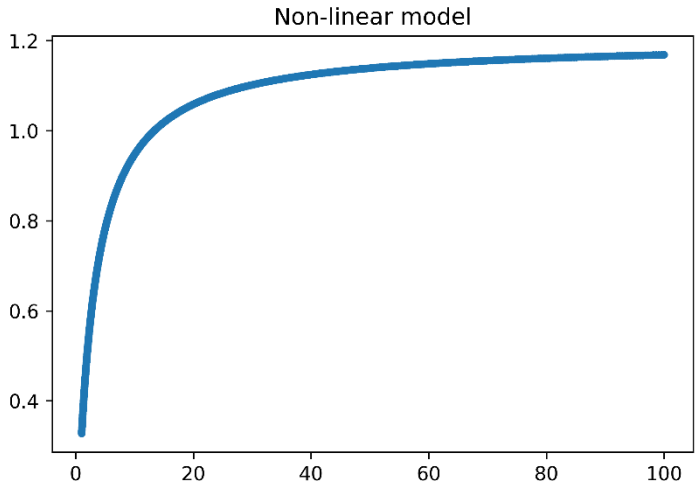
Exemplo: Linearização

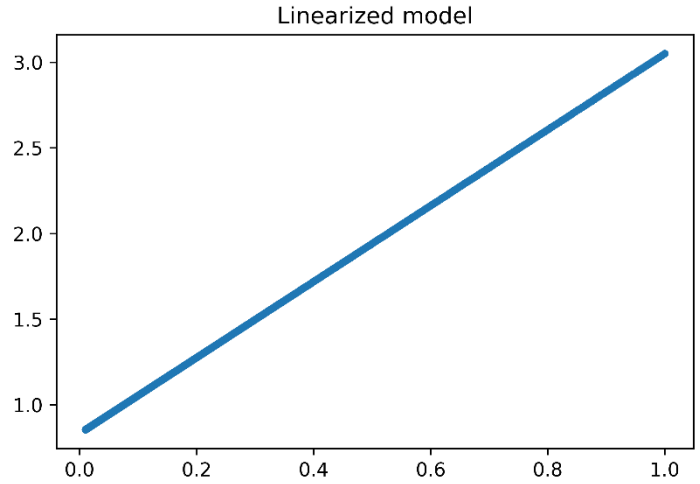
x1 = logspace(0,2,M)

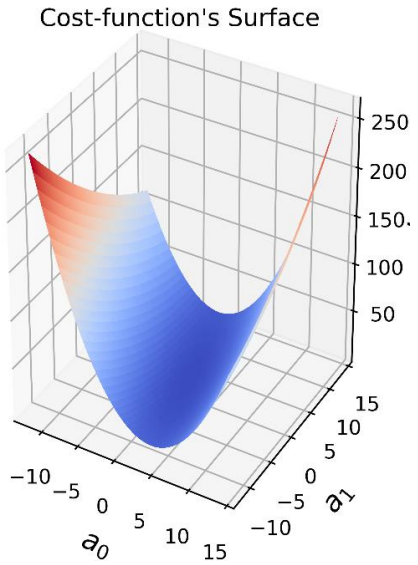
# Non-linear model.
y = (a0*x1) / (a1 + x1)

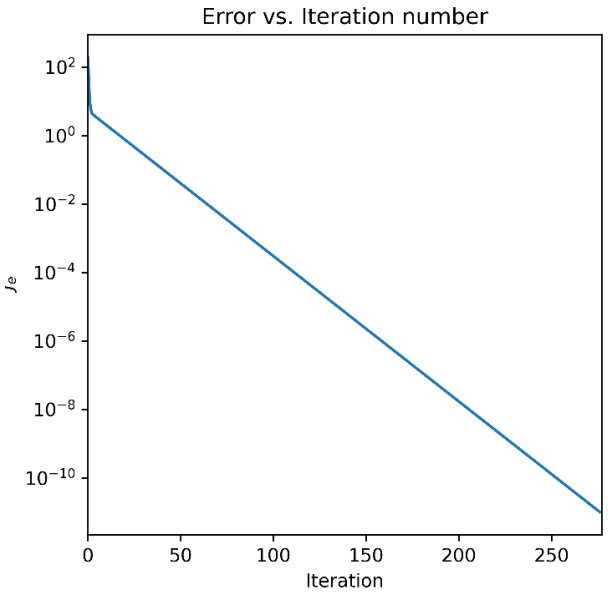
# Linearized model.
y_lin = 1/y = (1/a0) + (a1/a0)*(1/x1)
= a0' + a1' * x1'

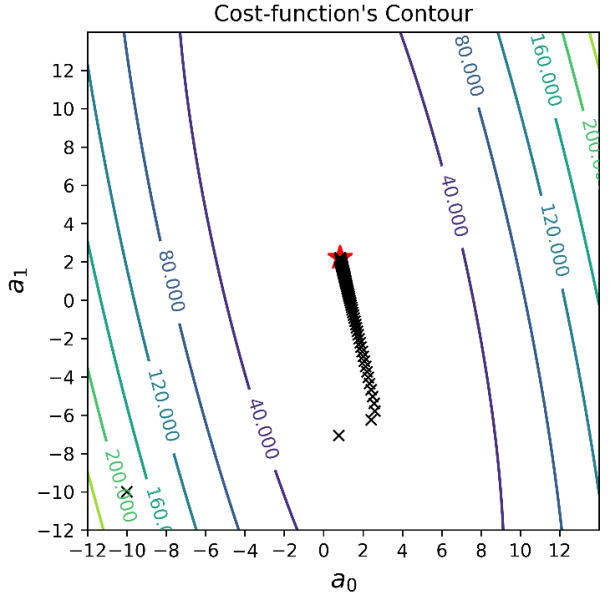
Depois do modelo ser linearizado, os pesos podem ser encontrados com a equação normal ou com o gradiente descendente.











Exemplo: model_linearization.ipynb

Regressão Polinomial

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

$$y(\boldsymbol{x}) = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_1 x_3 + a_4 x_1 x_2 {x_3}^2 + a_5 {x_1}^3$$

- Por simplicidade, para nossa análise, nós vamos considerar o seguinte modelo de ***regressão polinomial em uma váriável***

$$h(\boldsymbol{x}(i)) = \hat{y}(\boldsymbol{x}(i)) = \ a_0 + a_1 x_1(i) + a_2 {x_1}^2(i) + \cdots + a_M {x_1}^M(i).$$

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

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

- *"Qualquer função contínua no intervalo fechado [a, b] pode ser uniformemente aproximada tão bem quanto desejado por um polinômio",* ***Teorema da aproximação de Weierstrass***.
- Um problema que se agrava com modelos polinomiais é a presença de ***outliers*** nos dados de treinamento. Um único ponto ***outlier*** pode ***perturbar*** o modelo obtido, ou seja, afetar o treinamento do modelo.

Exemplo: polynomial_regression_example_with_outliers.ipynb

Regressão Polinomial: Exemplo

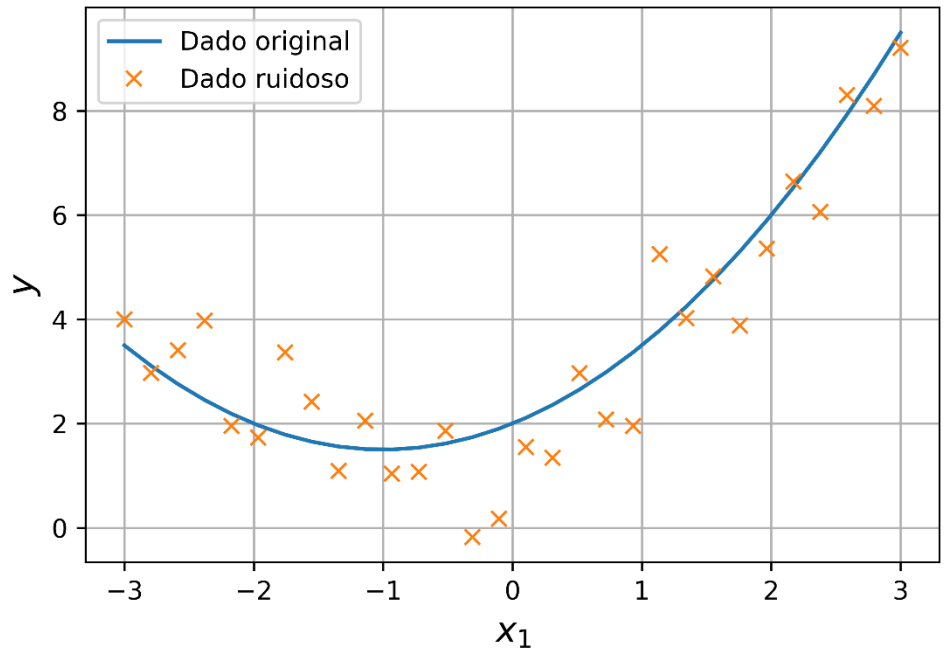
Exemplo: Geraremos 30 pontos do seguinte mapeamento, $y = f(x_1)$

$$y\big(x_1(i)\big) = 2 + x_1(i) + 0.5{x_1}^2(i),$$

e adicionaremos ruído Gaussiano branco

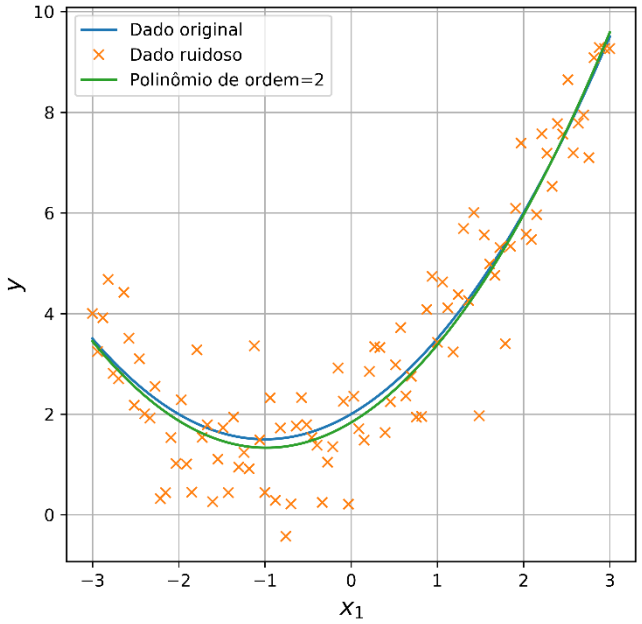
$$y_{\text{ruidoso}}\big(x_1(i)\big) = y\big(x_1(i)\big) + w(i),$$

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



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 predições.
- include_bias=**True** é usado para se incluir o vetor de **1s** dado que a função original possui um termo de bias.

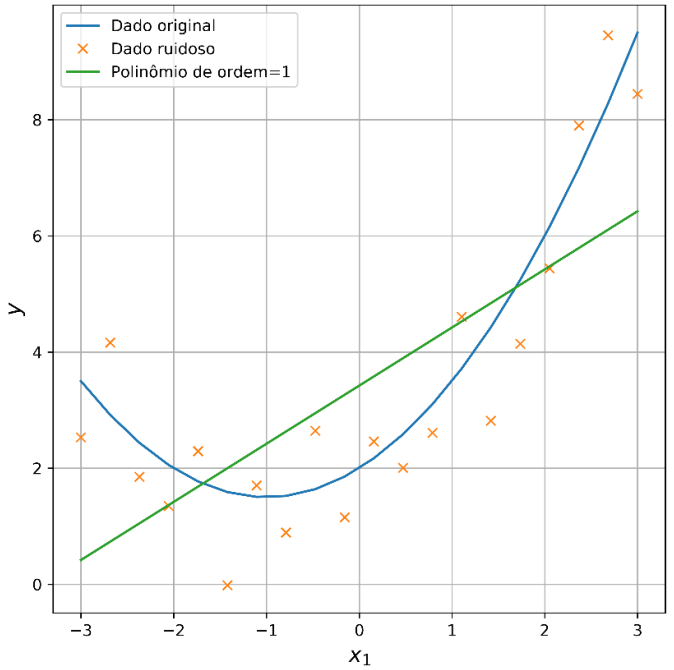
Exemplos:

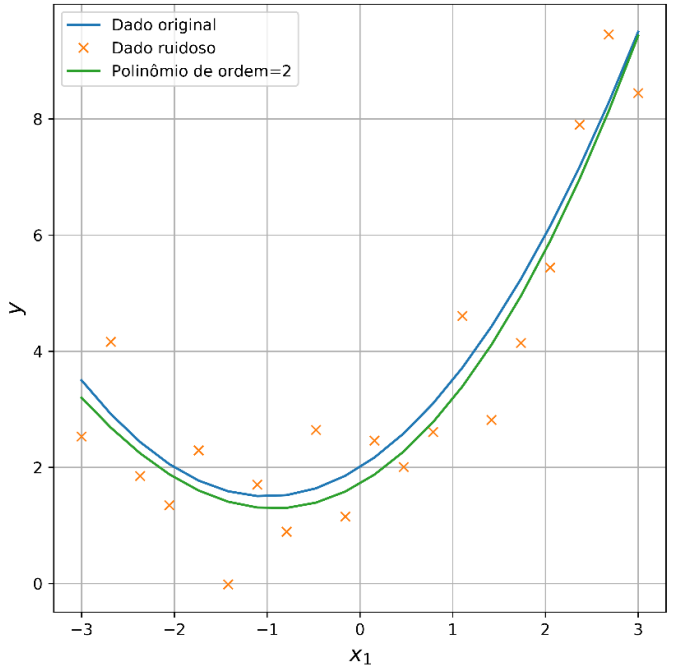
- using_poly_features_from_scikit.ipynb
- polynomial_regression_mse_calc. ipynb

Regressão Polinomial: Qual ordem usar?

Se nós não soubéssemos a ordem do polinômio gerador, qual ordem escolher para a aproximação, dado uma base de dados?

```
# Create target function.
x1 = np.linspace(-3, 3, N).reshape(N, 1)
y = 2 + x1 + 0.5*x1**2;
# Create noisy version of target function.
y_noisy = y + np.random.randn(N, 1)
```

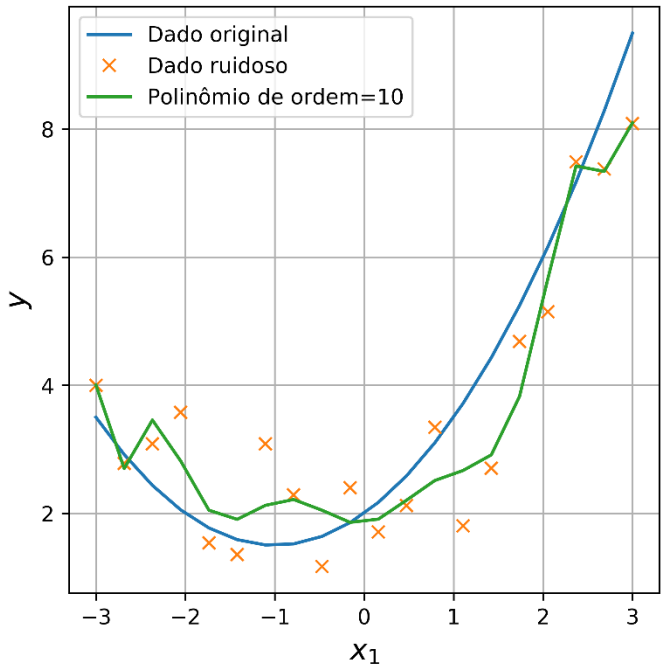


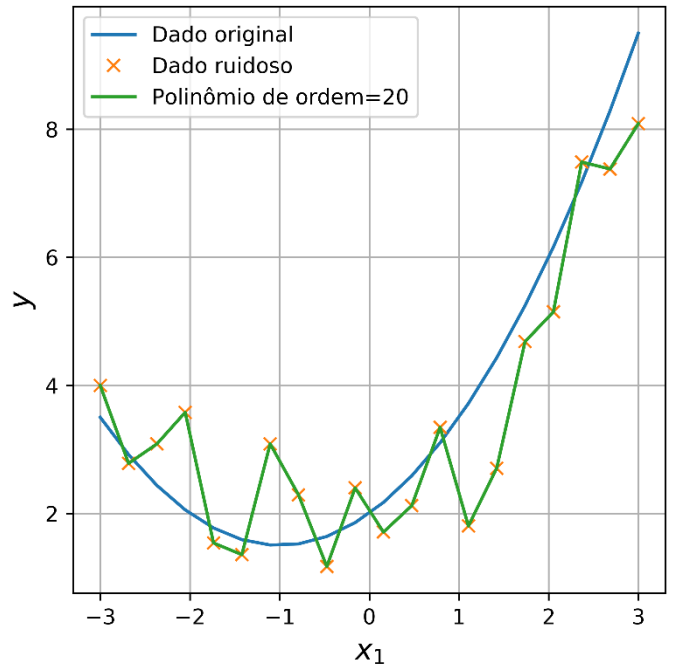


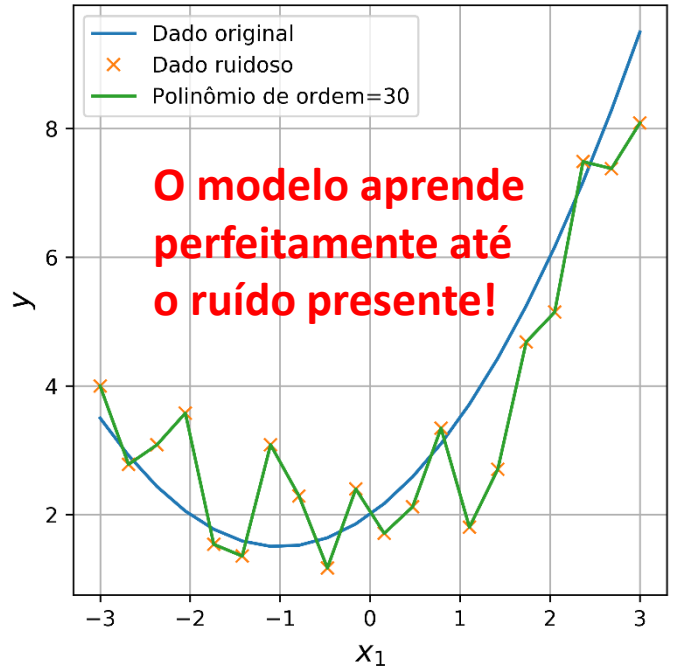
Exemplo: polynomial_regression_mse_calc.ipynb

- Observem que um polinômio de ordem 1 (i.e., uma reta) não tem flexibilidade/complexidade o suficiente para aproximar bem os dados.
- Porém, como esperado, um polinômio de ordem 2 produz a melhor aproximação dos dados.
  - Essa aproximação será melhor quanto maior for o conjunto de treinamento e/ou menor o ruído.

