

TP555 - Inteligência Artificial e Machine Learning: *Regressão para Modelos Não- Lineares*

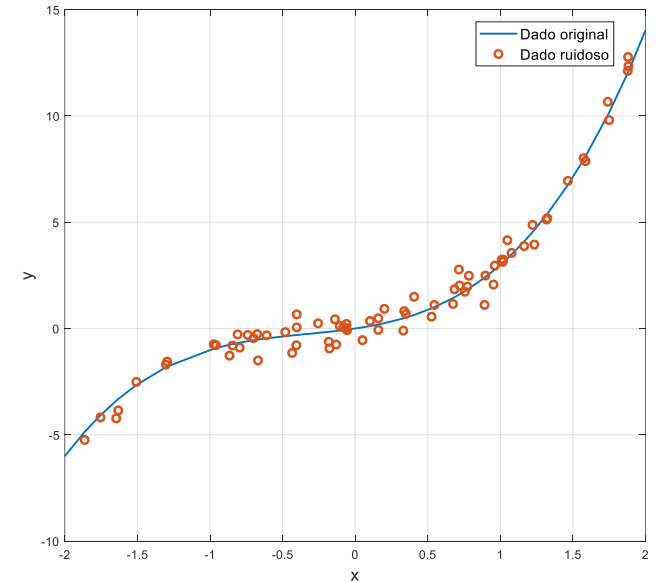
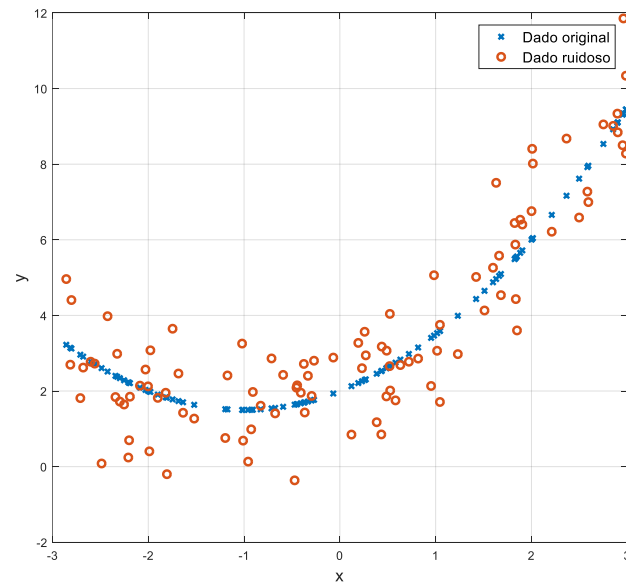
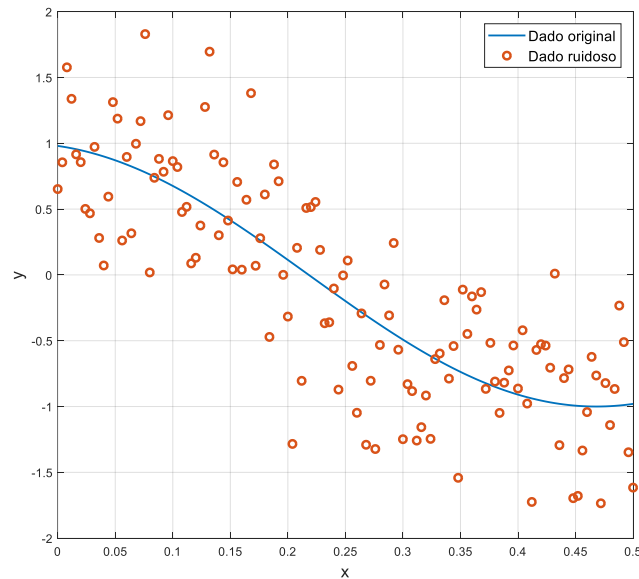


Inatel

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

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

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**, $\Phi(i)$, como o **vetor de funções-base**