Regressão Polinomial: Qual ordem usar?







- Polinômios com ordem > 2 tendem a produzir aproximações perfeitas ***nos pontos disponíveis***, ou seja, o mapeamento acaba por ***ligar os pontos*** do conjunto de treinamento.
- Porém, esse mapeamento se distancia bastante da ***função original*** nas demais regiões.
- Portanto, esses modelos apresentarão ***erros significativamente maiores*** quando forem apresentados a dados novos (ou seja, dados não vistos durante o treinamento).

Observações quanto à regressão polinomial

- Através dos exemplos anteriores, percebemos que o real objetivo de se construir um modelo de regressão é o de encontrar a ***melhor aproximação possível*** do ***mapeamento verdadeiro***, i.e., $y = f(\boldsymbol{x})$, com 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 $y = f(\boldsymbol{x})$, pois em muitos casos $f(\boldsymbol{x})$, é continua e consequentemente possui infinitos valores.
- Vimos que polinômios de pequena ordem (e.g., 1) não conseguem ***aproximar*** o ***mapeamento verdadeiro***, pois não tem complexidade/flexibilidade o suficiente.
- Percebemos que é possível encontrar uma ***aproximação ótima*** para os ***exemplos de treinamento*** disponíveis que, ao mesmo tempo, produz um mapeamento, $\hat{y} = h(\boldsymbol{x})$, que difere significativamente do ***mapeamento verdadeiro***.

Observações quanto à regressão polinomial

- No caso, ao aumentarmos a ordem do polinômio, percebemos uma redução progressiva do erro de aproximação em relação aos dados utilizados no treinamento. Porém, isso não necessariamente significa que estamos, de fato, construindo um modelo melhor.
- Portanto, precisamos encontrar ***estratégias que nos forneçam indicativos*** de como o modelo se comporta ao aproximar o mapeamento verdadeiro ***como um todo*** (e não somente nas amostras de treinamento), e que também nos **auxiliem a selecionar de forma confiável** qual é o ***modelo mais adequado*** no problema de regressão.
- Uma estratégia para selecionar o modelo é comparar os erros de treinamento e validação/teste.
- Para isso, dividimos o conjunto de amostras em um conjunto usado para treinar o modelo e outro apenas para validá-lo. A ideia é que ambos tenham amostras diferentes, mas que os conjuntos sejam representativos do fenômeno sendo modelado.

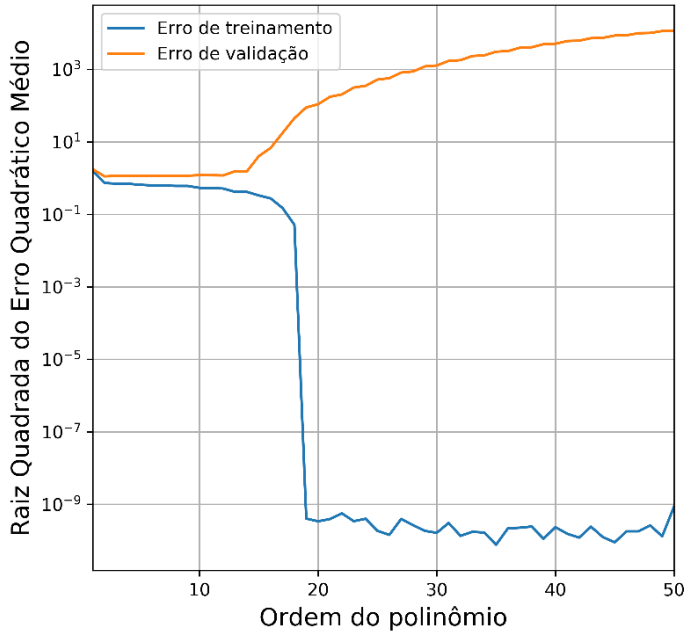
A maldição da complexidade

- 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 um valor consideravelmente pequeno para o ***erro de validação*** quando comparado com o ***erro de treinamento***.

Modelo gerador

$$y = 2 + x_1(i) + 0.5x_1^2(i) + w(i),$$

onde $w(i) \sim N(0,1)$ e $-3 \leq x_1(i) \leq 3$.




```
# Training set size.
N = 24

# Create training set.
x1 = np.linspace(-3, 3, N).reshape(N, 1)
y = 2 + x1 + 0.5*x1**2;
y_noisy = y + np.random.randn(N, 1)

# Validation set size.
N = 6

# Create training set.
x1 = np.linspace(-3, 3, M).reshape(N, 1)
y = 2 + x1 + 0.5*x1**2;
y_noisy = y + np.random.randn(N, 1)
```

Exemplo: training_error_versus_validation_error.ipynb

Capacidade e generalização de um modelo

- A ***capacidade de um modelo*** diz respeito à sua capacidade em aprender as regularidades/características dos dados do conjunto de treinamento.
  - Medida através do erro de treinamento.
- A ***generalização de um modelo*** está ligada à qualidade da aproximação produzida por um modelo quando exposto a dados de entrada não vistos durante o seu treinamento.
  - Medida através do erro de validação/teste.
- Infelizmente essas duas ***forças*** então em lados opostos, de forma que ter mais de uma geralmente implica ter menos da outra.
- O modelo ótimo será aquele que encontre uma relação de compromisso (i.e., balanço) entre a ***capacidade*** e ***generalização***.

Sobreajuste e subajuste

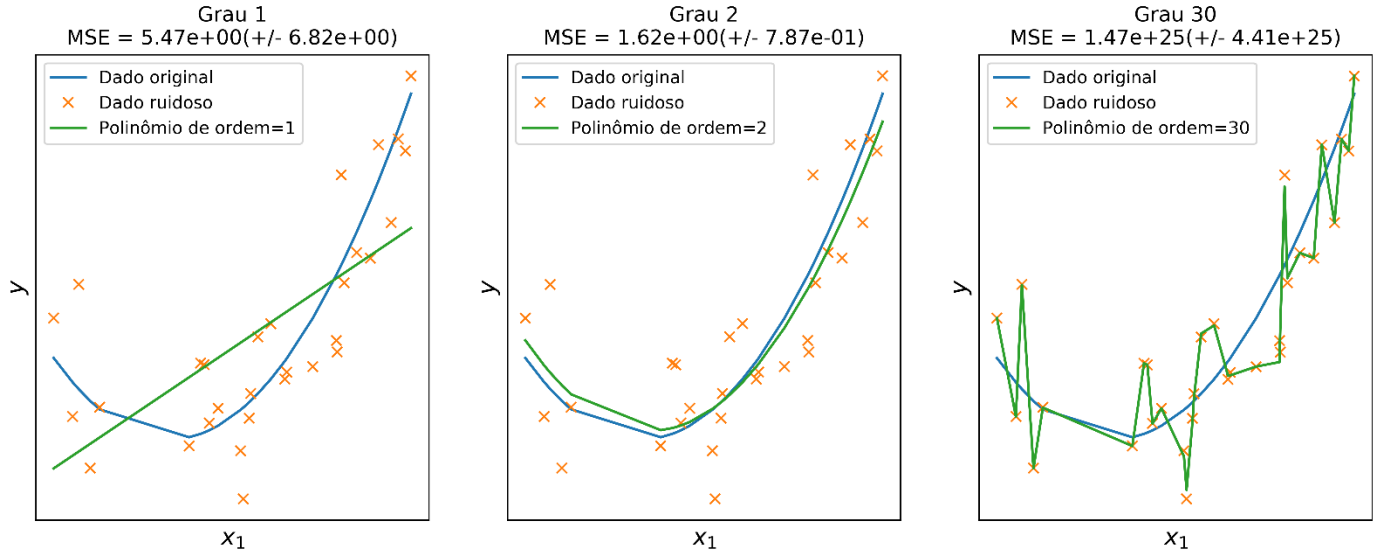
- No exemplo que estudamos anteriormente, o modelo com polinômio de grau 1 não conseguiu capturar a curvatura dos pontos de treinamento.
- Ele erra muito tanto para pontos de treinamento quanto para pontos não vistos durante o treinamento (validação).
- Esse fenônemo é chamado de ***subajuste (underfitting)*** ou ***alto-viés***.
  - **Capacidade é muito baixa, pois o modelo não tem flexibilidade o suficente.**
- Por outro lado, o modelo com polinômio de grau 30 acerta todos os pontos de treinamento mas erraria muito para pontos não vistos durante treinamento.
- Esse fenônemo é chamado de ***sobreajuste (overfitting)*** ou ***alta-variância***.
  - **Capacidade é tão alta que o modelo aprende também o ruído presente no conjunto de treinamento.**

Sobreajuste e subajuste

- A figura abaixo apresenta exemplos de subajuste, sobreajuste e ajuste ótimo.
- O modelo com polinômio de grau 2 parece se ajustar bem aos pontos, sem acertá-los perfeitamente.
- Este é provavelmente o modelo ótimo em termos da ***relação de compromisso/equilíbrio*** entre os erros de **capacidade** e de **generalização**.

Subajuste

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

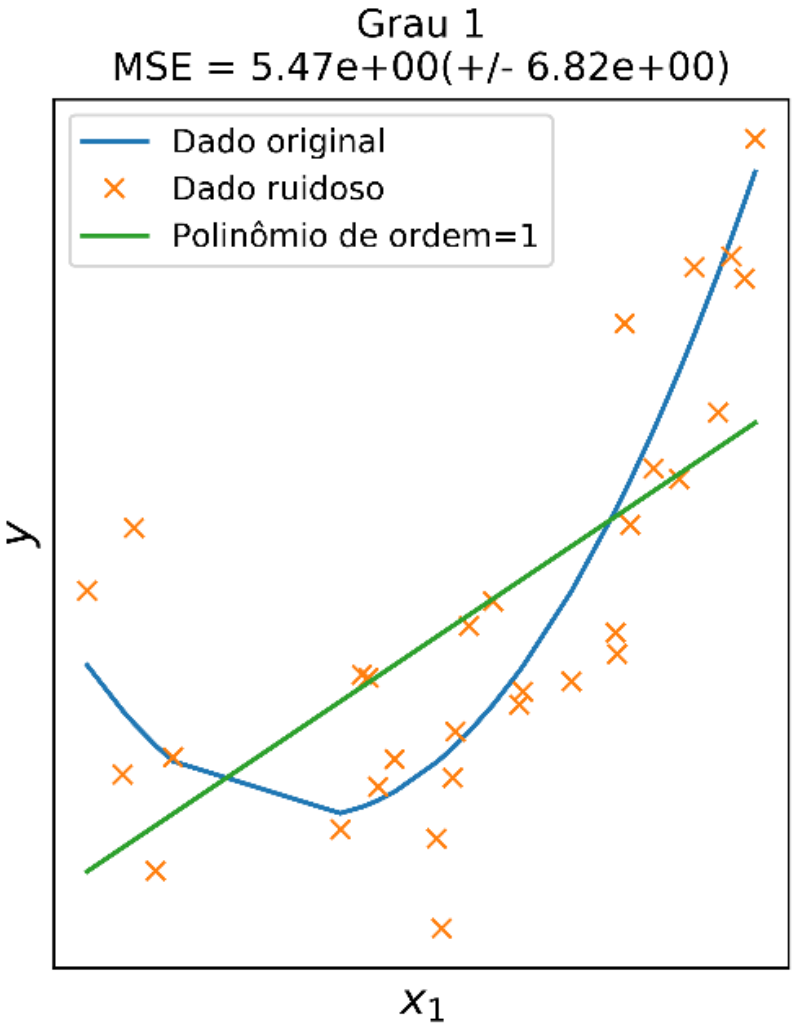


Sobreajuste

O modelo com polinômio de grau 30 tem **capacidade** muito alta e **grau de generalização** muito baixo.

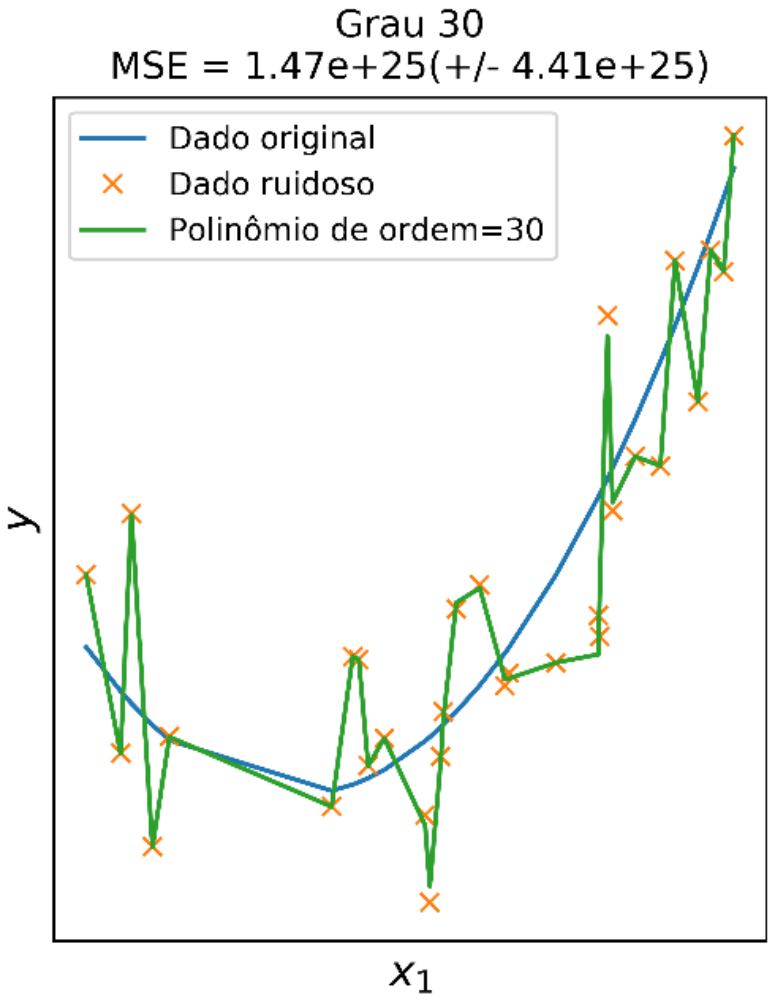
Subajuste (Resumo)

- ***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 baixo grau de complexidade do modelo ou por problemas de convergência durante o treinamento.
- Se o modelo está ***subajustando***, mesmo que o número de exemplos aumente esta sitação não vai desaparecer, é necessário aumentar a complexidade do modelo, ou seja, no caso da regressão polinomial, sua ordem.



Sobreajuste (Resumo)

- ***Sobreajuste***: é a situação em que o modelo se ajusta tão bem aos exemplos do conjunto de treinamento que ele aprende até o ruído presente nos mesmos (ou seja, o modelo apresenta baixo ***erro de treinamento***).
- Porém, o modelo produz erros significativos quando apresentado a dados inéditos (ou seja, alto erro de ***erro de generalização***).
- 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.



Erros envolvidos no processo de aproximação

- **Erro de aproximação**: é o erro devido a suposições erradas feitas sobre os dados.
  - Também conhecido como ***erro de representação*** ou de ***efeito bias*** (viés).
  - Depende do nível de complexidade do modelo usado para a aproximação.
  - Mesmo que o conjunto de treinamento contivesse infinitos exemplos do meapeamento $y = f(\boldsymbol{x})$, devido a suposição errada sobre a complexidade dos dados, não seria possível obter-se um modelo que se aproximesse do ***mapeamento verdadeiro***, $y = f(\boldsymbol{x})$.
  - Modelos com ***alto bias*** tendem a ***subajustar*** os dados de treinamento, perdendo relações importantes entre os atributos e os rótulos/objetivos.
  - Um ***bias alto*** leva a altos erros de treinamento e teste.
  - **Exemplo**: o modelo de ordem 1 supõe que o ***mapeamento verdadeiro*** pode ser modelado por uma ***reta***. O que é claramente um **erro de representação**.