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

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)
```

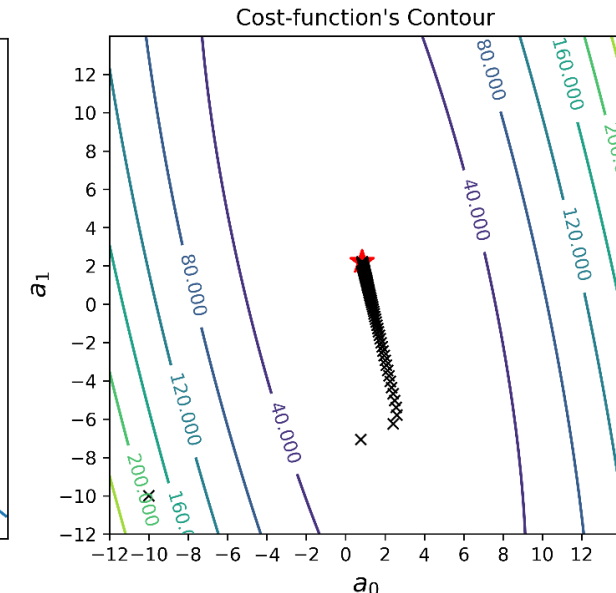
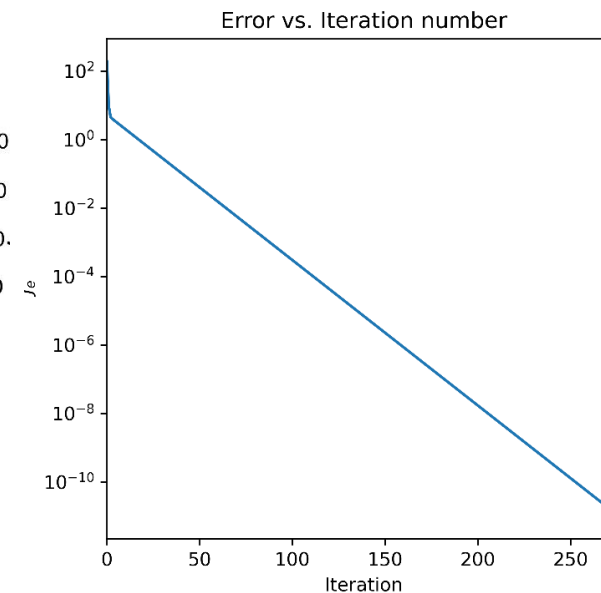
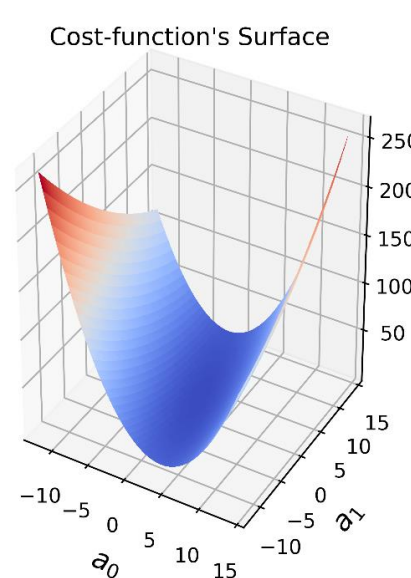
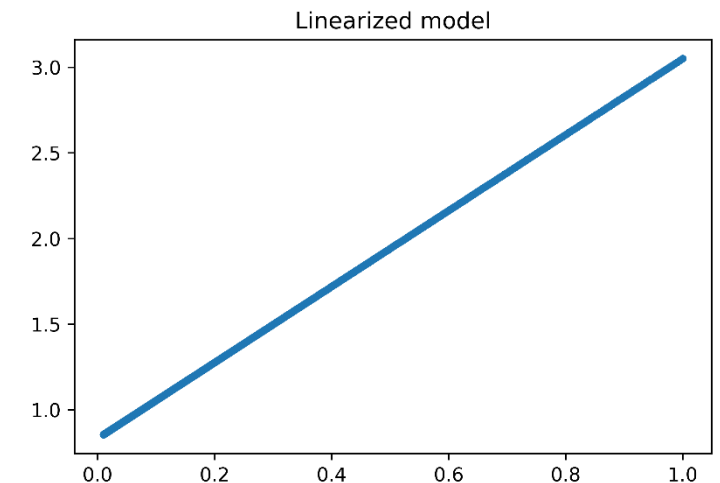
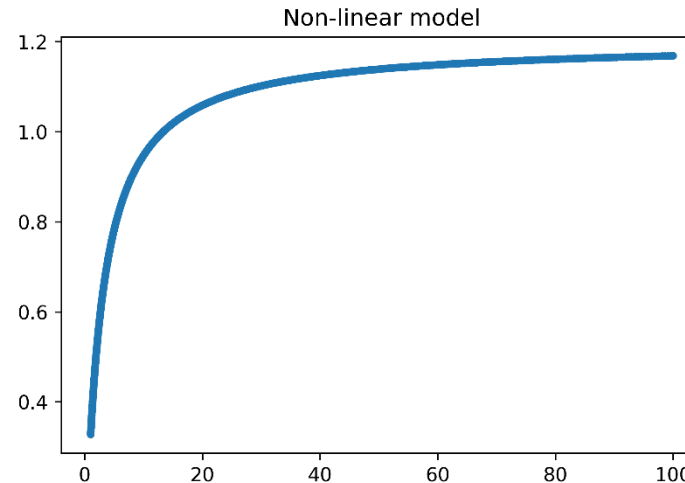
```
# Non-linear model.
```

$$y = (a_0 * x_1) / (a_1 + x_1)$$