Erros envolvidos no processo de aproximação

- **Erro de estimação**: é o erro devido à sensibilidade excessiva do modelo à pequenas variações nos dados de treinamento, ou seja, esse erro mede o quanto o modelo depende dos dados de treinamento.
  - Também conhecido como ***variância***.
  - Depende do nível de complexidade do modelo de aproximação.
  - Um modelo com alto grau de complexidade (como um modelo polinomial de alto grau) provavelmente apresenta alta **variância** e, portanto, ***sobreajusta*** o modelo aos dados de treinamento.
  - Em outras palavras, a alta variância faz com que modelos aprendam o ruído presente no conjunto de treinamento, resultando em baixo erro de treinamento e alto erro de validação.
  - **Exemplo**: o polinômio de ordem 30 do exemplo anterior passa por todos os pontos e não aprende o mapeamento verdadeiro.

Erros envolvidos no processo de aproximação

- **Erro irredutível**: é o erro devido ao ruído contido nos dados.
  - Também conhecido como ***erro de Bayes***.
  - É o erro que não pode ser reduzido através da criação de bons modelos.
  - Por melhor que seja nosso modelo, os dados terão uma certa quantidade de ruído que não pode ser removido.
- Exemplos:
  - polynomial_regression_mse_big_data_set.ipynb
  - bias_variance_tradeoff.ipynb
- **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.

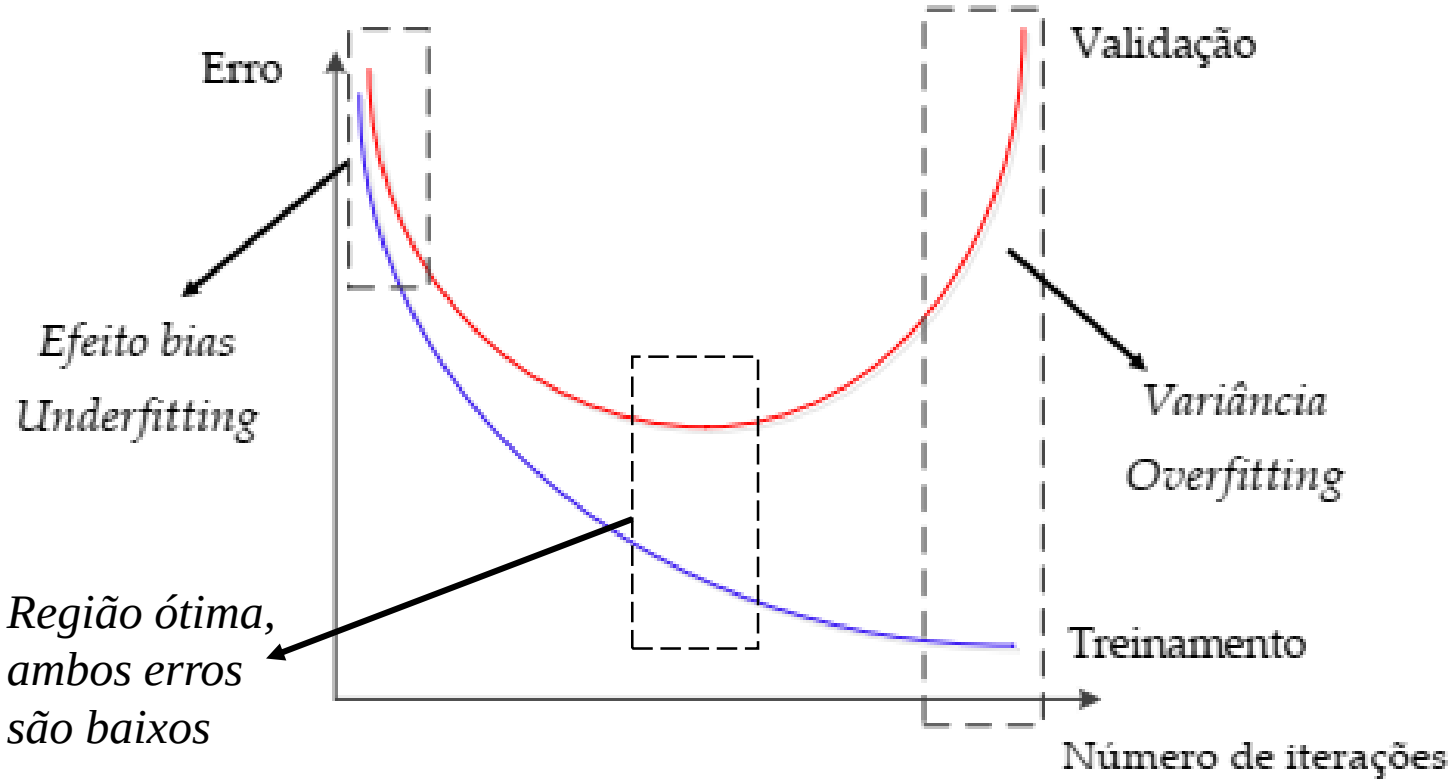
Como avaliar e escolher a melhor hipótese?

- Após todas essas informações, algumas perguntas surgem a respeito do modelo e seu treinamento.
- Como escolhemos a ***função hipótese*** quando não conhecemos o ***mapeamento verdadeiro***?
  - Ou seja, o quão complexo deve ser o modelo?
- Como podemos dizer que o modelo está ajustando demais ou insuficientemente?
- O quão bem podemos prever dados futuros resultantes do mesmo modelo gerador com um dado modelo (i.e., o quão bem podemos ***generalizar***) ?

Validação cruzada

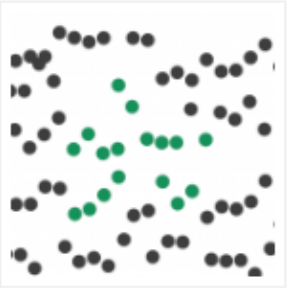
- ***Validação cruzada*** é uma das formas de se avaliar ***quantitativamente*** o sobreajuste ou subajuste de um modelo, ou seja, ***validar o modelo***.
- Na ***validação cruzada***, nós dividimos o conjunto de exemplos em 2 outros conjuntos, o de treinamento e o de validação (ou teste) do modelo.
- O objetivo é testar a capacidade do modelo em predizer valores de saída para dados que não foram utilizados durante o treinamento (conjunto de validação), ou seja, ***generalizar***.
- **OBS**.: devemos nos assegurar que os conjuntos sejam suficientemente ***representativos*** do mapeamento que se pretende aproximar.

Comportamento "*típico*" dos erros de treinamento e validação

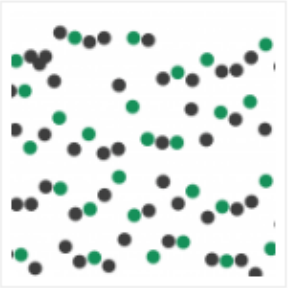


- **Subajuste ou bias**: erros de treinamento e validação tendem a serem elevados.
- **Sobreajuste ou variância**: erro de treinamento é baixo mas erro de validação é alto.

Estratégias para validação cruzada



selection bias

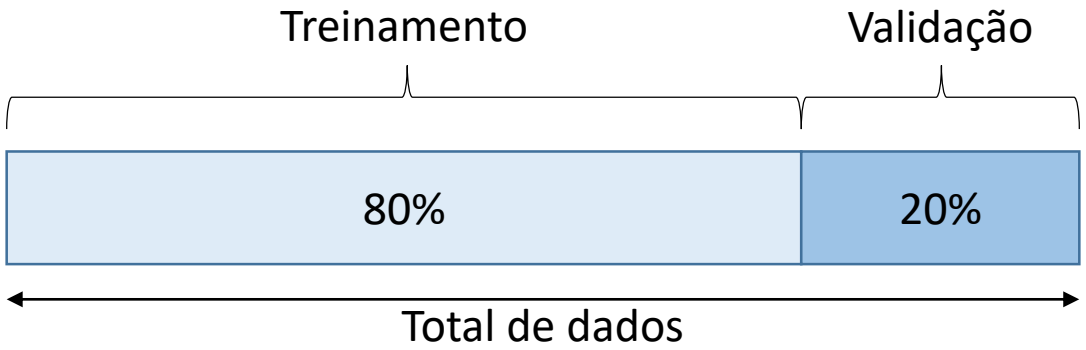


proper random sampling

- Estratégias para ***validação cruzada*** são técnicas de validação de modelos usadas para avaliar como os resultados de uma análise estatística irão generalizar para um conjunto de dados independente, i.e., dados que não foram usados durante o treinamento.
- A validação cruzada é utilizada para detectar problemas como ***sobreajuste, subajuste*** ou ***viés de seleção*** e para dar uma visão sobre como o modelo irá generalizar para um conjunto de dados independente.
  - O ***viés de seleção*** é o viés introduzido pela seleção de amostras para análise de um modelo de tal forma que este conjunto de amostras não seja representativo da população que se pretende analisar.
- Existem diversas estratégias para validação cruzada, sendo que as mais utilizadas são: ***holdout***, ***k-fold*** e ***leave-p-out***.
- Existem outras estratégias, mas todas outras podem ser vistas como variações de uma dessas três.

Holdout

- É a estratégia mais simples de validação cruzada e não acarreta em aumento da complexidade computacional, pois têm-se apenas um único par treinamento/validação.
- Nessa abordagem, divide-se ***aleatóriamente*** o conjunto de dados em ***p*** (%) para treinamento e ***(1-p)*** (%) para validação.
- Normalmente divide-se o conjunto de dados em 70/80% para treinamento e 20/30% para testes.

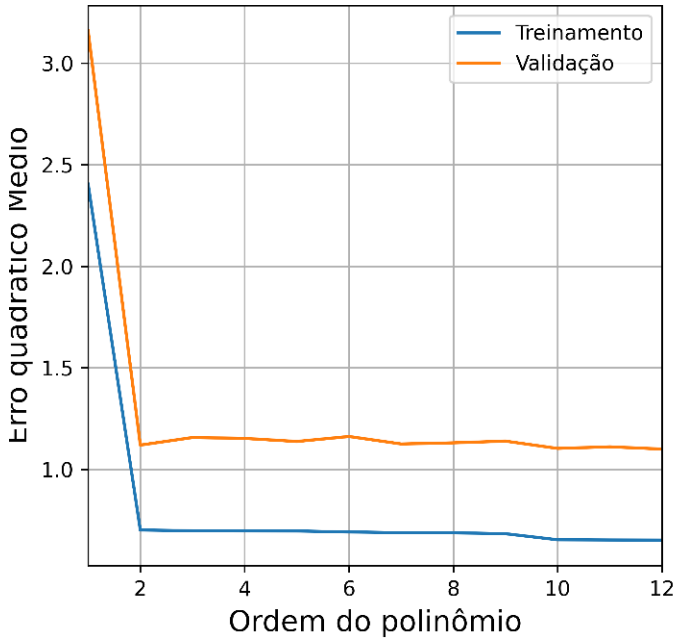


- **Desvantagem**
  - Sofre com o problema do ***viés de seleção*** o que acarreta em ***alta variância***, pois a validação pode depender muito de quais amostras vão para o conjunto de treinamento e quais vão para o conjunto de testes e, portanto, a performance pode ser significativamente diferente dependendo de como a divisão é feita, ou seja, os resultados podem depender de uma escolha aleatória particular para conjuntos de treinamento e validação.

Exemplo: holdout_and_kfold_comparison_of_shuffling.ipynb

Holdout com SciKit-Learn

```python
# Import all the necessary libraries.
import numpy as np
from sklearn.model_selection import train_test_split

# data set size.
N = 100

# Create target function and its noisy version.
x = 6*np.random.rand(N, 1) - 3
y = 0.5*x**2 + x + 2
y_noisy = y + np.random.randn(N, 1)

# Split the whole set into random training and validation set.
# 70% training set
# 30% validation set
x_train, x_val, y_train, y_val = train_test_split(x, y_noisy, test_size=0.3, random_state=10)
```

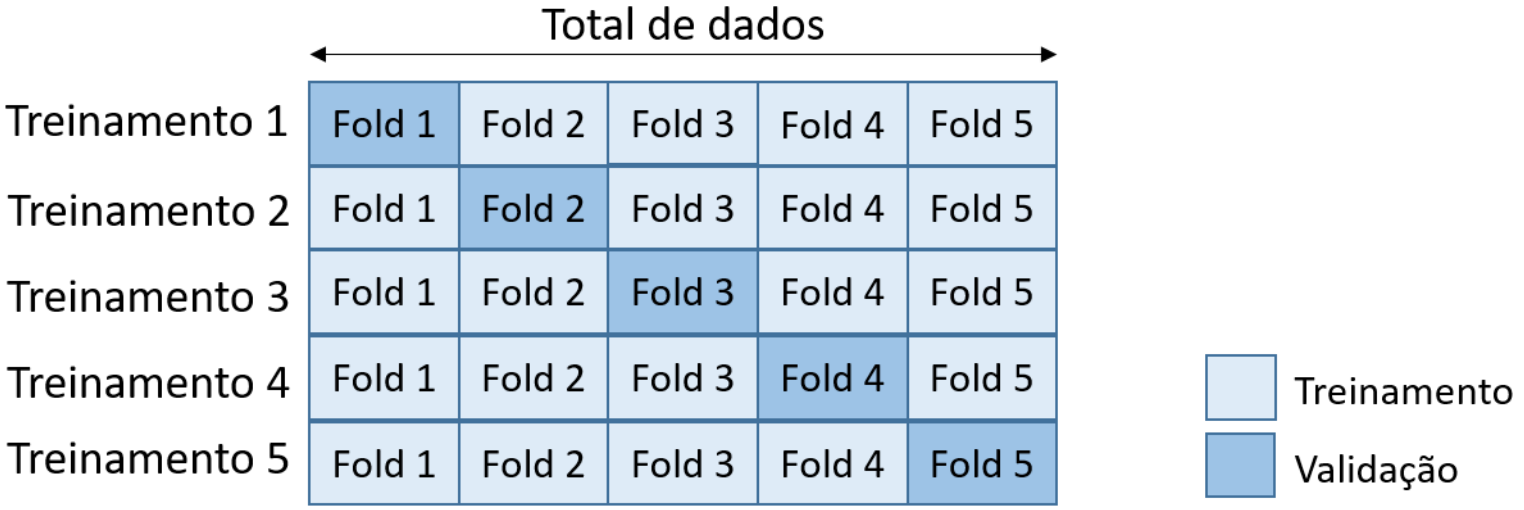


Exemplo: validacao_cruzada.ipynb

- 70% conjunto de treinamento e 30% conjunto de validação.
- Tempo médio para execução com N = 100 é de aproximadamente 40 ms.
- Erro quadrático médio diminui conforme ordem do polinômio aumenta.
- Qual ordem escolher?
  - Observando o gráfico eu diria 12, pois apresenta o menor erro.

k-Fold

- É uma estratégia mais elaborada que a anterior e que consiste em dividir o conjunto de dados em **k** folds (pastas ou dobras) de tamanhos iguais (se possível) e realizar **k** treinamentos, onde cada um dos **k** treinamentos considera **k-1** folds para treinamento e **1** fold para a 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.