```
# Linearized model.
```

$$y_{lin} = 1/y = (1/a_0) + (a_1/a_0) * (1/x_1)$$
$$= a_0' + a_1' * x_1'$$

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 combinações de **potências 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, 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.

Regressão Polinomial: Exemplo

- Geramos 30 exemplos do seguinte mapeamento verdadeiro:

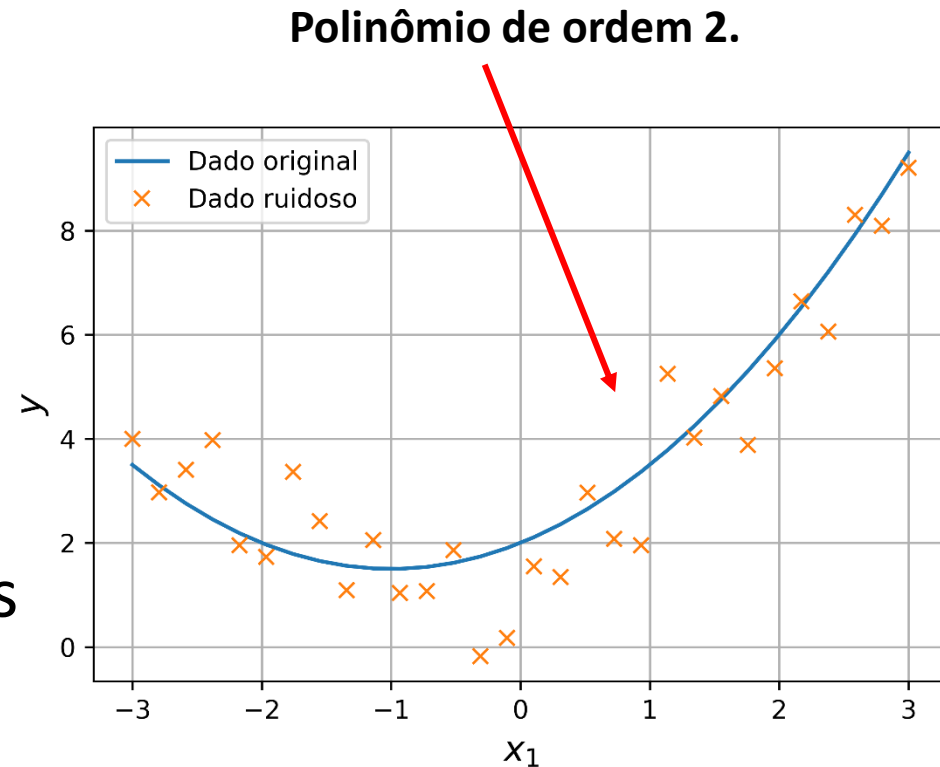
$$y(x_1(n)) = 2 + x_1(n) + 0.5x_1^2(n),$$

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, se não soubéssemos a ordem por trás do **modelo gerador, 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)

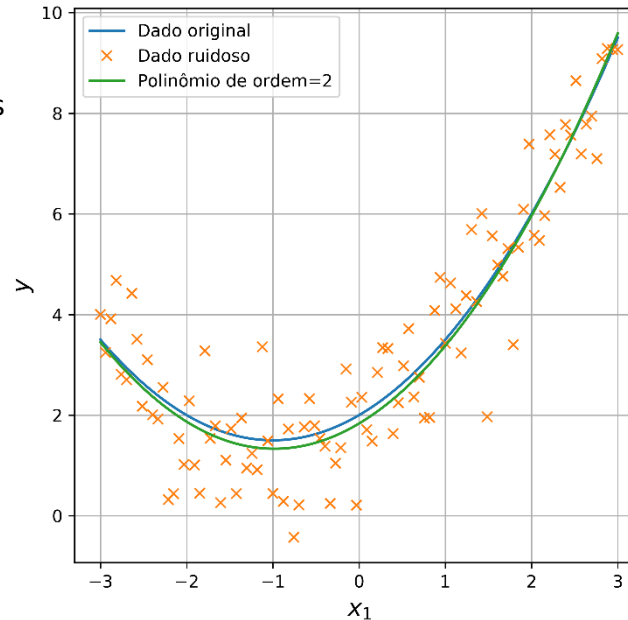
# Instantiate a degree 2 polynomial.
poly_features = PolynomialFeatures(degree=2, include_bias=True)
# Instantiate a scaler.
std_scaler = StandardScaler()
# Instantiate a linear regressor.
lin_reg = LinearRegression()