Exemplo: holdout_and_kfold_comparison_of_shuffling.ipynb

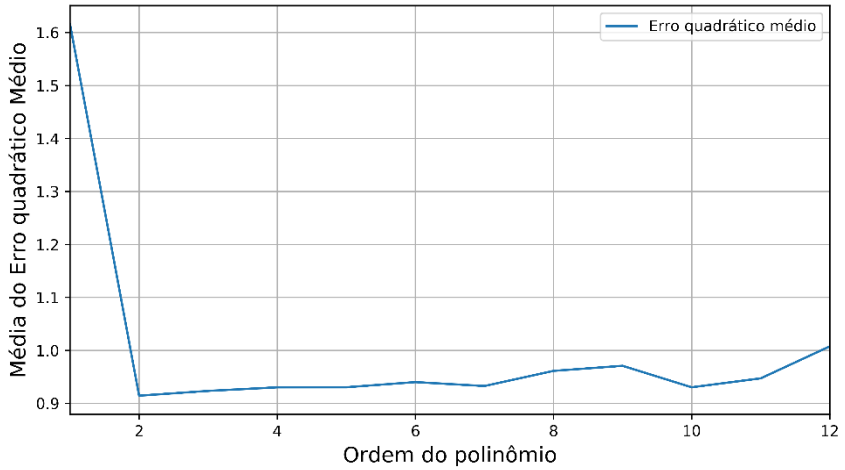
k-Fold

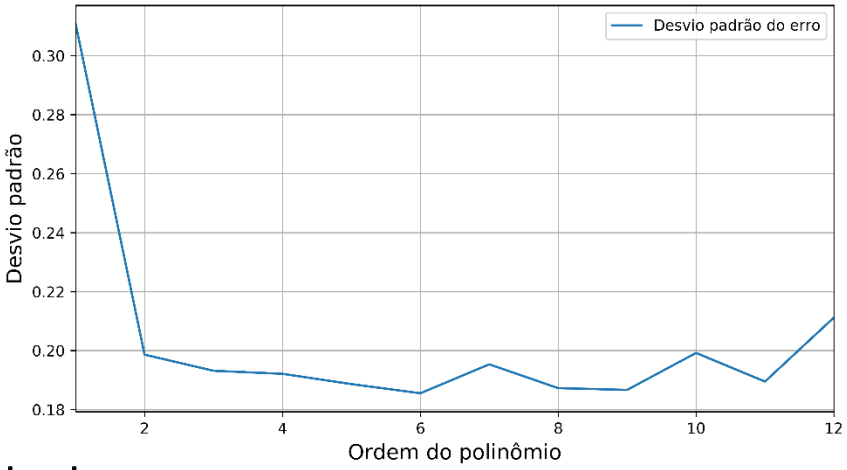
- Essa estratégia reduz significativamente o problema do ***viés de seleção*** em relação ao ***holdout***, pois ela garante que todos os exemplos do conjunto de dados original tenham a chance de aparecer nos conjuntos de treinamento e validação, o que consequentemente, reduz a ***variância*** do modelo.
- Como regra geral e evidência empírica, normalmente, utiliza-se **k** = 5 ou 10. Porém, tenha em mente que o valor de ***k*** é escolhido de forma que cada conjunto de treinamento e validação seja grande o suficiente para ser ***estatisticamente representativo*** do conjunto de dados original.
- A ***variância*** da estimativa resultante é reduzida à medida que **k** é aumentado.
- Caso **k=N**, ou seja, o número folds for feito igual ao número total de exemplos do conjunto original, então a estratégia passa a ser conhecida como validação cruzada **leave-one-out**.
- **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 com SciKit-Learn

```
# Import all the necessary libraries.
import sklearn as skl
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

# Instantiate the k-Fold object.
kfold = KFold(n_splits=10, shuffle=True, random_state=100)

for d in range(1, 13):
    polynomial_regression = skl.pipeline.Pipeline([
        ("poly_features", skl.preprocessing.PolynomialFeatures(degree=d, include_bias=True)),
        ("std_scaler", skl.preprocessing.StandardScaler()),
        ("lin_reg", skl.linear_model.LinearRegression()),
    ])

    lin_scores = cross_val_score(polynomial_regression, x, y_noisy, scoring='neg_mean_squared_error', cv=kfold)
    scores = np.sqrt(-lin_scores)
    mean_vec.append(scores.mean())
    std_vec.append(scores.std())
```

Exemplo: validacao_cruzada.ipynb





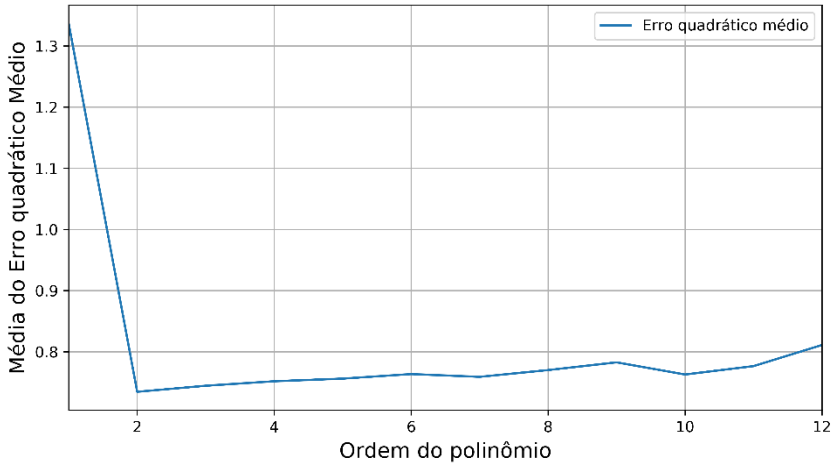
- **k** = 10 folds: 10 iterações com 9 grupos para treinamento e 1 para teste.
- shuffle=**True**: os exemplos são “embaralhados” antes de dividí-los em **k** folds.
- Cada ponto no gráfico é a média do erro quadrático médio para as 10 iterações.
- Tempo médio para execução com M = 100 é de aproximadamente 430 ms.
- Qual ordem escolher?
  - 2? Pelo erro sim, mas pelo desvio padrão não.

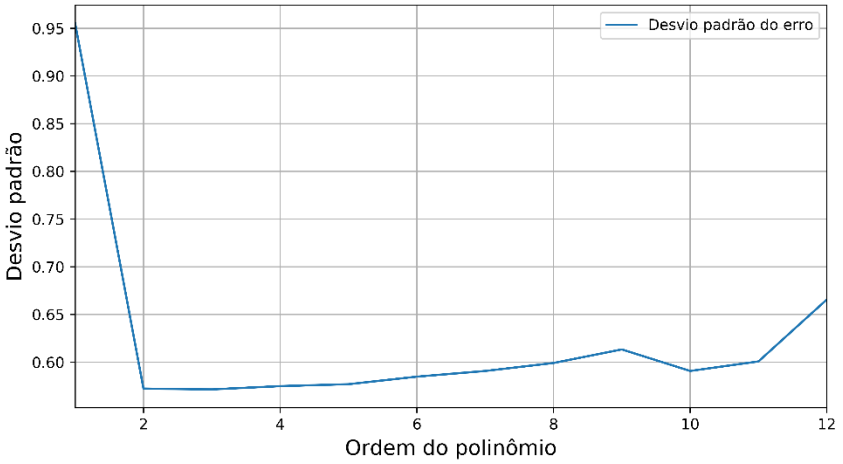
Leave-p-out

- Essa estratégia 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 $\binom{N}{p} = \frac{N!}{p!(N-p)!}$ pares de conjuntos trainamento-teste, portanto, a complexidade computacional desta estratégia aumenta drasticamente com o aumento de ***p***. Exemplos para $N = 100$:
  - p = 1 -> 100 combinações
  - p = 2 -> 4.950 combinações
  - p = 5 -> 75.287.520 combinações
- Fornece estimativas de erro mais precisas do que as abordagens anteriores.
- **Desvantagem**
  - É uma estratégia exaustiva no sentido de que precisa treinar e validar o modelo para todas as combinações possíveis e, para uma base de dados grande e um valor de **p** moderadamente grande, pode se tornar inviável computacionalmente.
- Caso **p=1**, então temos novamente a abordagem ***leave-one-out***.
- No caso do k-Fold, quando k=N, então temos que o k-Fold é equivalente à estratégia do leave-one-out.

Leave-p-out com SciKit-Learn

```
# Import all the necessary libraries.
import numpy as np
import sklearn as skl
from sklearn.model_selection import LeavePOut
from sklearn.model_selection import cross_val_score

# Instantiate the LPOCV object.
lpocv = LeavePOut(p=1)

for d in range(1, 13):
    polynomial_regression = skl.pipeline.Pipeline([
        ("poly_features", skl.preprocessing.PolynomialFeatures(degree=d, include_bias=True)),
        ("std_scaler", skl.preprocessing.StandardScaler()),
        ("lin_reg", skl.linear_model.LinearRegression()),
    ])

    lin_scores = cross_val_score(polynomial_regression, x, y_noisy, scoring='neg_mean_squared_error', cv=lpocv)
    scores = np.sqrt(-lin_scores)
    mean_vec.append(scores.mean())
    std_vec.append(scores.std())
```





Exemplo: validacao_cruzada.ipynb

- **p** = 1: 100 combinações possíveis com 99 exemplos para treinamento e 1 para teste.
- Cada ponto no gráfico é a média do erro para as 100 combinações.
- Tempo médio para execução com M = 100 é de aproximadamente 3.30 s.
- Qual ordem escolher? 2!

Comparação

- Conforme nós vimos, o **leave-p-out** dá indicações mais claras de qual ordem usar, pois usa um maior número de pares treinamento/teste, aumentando a confiabilidade tanto do erro médio quanto do desvio padrão.
- Porém, ele é bastante custoso em relação ao tempo necessário para se executá-lo, mesmo com uma base de 100 amostras leva-se mais de 3 [s].
- Portanto, é interessante utilizá-lo com bases relativamente pequenas.
- Para bases maiores, o **k-fold** é uma opção melhor e mais eficiente do que o **holdout**.
- Bases maiores ainda já dariam melhores indicações de qual modelo utilizar mesmo com o **holdout**.

Exemplo: validacao_cruzada_base_maior.ipynb

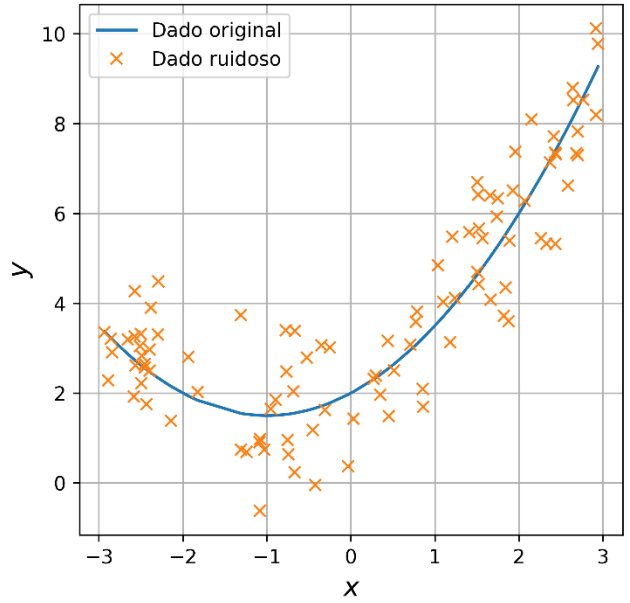
Qual ordem escolher para o modelo?

- E se os erros são pequenos para vários graus de polinômio, incluindo altos graus?
- Uma resposta é usar o princípio da ***navalha de Occam***.
- A ***navalha de Occam*** é um princípio lógico que diz: "*as entidades não devem ser multiplicadas além da necessidade*“.
  - Deve-se preferir explicações mais simples às mais complicadas.
- ***Occam*** costumava argumentar contra todos os tipos de complicações.
- Portanto, escolhemos modelos usando a **navalha de Occam**: escolhemos a ***hipótese*** menos complexa que se ajusta bem aos dados.

Curvas de Aprendizado

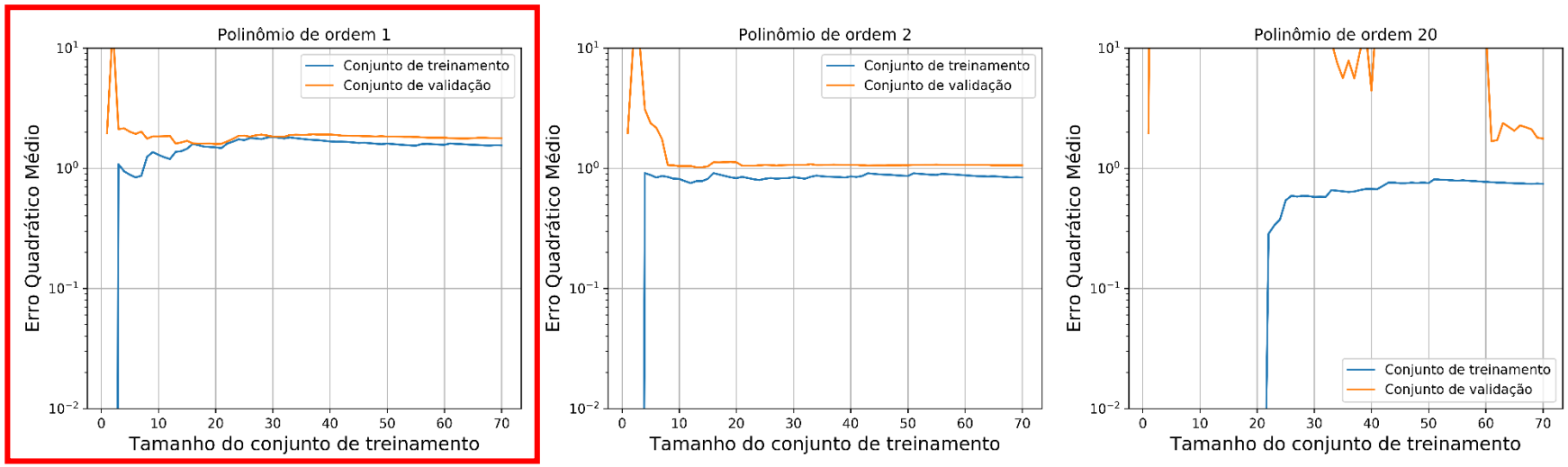
- **Curvas de aprendizado**: são gráficos que comparam o desempenho do modelo no conjunto de treinamento e no conjunto de validação em função do tamanho do conjunto de treinamento.
  - Tamanho do conjunto de validação permanece constante.
  - Caso não tenhamos dois conjuntos, treinamento e validação, normalmente se divide a base original em 70 a 80% para treinamento e 30 a 20% para validação.
- Essa comparação normalmente é usada para avaliar:
  - Qual é o melhor nível de complexidade (no caso de polinômios, sua ordem) um modelo deve ter. Em outras palavras, avalia-se se o modelo é mais sensível ao erro de variância (sobreajuste) ou ao erro de viés (subajuste).
  - O quanto o modelo se beneficia de mais dados (por exemplo, se temos "dados suficientes" ou se o desempenho modelo ficará melhor se usado de forma online).

Curvas de Aprendizado: Exemplo



- Caso não conhecêssemos o ***mapeamento verdadeiro*** dos dados ruidosos da figura, qual ordem de polinômio melhor aproximaria o ***mapeamento verdadeiro***?

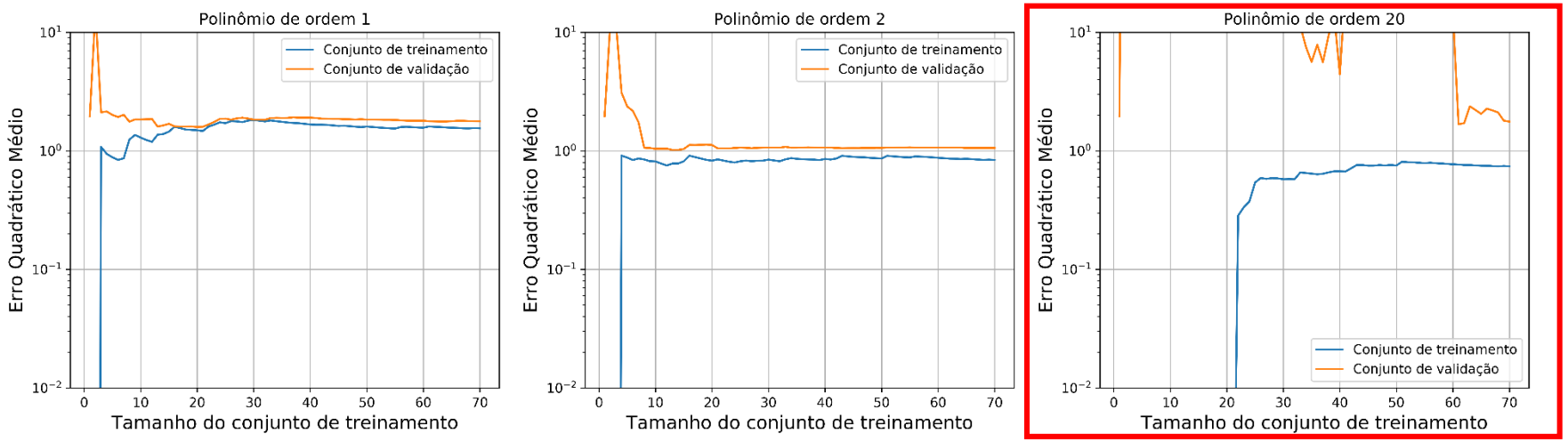
Curvas de Aprendizado: Polinômio de ordem 1



- 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 ele aumenta, terminando em um platô próximo do erro de treinamento.
- Essas curvas são típicas de um modelo que está ***subajustando***.
- **O que fazer?** Aumentar a ordem do modelo.

Exemplo: curvas_de_aprendizado.ipynb

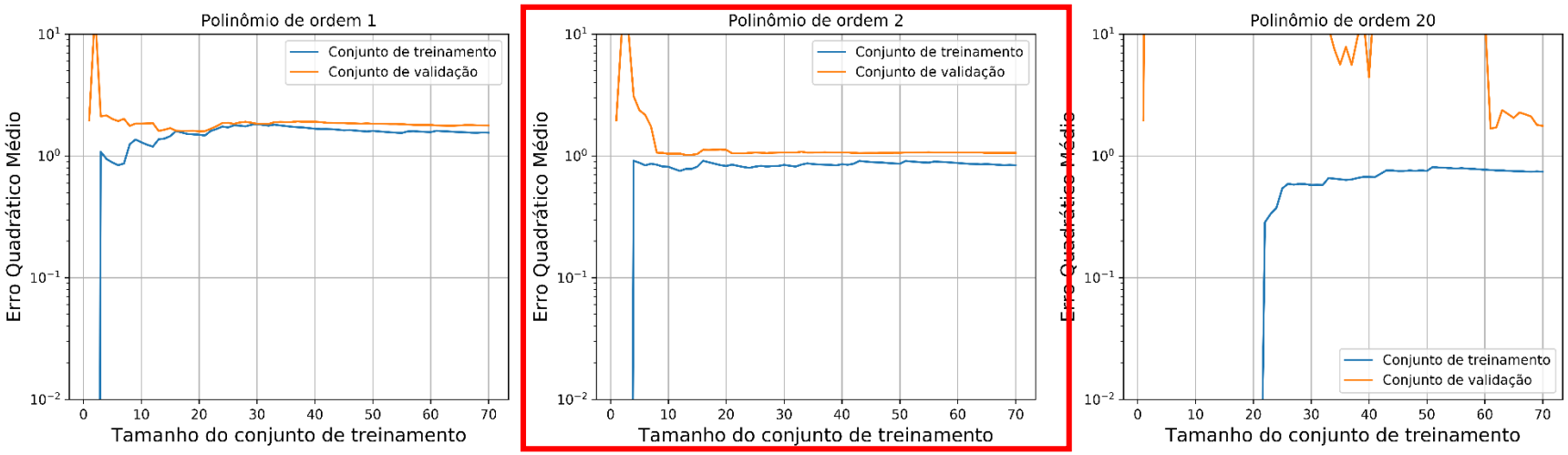
Curvas de Aprendizado: Polinômio de ordem 20



- 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 o modelo está ***sobreajustando***.
- A performance do modelo melhora caso o conjunto de treinamento seja aumentado.
- 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.

Exemplo: curvas_de_aprendizado.ipynb

Curvas de Aprendizado: Polinômio de ordem 2



Exemplo: curvas_de_aprendizado.ipynb

- 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 de regressão linear, i.e., reta.
- Isso é a indicação de um modelo que está se ajustando bem aos dados de treinamento 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

- A **regularização** é outra forma de se escolher o melhor modelo.
- A ideia por trás da ***regularização*** é penalizar, explicitamente, ***hipóteses*** complexas.
- As técnicas que veremos a seguir, são chamadas de técnicas de **regularização**, pois buscam por uma ***hipótese*** que seja mais ***regular***, ou seja, ***menos complexa***.
- Ela é uma técnica que visa obter um modelo bem comportado (ou seja, que não ***sobreajuste***) através da incorporação de informações adicionais ao processo de treinamento do modelo, na forma de ***restrições de suavidade*** junto ao mapeamento ou de ***penalizações*** proporcionais à norma do vetor de pesos.
- Técnicas de ***regularização*** podem reduzir o risco de ***sobreajuste*** do modelo ao conjunto de treinamento, aumentando sua capacidade de ***generalização***.
- As técnicas de ***regularização*** que veremos a seguir são: *rigde regression*, LASSO e *elastic net*.
- Todas elas introduzem ***restrições*** ligadas a alguma ***norma*** do vetor de ***pesos*** do modelo, sendo, por este motivo, também conhecidas como técnicas de ***shrinkage*** (i.e., redução, encolhimento).

Ridge regression

- Com a regressão de Ridge, ao invés de minimizarmos apenas o erro quadrático médio, como fizemos antes, agora há a introdução de um ***termo de penalização*** proporcional à ***norma Euclidiana*** (ou seja, a norma L2) do vetor de pesos:

$$\min_{\boldsymbol{a}\in\mathbb{R}}(\|\boldsymbol{y} - \boldsymbol{\Phi a}\|^2 + \lambda\|\boldsymbol{a}\|_2^2),$$

onde $\lambda \geq 0$ é o ***coeficiente ou fator de regularização/flexibilidade***, $\boldsymbol{\Phi}$ é a matriz de atributos e $\boldsymbol{a}$ é o vetor de pesos.

- O aumento na flexibilidade de um modelo é representado pelo aumento de seus pesos e, se quisermos minimizar a função de erro, esses pesos precisam ser pequenos.
- **OBS.**: normalmente, o coeficiente a_0 não é levado em consideração no cálculo do ***termo de penalização*** ($\|\boldsymbol{a}\|_2^2$), para evitar que o resultado dependa da origem escolhida para o vetor y. Assim, a norma L2 é dada por

$$\|\boldsymbol{a}\|_2^2 = \sum_{i=1}^{K} {a_i}^2.$$

Início em 1 e não em 0.

O peso a_0 fornece apenas o deslocamento da função hipótese em relação ao eixo y e não tem influência na complexidade da ***hipótese*** pois não é multiplicado por nenhum atributo. Lembre-se que a complexidade se deve à ordem do modelo e a_0 apenas dita o *offset* (deslocamento) em relação ao eixo y.

Exemplo: absolute_value_of_weights_when_model_overfits.ipynb

Ridge regression

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

$$\min_{\boldsymbol{a}\in\mathbb{R}} \|\boldsymbol{y} - \boldsymbol{\Phi a}\|^2$$
$$\text{s.}\,a.\ \|\boldsymbol{a}\|_2^2 \leq c,$$

onde c restringe a magnitude dos pesos e é inversamente proporcional à λ.

- Observe que há a imposição de uma restrição à norma Euclidiana do vetor de pesos.
  - Conforme o valor de c diminui, maior é a redução da magnitude dos pesos, até o limite em que $c \to 0$, quando $a_i \to 0$.
  - Por outro lado, conforme c aumenta, maior será a magnitude dos pesos, até que $c \to \infty$, quando $a_i \to \infty$.
  - Portanto, o parâmetro c (e consequentemente λ) controla, o balanço entre a redução do erro de aproximação e a limitação da magnitude dos pesos.

Ridge regression

- Perceba que a equação $\|\boldsymbol{y} - \boldsymbol{\Phi a}\|^2 + \lambda\|\boldsymbol{a}\|_2^2$ 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***.

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

- **OBS.1**: mesmo que a matriz $\boldsymbol{\Phi}$ não possua ***posto completo***, a inversa na equação acima sempre existe por conta da adição do ***termo de regularização*** à diagonal principal.
- **OBS.2**: como a **norma L2** é diferenciável, os problemas de aprendizagem usando a regularização de Ridge podem ser resolvidos iterativamente através do ***algoritmo do gradiente descendente***.
- **OBS.3**: o termo de regularização, λ, 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

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

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

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

$$\frac{\partial J_e(\boldsymbol{a})}{\partial a_k} = \frac{-2}{N}\sum_{i=0}^{N-1}[y(i) - \hat{y}(i)]\, x_k(i)\ ,\ k = 0$$

Gradiente com relação ao peso de bias

$$\frac{\partial J_e(\boldsymbol{a})}{\partial \boldsymbol{a}} = \frac{-2}{N}\boldsymbol{\Phi}^T(\boldsymbol{y} - \boldsymbol{\Phi a}) + 2\lambda \boldsymbol{I'a},\ \forall k$$

Equação geral do vetor gradiente

onde $\boldsymbol{I'}$ 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.

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

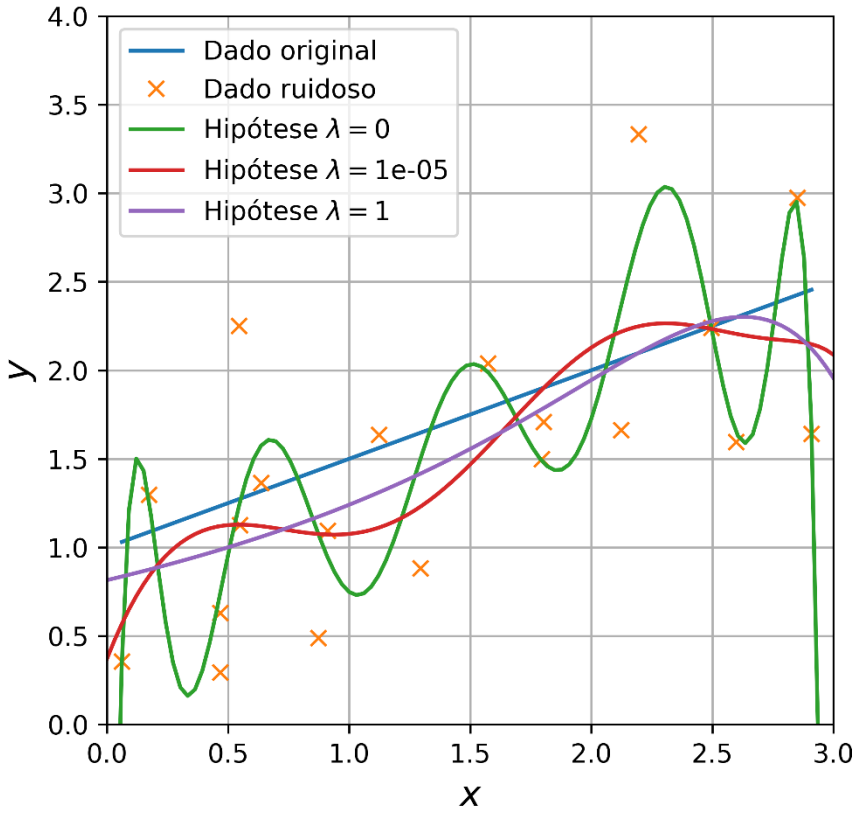
Ridge regression com a biblioteca Scikit-Learn

```
# Import all the necessary libraries.
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge
from sklearn.pipeline import Pipeline

# Sets the number of examples.
N = 20

# Create target function and its noisy version.
x = np.sort(3*np.random.rand(N, 1), axis=0)
y_noisy = 1 + 0.5*x + np.random.randn(N, 1)/1.5

# Instantiate a polynomial with the given degree.
poly_features = PolynomialFeatures(degree=10, include_bias=True)
# Instantiate a scaler that will standardize the features.
std_scaler = StandardScaler()
# Instantiate a Ridge regressor.
reg = Ridge(alpha=lambdas[i], solver="cholesky")

# Create a pipeline of actions.
model = Pipeline([("poly", poly_features),("scaler", std_scaler),("reg", reg),])

# Train model.
model.fit(x, y_noisy)
# Predict.
y_predicted = model.predict(x)
```

Importa classe

Polinômio de ordem 10

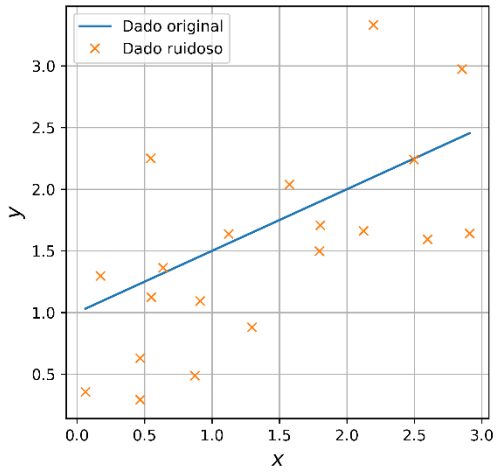
Instancia objeto



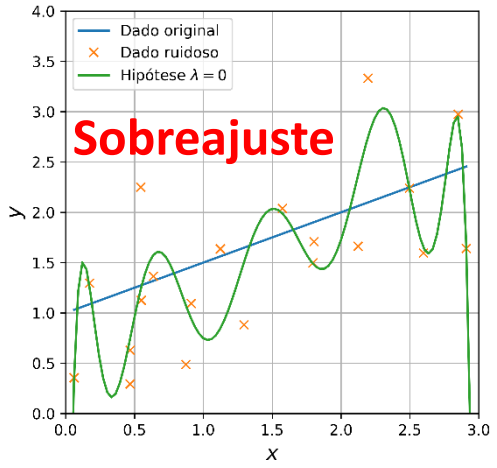
Exemplo: ridge_regressionv2.ipynb

OBS.: É importante escalonar os atributos antes de executar a regressão de Ridge, pois ela é bastante sensível à escala dos atributos.

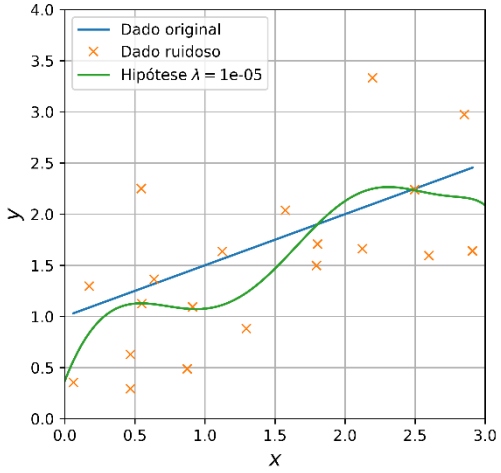
Ridge regression: Exemplo



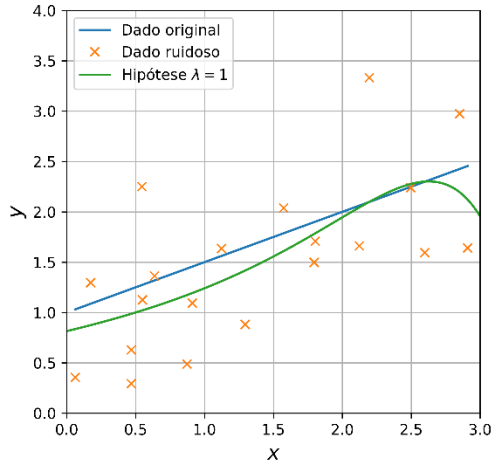
Função objetivo + ruído



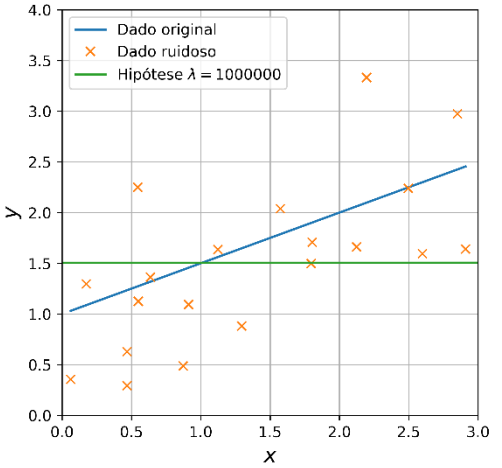
Hipótese com polinômio de ordem 10 e $\lambda = 0$.



Hipótese com polinômio de ordem 10 e $\lambda = 1\mathrm{e} - 5$.



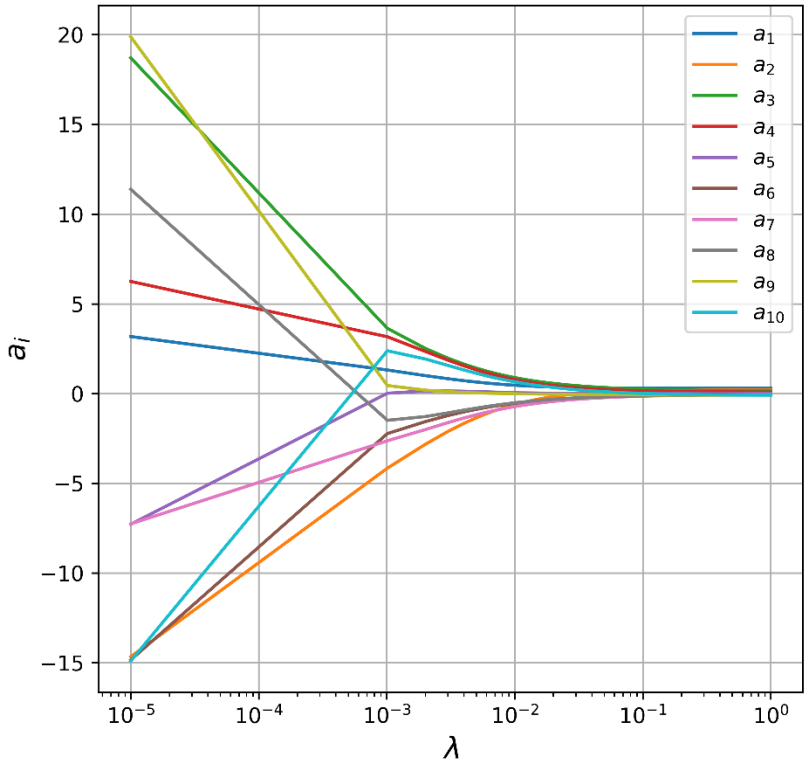
Hipótese com polinômio de ordem 10 e $\lambda = 1$.