# 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.

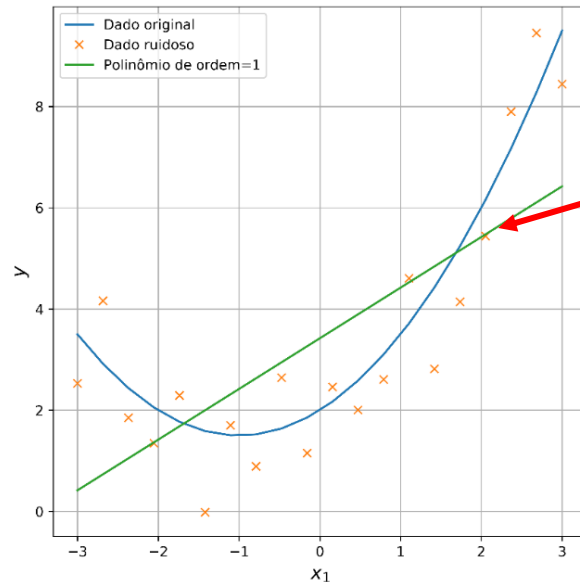
Exemplos:

- `using_poly_features_from_scikit.ipynb`
- `polynomial_regression_mse_calc.ipynb`



Regressão Polinomial: Qual ordem usar?

Polinômio de ordem 1.



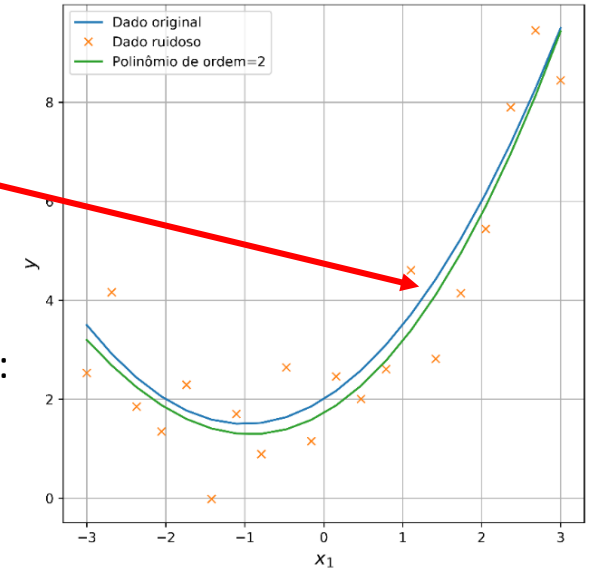
Reta não é flexível o suficiente para se contorcer e aproximar os dados.

Flexibilidade e grau de generalização muito baixos.

Ordem ótima pois é a mesma do modelo gerador.

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

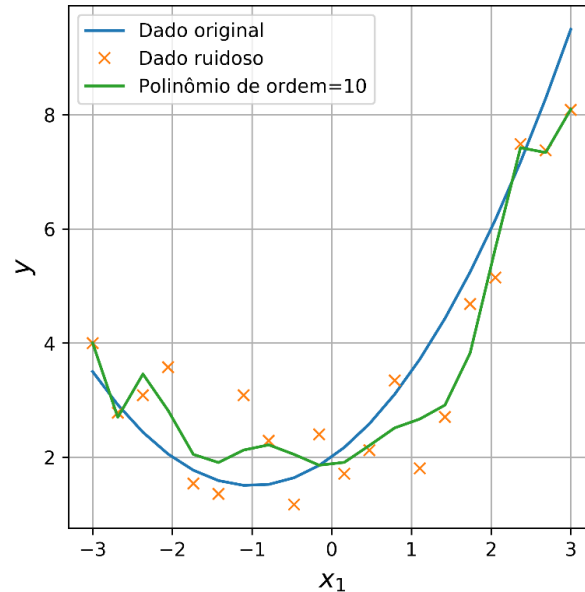
Polinômio de ordem 2.



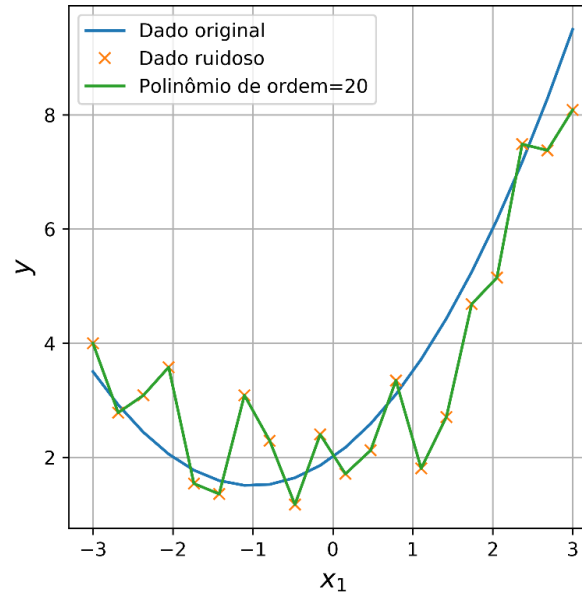
- Polinômio de ordem 1 (i.e., uma reta) não tem flexibilidade o suficiente para aproximar bem os dados.
- O modelo erra muito tanto para predição dos exemplos de treinamento quanto para exemplos de validação (ou seja, exemplos não vistos durante o treinamento).