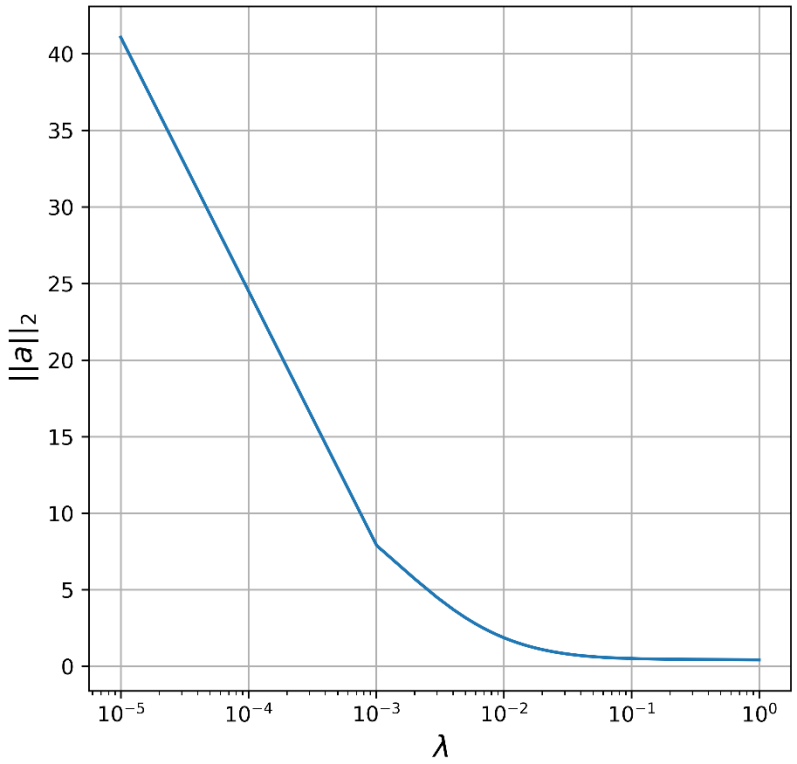
Hipótese com polinômio de ordem 10 e $\lambda = 1\mathrm{e}6$.

- Com $\lambda = 0$, regressão de Ridge é apenas uma regressão polinomial convencional, pois o termo de penalização (regularização)é anulado.
- Conforme λ aumenta, o modelo não se "*contorce*" tanto para se ajustar aos dados de treinamento.
- Se λ for muito grande, todos os pesos acabarão muito próximos de zero e o resultado será uma linha reta que passa pela ***média dos dados de treinamento***.
- O aumento de λ leva a hipóteses mais planas (ou seja, menos extremas, menos complexas); isso reduz a variância do modelo, mas aumenta seu bias. Ou seja, pode ***subajustar***.

Ridge regression: Exemplo



Evolução dos pesos em função do fator de regularização.



Evolução da norma do vetor de pesos

- Conforme λ aumenta, os pesos diminuem e consequentemente a norma do vetor de pesos.

LASSO regression

- Diferentemente da regressão de Ridge, a ***regressão LASSO*** (Least Absolute Shrinkage and Selection Operator) adiciona à função de erro um termo de penalização proporcional à **norma (L1)** do vetor de pesos.

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

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

- Também podemos re-escrever o problema de regularização acima como um problema de otimização com restrições da seguinte forma

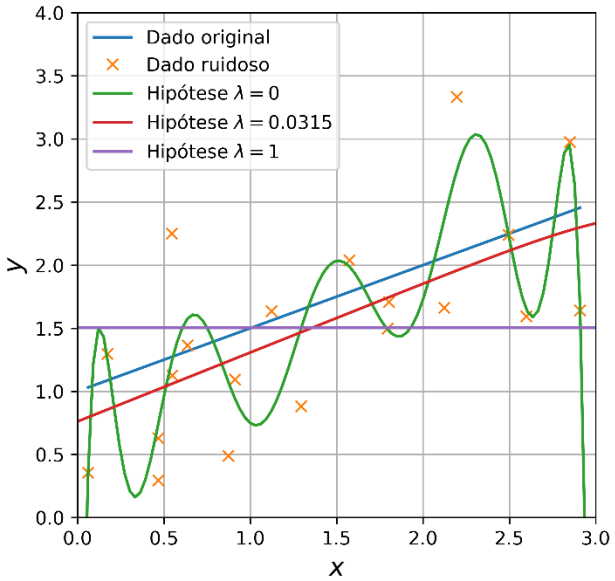
$$\min_{\boldsymbol{a}\in\mathbb{R}} \|\boldsymbol{y} - \boldsymbol{\Phi a}\|^2$$
$$\text{s.}\,a.\,\|\boldsymbol{a}\|_1 \leq c,$$

onde c restringe a magnitude dos pesos e é inversamente proporcional à λ

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

LASSO regression

- **Vantagem**: a regularização com a ***norma L1*** tem como vantagem a produção de ***modelos esparsos***.
  - Ou seja, a **vantagem** da regressão LASSO está no fato de que vários elementos do vetor de pesos acabam sendo ***anulados***, mostrando efetivamente que os pesos correspondentes são irrelevantes.
  - Isso sugere a ocorrência implícita de um processo de ***seleção de variáveis***, e leva a ***modelos*** mais ***regulares***, ou seja, ***menos complexos***.
- **Desvantagem**: como a ***norma*** não possui derivada no ponto $a_i = 0, \forall i$, o problema da minimização não possui solução em forma fechada.



Por exemplo, na figura ao lado foi utilizado um polinômio de ordem 10, onde a linha em vermelho, com $\lambda = 0.0315$, parece quase linear pois todos os pesos dos atributos de grau mais alto do polinômio são iguais a zero.

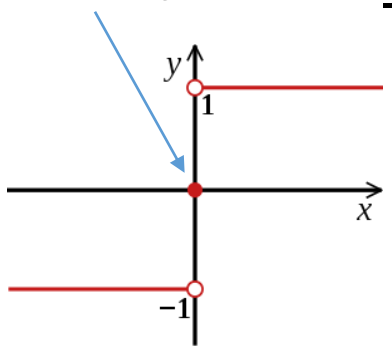
Exemplo: lasso_regressionv2.ipynb

LASSO regression com gradiente descendente

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

$$J_e(\boldsymbol{a}) = \frac{1}{N}\|\boldsymbol{y} - \boldsymbol{\Phi a}\|^2 + \lambda\|\boldsymbol{a}\|_1$$

indeterminação



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

$$\frac{\partial J_e(\boldsymbol{a})}{\partial a_k} = \frac{-2}{N}\sum_{i=0}^{N-1}[y(i) - \hat{y}(i)]\, x_k(i)\,,\ k = 0$$

← Gradiente com relação ao peso de bias

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

← Equação geral do vetor gradiente

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

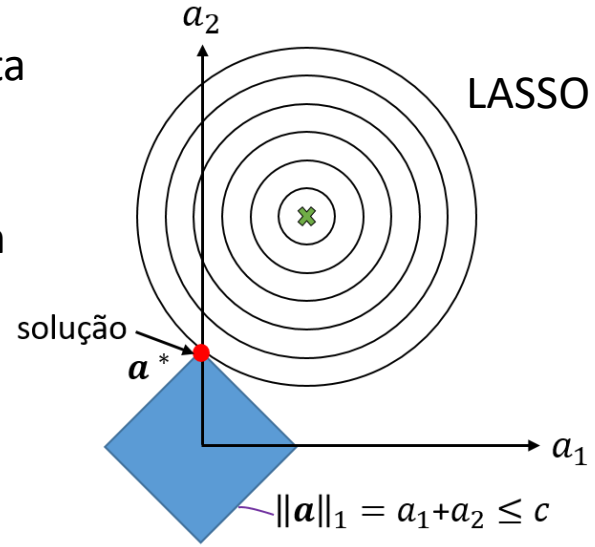
$$\boldsymbol{a} = \boldsymbol{a} + 2\alpha\left[\frac{1}{N}\boldsymbol{\Phi}^T(\boldsymbol{y} - \boldsymbol{\Phi a}) - \frac{\lambda}{2}\boldsymbol{I}'\text{sign}(\boldsymbol{a})\right], \forall k$$

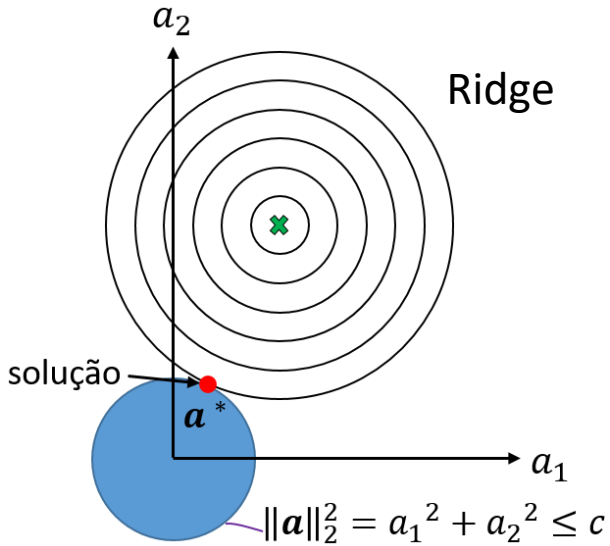
Vantagem do LASSO sobre Ridge

- A vantagem do LASSO sobre a regressão Ridge está na forma de diamante/losango da ***região de factibilidade*** criada pela penalidade da norma L1, o que leva à eliminação de alguns dos pesos (i.e., os pesos são zerados) rapidamente.
- Isso significa que a regressão LASSO realiza ***seleção automática*** de atributos, enquanto a regressão Ridge não apresenta esta característica.
- Podemos também entender a diferença entre as regressões de Ridge e LASSO ao compreender que, a penalização L2 usada na regressão de Ridge penaliza fortemente pesos grandes, mas quase não penaliza pesos pequenos (devido ao quadrado na norma L2), enquanto a penalização L1 usada na regressão LASSO penaliza apropriadamente até mesmo pesos pequenos.

Interpretação geométrica: LASSO vs. Ridge

O quadrado azul representa o conjunto de pontos $\boldsymbol{a}$ no espaço de pesos bidimensional que tenham norma L1 menor do que c. A solução deve estar em algum lugar dentro do quadrado.





O círculo azul representa o conjunto de pontos $\boldsymbol{a}$ no espaço de pesos bidimensional que tenham norma L2 menor do que c. A solução deve estar em algum lugar dentro do círculo.

- **Por que a regressão LASSO tem maior probabilidade de apresentar pesos iguais a zero**?
  - Figura mostra as curvas de nível da função de erro de um problema de regressão linear, bem como as regiões do ***espaço de hipóteses*** em que as restrições L2 (direita) e L1 (esquerda) são válidas, considerando o caso em que dois pesos estão sujeitos a regularização (a_1 e a_2).
  - A solução para ambos os métodos corresponde ao ponto, dentro da ***região de factibilidade*** (área em azul), mais próximo do mínimo.
  - É fácil ver que para uma posição arbitrária do mínimo, será comum que um canto do quadrado seja o ponto mais próximo do ponto de mínimo
  - A existência de ***cantos*** na região de ***factibilidade*** da restrição L1 (i.e., $\|\boldsymbol{a}\|_1$) aumenta as chances de alguns pesos assumirem o valor zero, só porque esses cantos são pontudos. E claro, os cantos são os pontos que possuem um valor igual a 0 em alguma das dimensões (i.e., pesos).

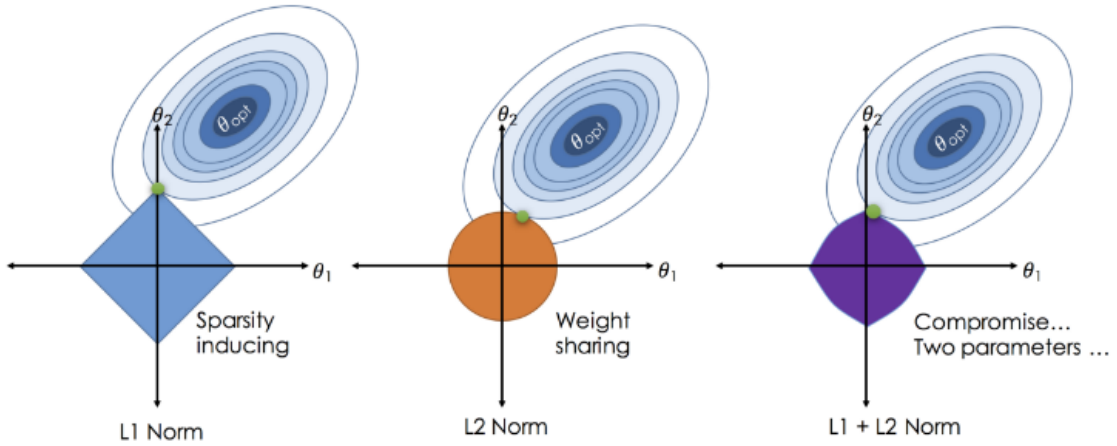
Elastic-net

Exemplo: elastic_net_regression.ipynb

- Elastic-net é uma solução intermediária entre as regressões Ridge e LASSO.
- Ela nada mais é do que uma combinação entre as penalizações baseadas nas normas L1 e L2 do vetor de pesos.

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

  onde $\kappa\ \epsilon\ [0, 1]$.
- Quando κ = 0, a Elastic-net é equivalente a Ridge regression, e quando κ = 1, ela é equivalente a Regressão Lasso.
- A seleção dos hiperparâmetros κ e λ pode ser feita por meio de **validação cruzada**. Isso também se aplica ao dois outros métodos anteriores.

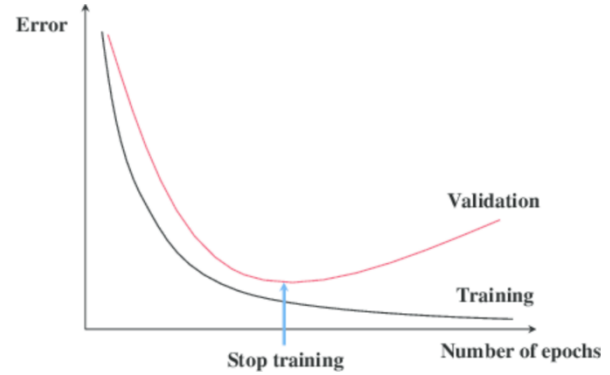


O hiperparâmetro κ dita a relação de compromisso entre as duas regularizações.

Quando utilizar cada tipo de regressão?

- **Regressão de Ridge**: 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 erraticamente se o número de atributos for maior que o número de exemplos de treinamento ou quando houverem atributos fortemente correlacionados. Nestes casos, deve-se usar a regressão Elastic-Net.
- **Elastic-Net**: é mais versátil que as anteriores, pois o parâmetro de elasticidade κ é ajustável. Uma proporção de 50% entre as penalizações L1 e L2 é uma boa escolha para esse parâmetro.

Early stopping



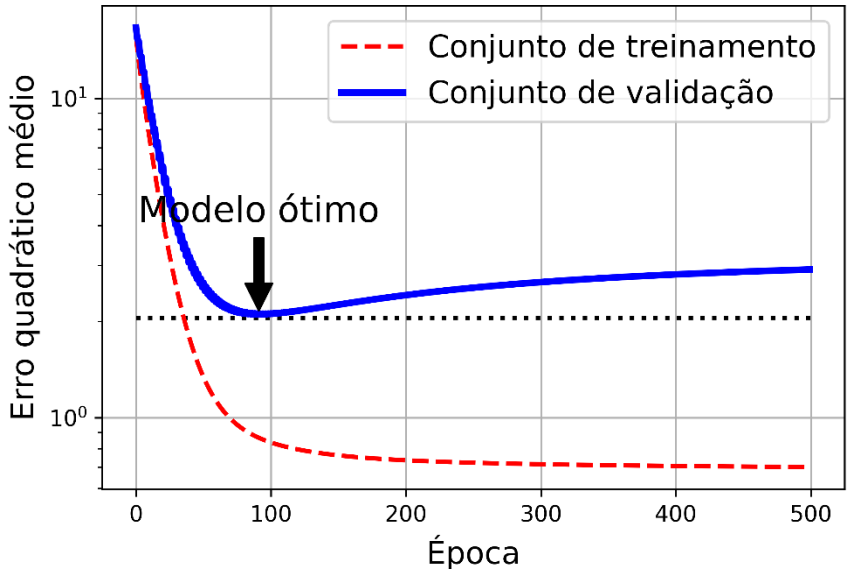
- Uma maneira de se ***regularizar*** algoritmos de aprendizado iterativo, como o ***gradiente descendente***, é interromper seu treinamento assim que o erro de validação começar a crescer de forma sistemática.
- Essa abordagem é chamada de ***early stopping*** ou ***parada antecipada***.
- Pode ser vista como uma ***regularização*** no ***tempo***.
- Assim como as outras abordagens, ela tem o objetivo de evitar o ***sobreajuste*** de um modelo.
- Intuitivamente, o algoritmo do ***gradiente descendente*** tenderá a aprender modelos cada vez mais complexos à medida que o número de épocas aumenta.
- Ao regularizar no ***tempo***, a complexidade do modelo pode ser controlada, melhorando sua ***generalização***.

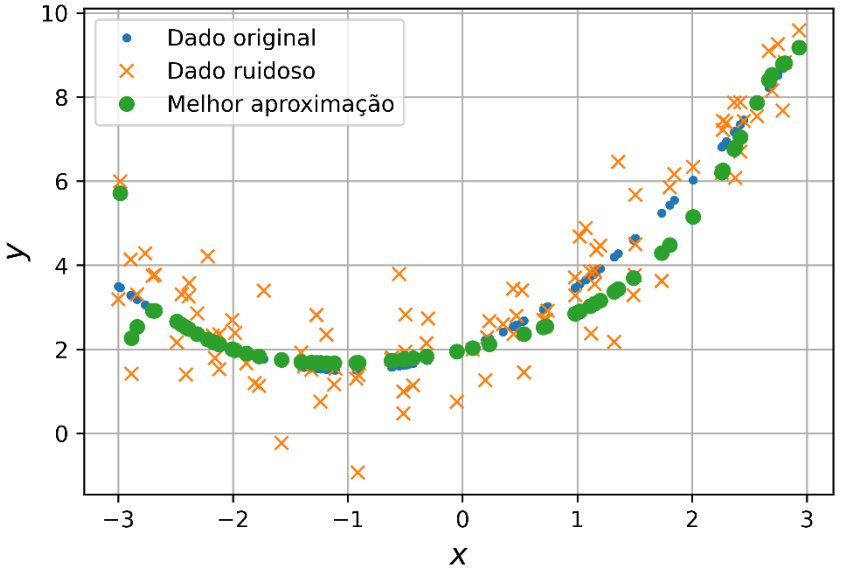
Early stopping

- Duas estratégias para se definir o critério de parada são:
  - Interromper o treinamento quando o valor do erro de validação aumenta por ***P*** épocas/iterações sucessivas, sendo o parâmetro ***P*** chamado de ***paciência***.
    - **Problema**: nem sempre o erro de validação apresenta um comportamento bem definido. Como a curva do erro de validação pode oscilar bastante e apresentar um comportamento pouco previsível, nem sempre é pertinente desenvolver detectores automáticos de mínimos e encerrar o treinamento ali.
  - Outra estratégia é permitir que o treinamento do modelo prossiga, mas sempre armazenando os pesos associados ao ***mínimo erro de validação***, os quais serão considerados como a solução para o modelo ao final do treinamento.

Exemplo: Early stopping

- A figura mostra um modelo de regressão polinomial de alto grau sendo treinado usando o gradiente descendente estocástico.
- À medida que as épocas passam, o algoritmo aprende e seu erro de treinamento diminui, juntamente com o erro de validação.
- No entanto, após algumas 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, para-se de treinar o modelo assim que o erro de validação atinge o mínimo, garantindo um modelo que tenha uma boa ***generalização***.

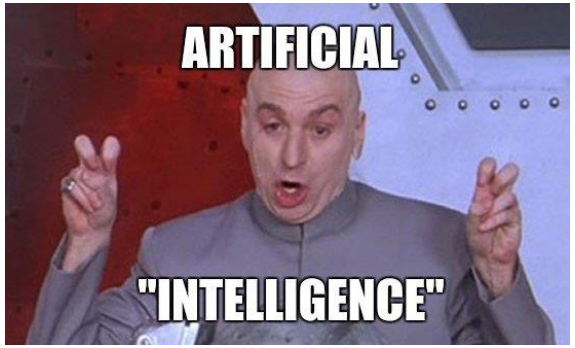


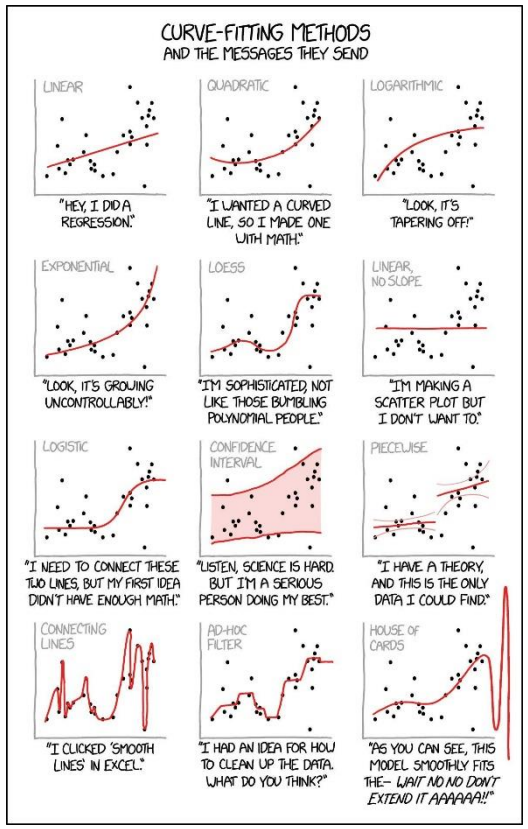


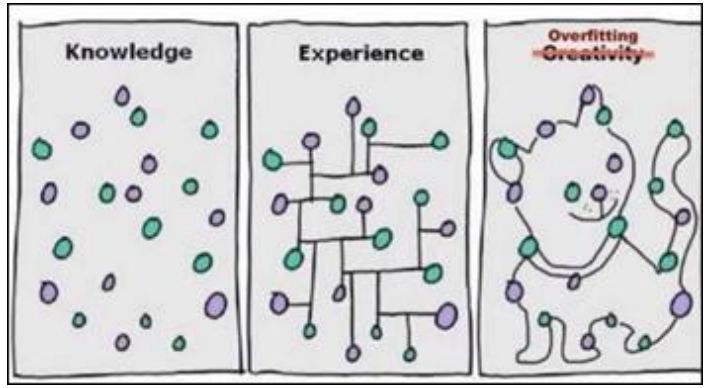
Exemplo: early_stoppingv3.ipynb

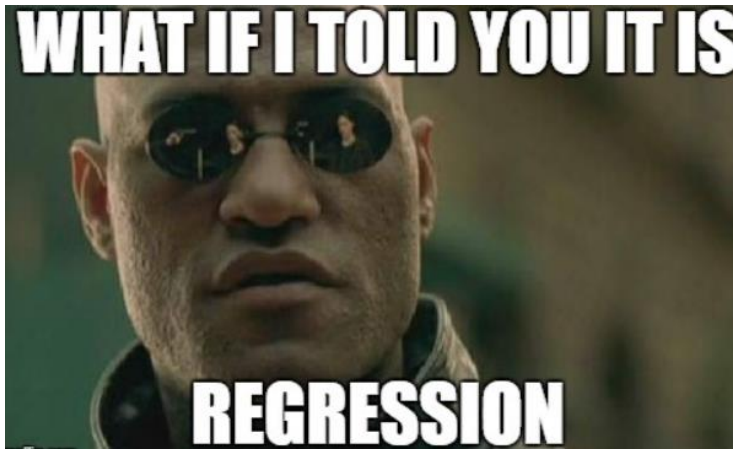
Obrigado!


THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS


ARTIFICIAL
"INTELLIGENCE"


CURVE-FITTING METHODS
AND THE MESSAGES THEY SEND
LINEAR
"HEY, I DID A REGRESSION."
QUADRATIC
"I WANTED A CURVED LINE, SO I MADE ONE WITH MATH."
LOGARITHMIC
"LOOK, IT'S TAPERING OFF!"
EXPONENTIAL
"LOOK, IT'S GROWING UNCONTROLLABLY!"
LOESS
"I'M SOPHISTICATED, NOT LIKE THOSE BUMBLING POLYNOMIAL PEOPLE."
LINEAR, NO SLOPE
"I'M MAKING A SCATTER PLOT BUT I DON'T WANT TO."
LOGISTIC
"I NEED TO CONNECT THESE TWO LINES, BUT MY FIRST IDEA DIDN'T HAVE ENOUGH MATH."
CONFIDENCE INTERVAL
"LISTEN, SCIENCE IS HARD. BUT I'M A SERIOUS PERSON DOING MY BEST."
PIECEWISE
"I HAVE A THEORY, AND THIS IS THE ONLY DATA I COULD FIND."
CONNECTING LINES
"I CLICKED 'SMOOTH LINES' IN EXCEL."
AD-HOC FILTER
"I HAD AN IDEA FOR HOW TO CLEAN UP THE DATA. WHAT DO YOU THINK?"
HOUSE OF CARDS
"AS YOU CAN SEE, THIS MODEL SMOOTHLY FITS THE— WAIT NO NO DON'T EXTEND IT AAAAAA!!"

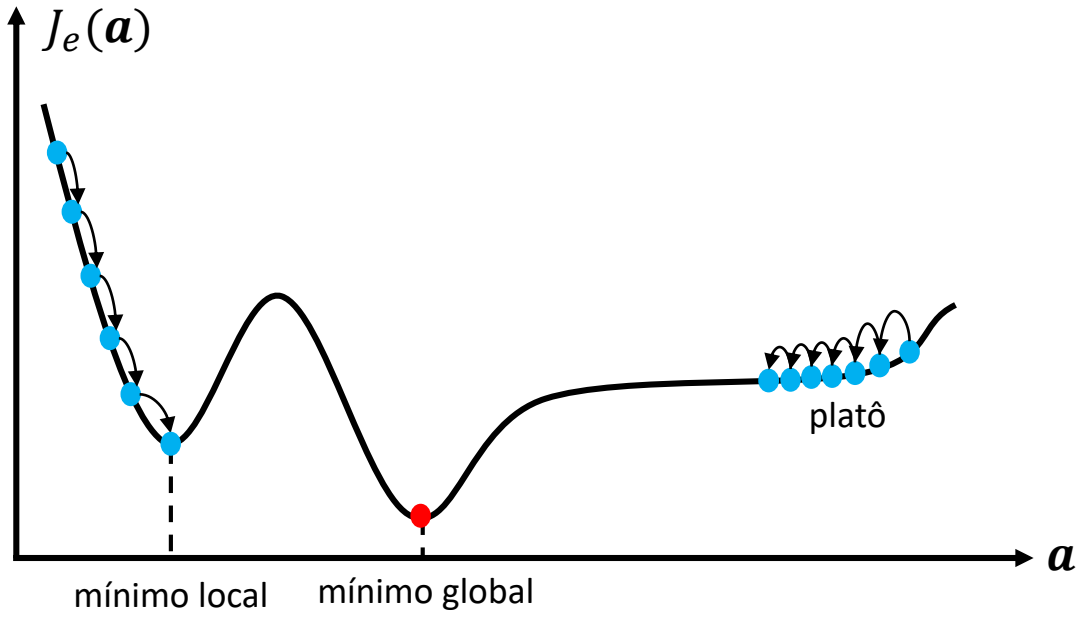

Knowledge
Experience
Overfitting
Creativity

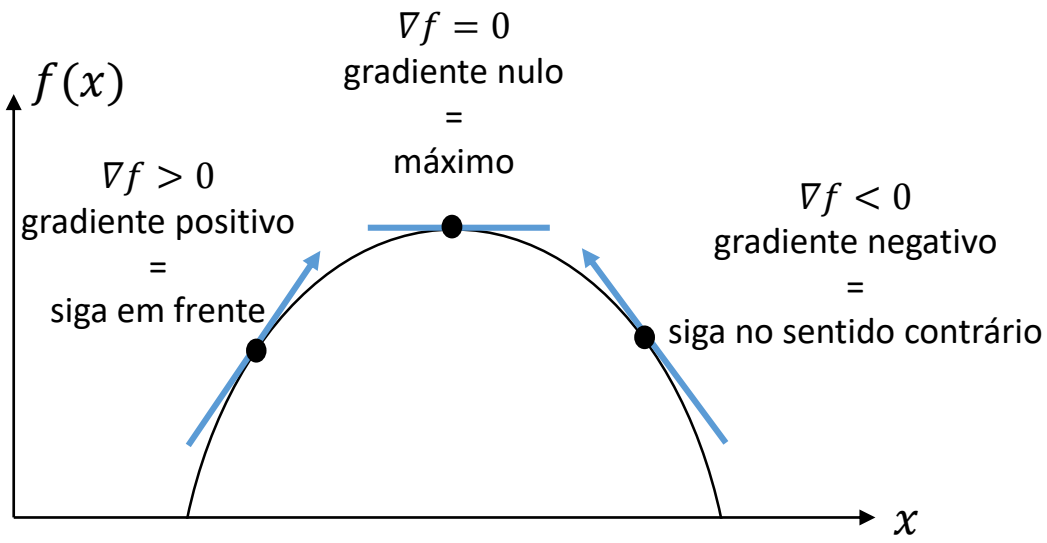

WHAT IF I TOLD YOU IT IS
REGRESSION

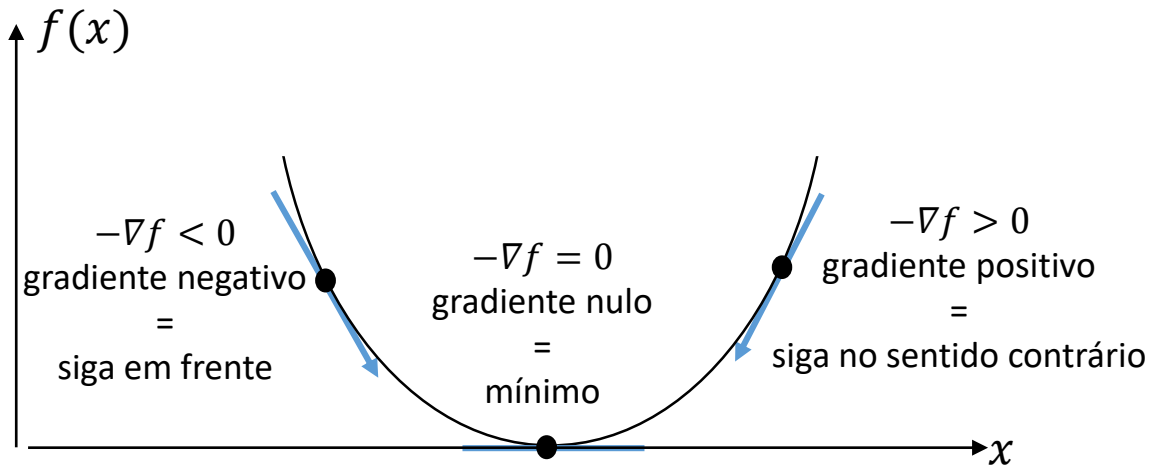

WHOA
I KNOW LINEAR REGRESSION

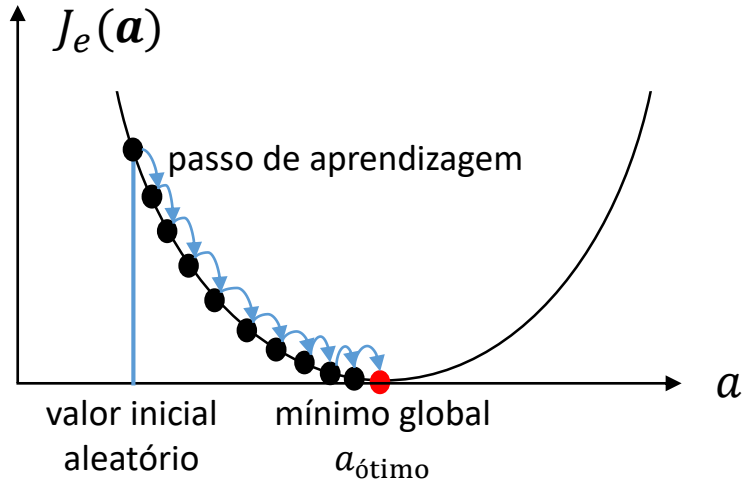

LETS LEARN
REGRESSION
memegenerator.net

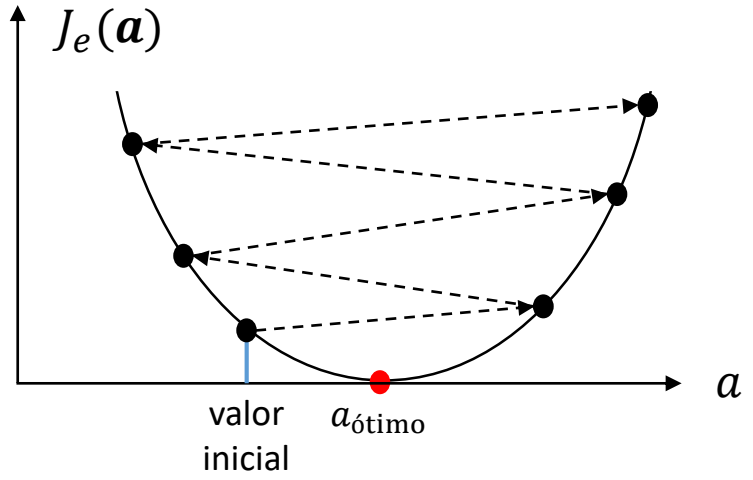
FIGURAS

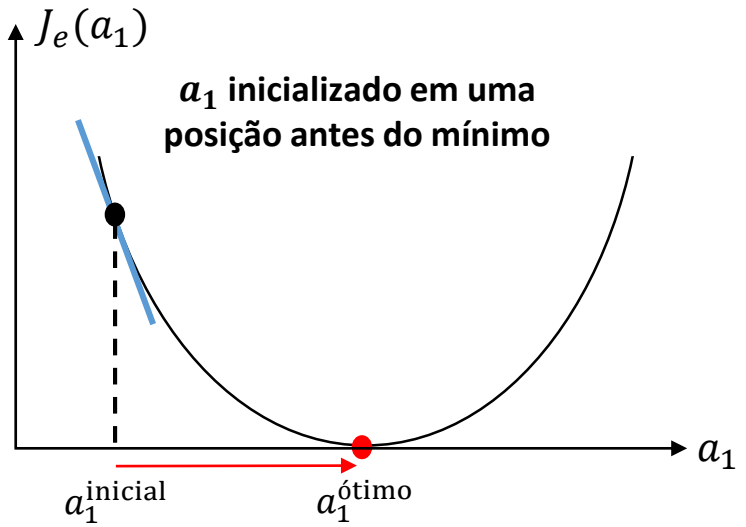

$J_e(\boldsymbol{a})$
platô
$\boldsymbol{a}$
mínimo local
mínimo global


$f(x)$
$\nabla f = 0$
gradiente nulo
=
máximo
$\nabla f > 0$
gradiente positivo
=
siga em frente
$\nabla f < 0$
gradiente negativo
=
siga no sentido contrário
x