- Efeito conhecido como **subajuste** ou **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?

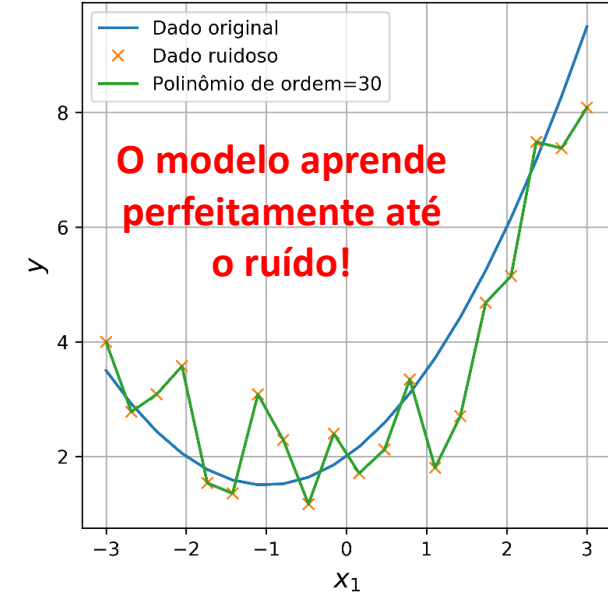
Polinômio de ordem 10.



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

- 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., $y = f(x)$, 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 $y = f(x)$, pois em muitos casos $f(x)$, 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 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, $\hat{y} = h(x)$, 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*** a um determinado problema de regressão.
- Uma estratégia simples 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.

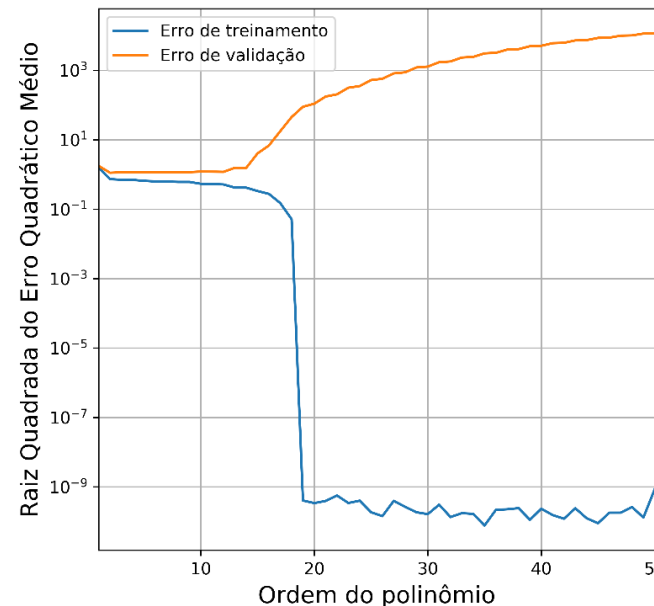
A maldição da complexidade

- 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 **erro de validação** próximo ao do **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;
```