$f(x)$
$-\nabla f < 0$
gradiente negativo
=
siga em frente
$-\nabla f = 0$
gradiente nulo
=
mínimo
$-\nabla f > 0$
gradiente positivo
=
siga no sentido contrário
x

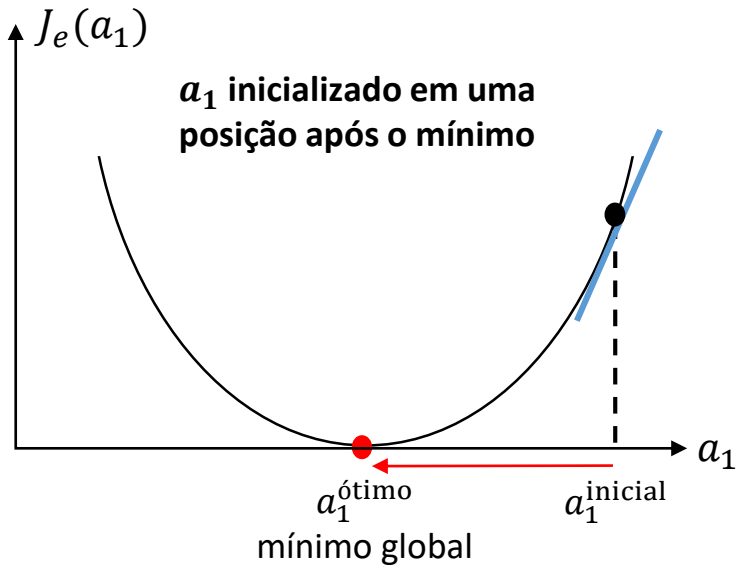

$J_e(\boldsymbol{a})$
passo de aprendizagem
a
valor inicial
aleatório
mínimo global
$a_{\text{ótimo}}$


$J_e(\boldsymbol{a})$
a
valor
inicial
$a_{\text{ótimo}}$



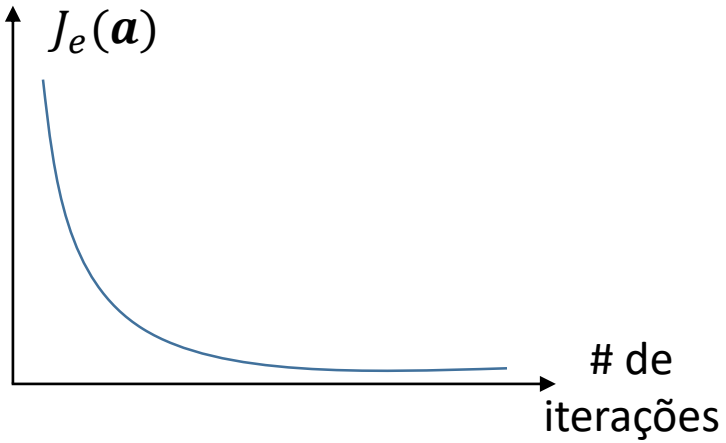
gradiente negativo: $a_1 = a_1^{\text{inicial}} + \alpha \nabla J_e(a_1)$

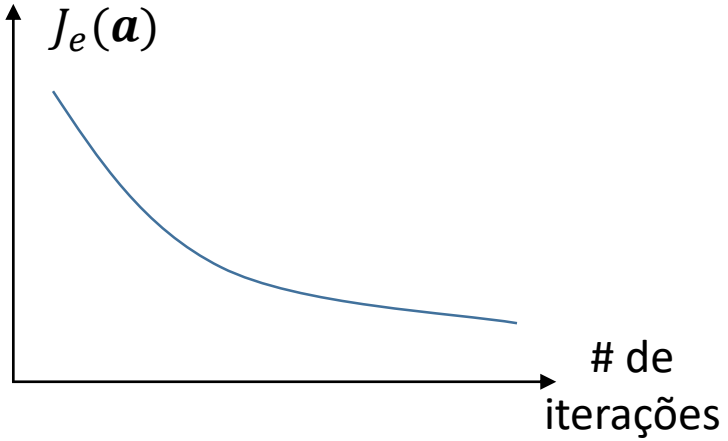
a_1 aumenta e se aproxima do mínimo

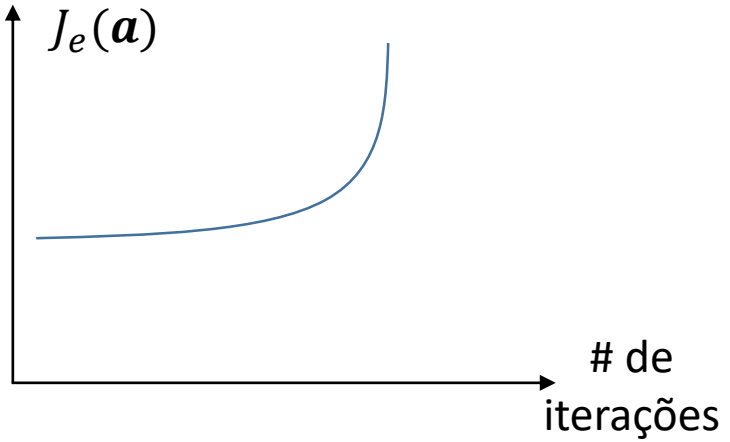


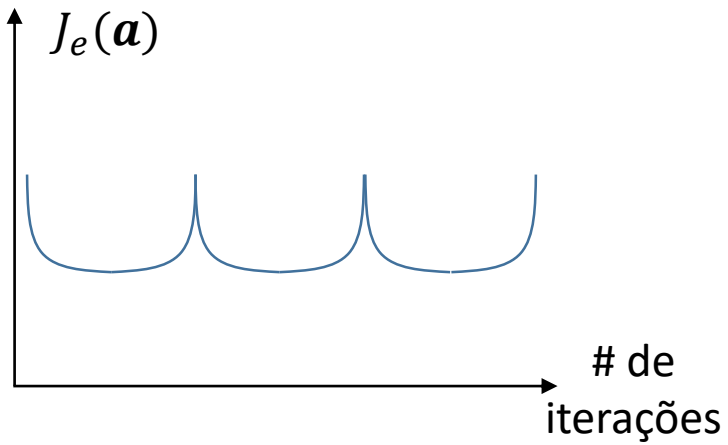
gradiente positivo: $a_1 = a_1^{\text{inicial}} - \alpha \nabla J_e(a_1)$

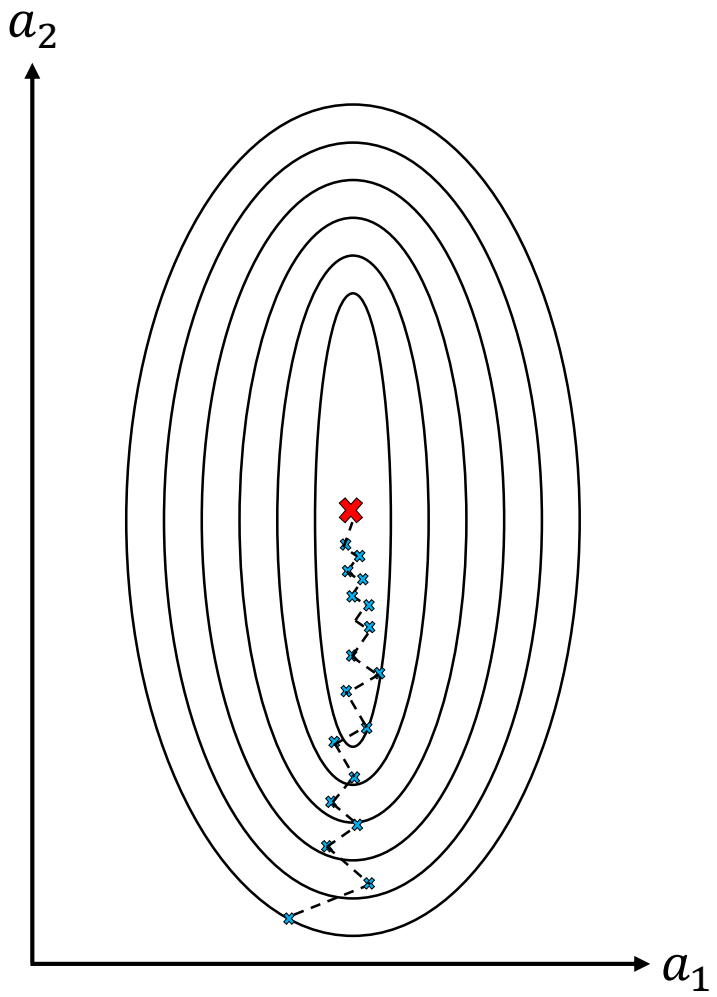
a_1 diminiu e se aproxima do mínimo

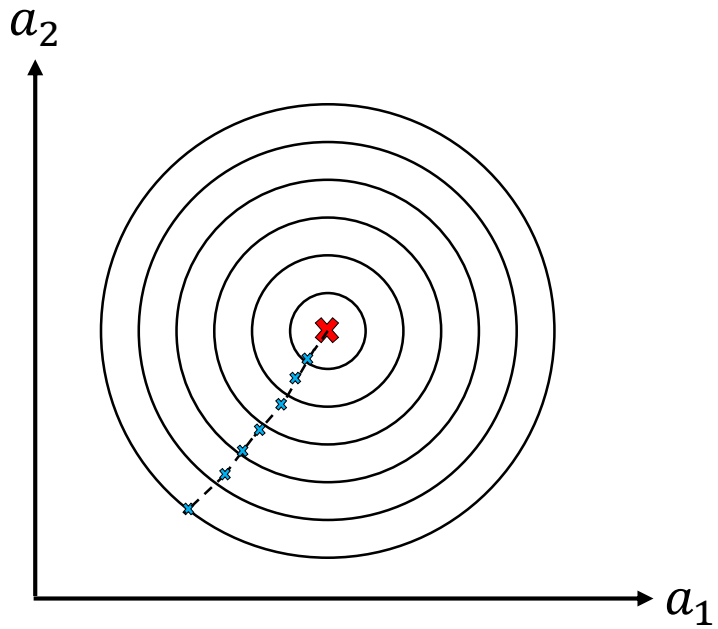

$J_e(\boldsymbol{a})$
de
iterações

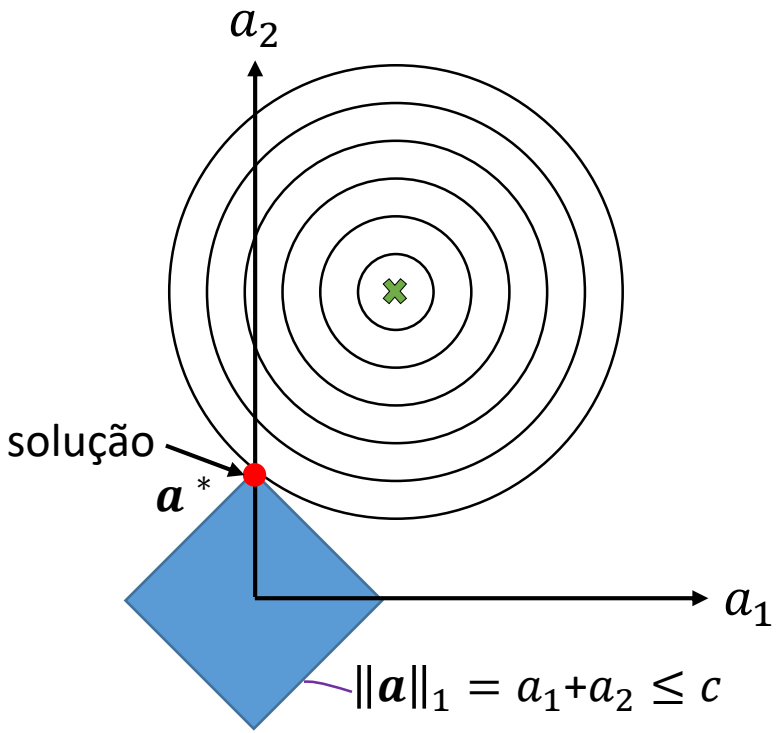

$J_e(\boldsymbol{a})$
de
iterações

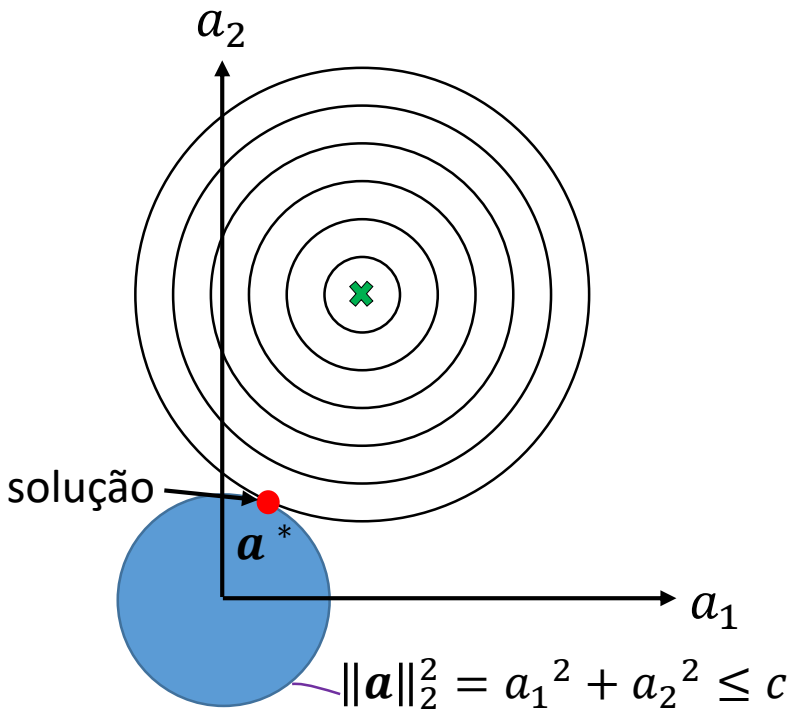

$J_e(\boldsymbol{a})$
de
iterações

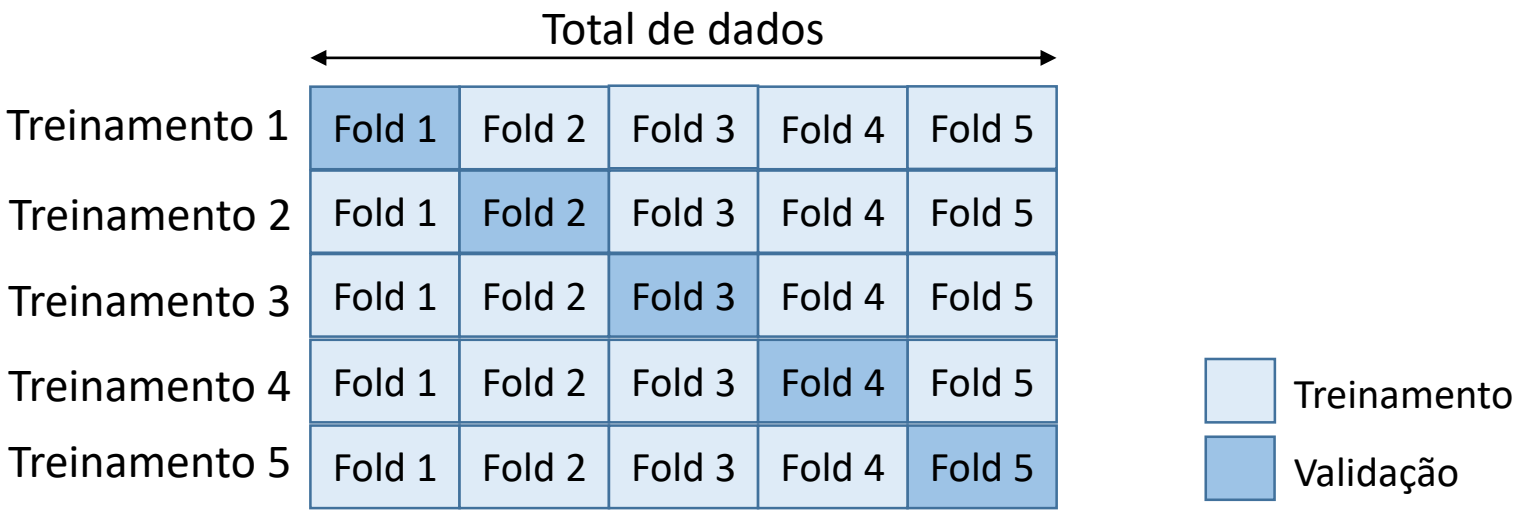

$J_e(\boldsymbol{a})$
de
iterações


a_2
a_1


a_2
a_1


a_2
solução
$\boldsymbol{a}^*$
a_1
$\|\boldsymbol{a}\|_1 = a_1 + a_2 \leq c$


a_2
solução
$\boldsymbol{a}^*$
a_1
$\|\boldsymbol{a}\|_2^2 = {a_1}^2 + {a_2}^2 \leq c$


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

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