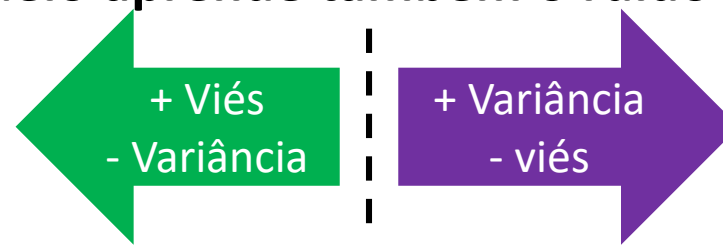
```
y_noisy = y + np.random.randn(N, 1)
```

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** estão 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 exemplos de treinamento quanto para exemplos não vistos durante o treinamento (validação).
- Esse fenômeno é chamado de ***subajuste (underfitting)*** ou ***alto-viés***.
 - Capacidade é muito baixa, pois o modelo não tem flexibilidade o suficiente.
- Por outro lado, o modelo com polinômio de grau 30 acerta todos os exemplos de treinamento mas erraria muito para exemplos não vistos durante treinamento.
- Esse fenômeno é chamado de ***sobreajuste (overfitting)*** ou ***alta-variância***.
 - Capacidade é tão alta que o modelo aprende também o ruído presente no conjunto de treinamento.



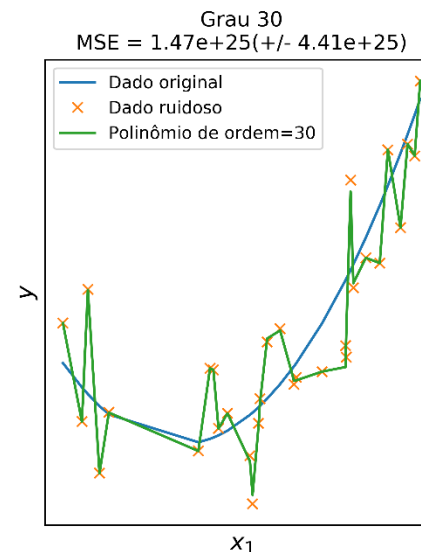
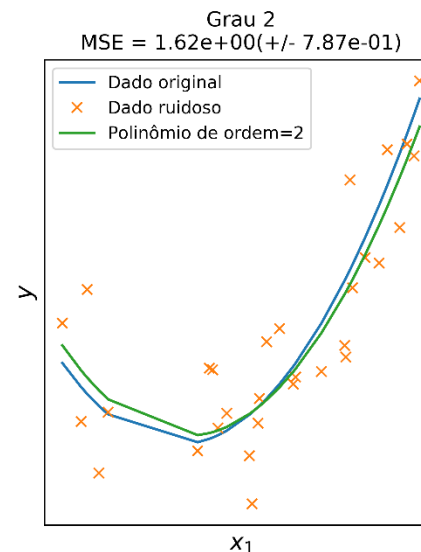
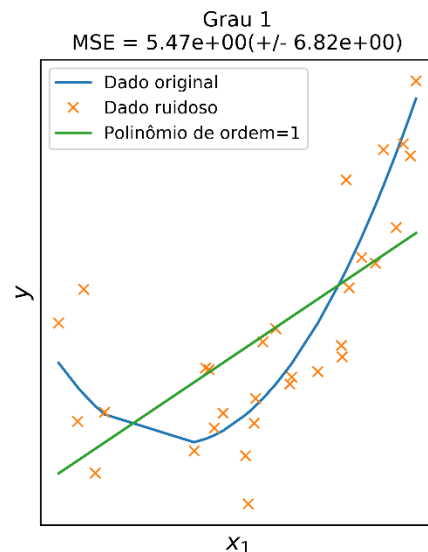
Sobreajuste e subajuste

- A figura abaixo apresenta exemplos de subajuste, sobreajuste e ajuste ótimo.
- O modelo com polinômio de grau 2 se ajusta bem aos exemplos, mas sem acertá-los perfeitamente.
- Este é provavelmente o modelo ótimo em termos da *relação de 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**



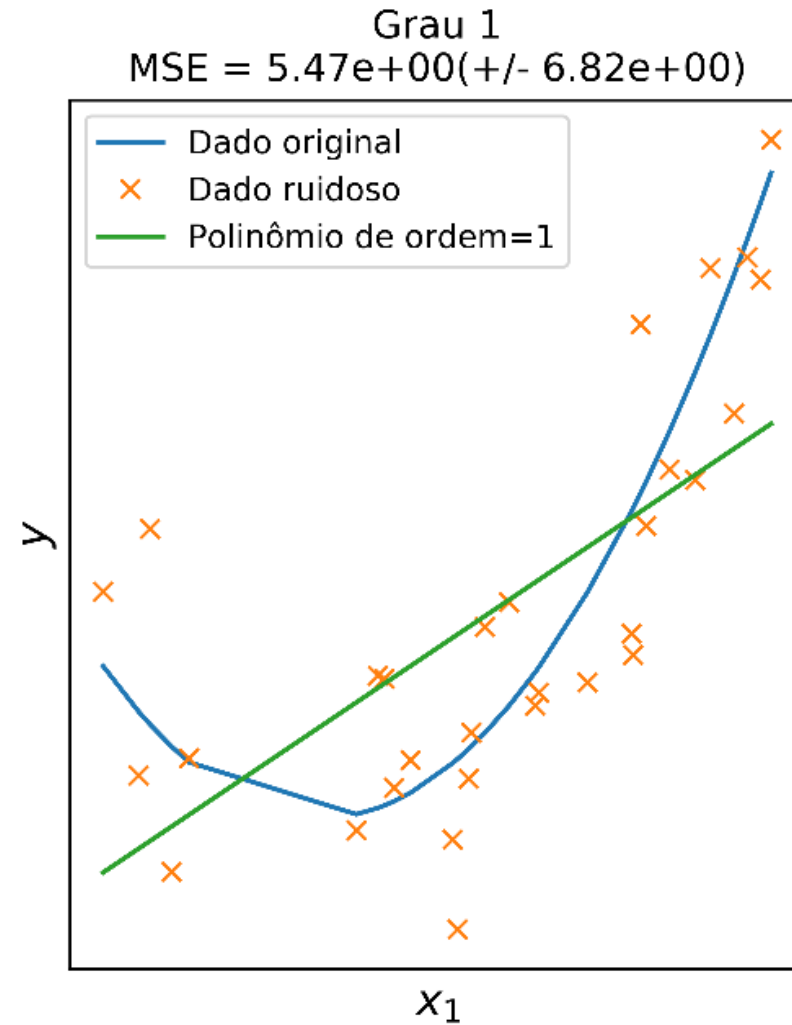
Sobreajuste



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

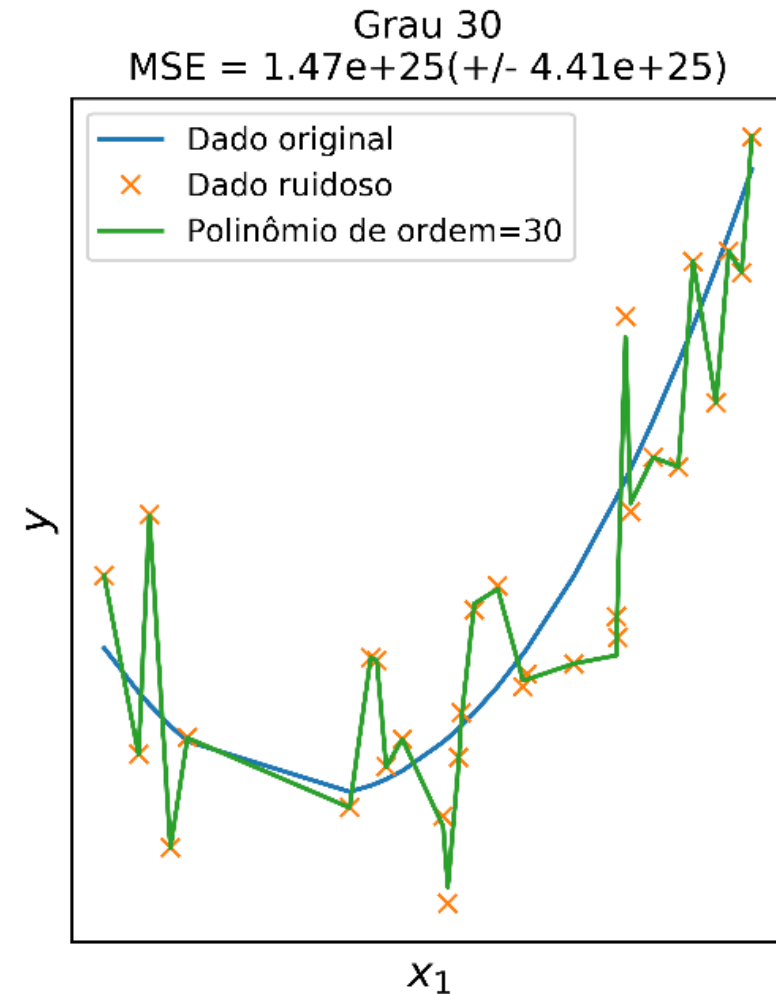
Subajuste (Resumo)

- **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 situação 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 “*memoriza*” 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 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, ou seja, sobre o formato do modelo gerador.
 - Também conhecido como ***erro de representação*** ou ***erro de bias*** (viés).
 - Depende do nível de complexidade do modelo usado para a aproximação.
 - Mesmo que o conjunto de treinamento contivesse infinitos exemplos do mapeamento verdadeiro, $y = f(x)$, devido a suposição errada sobre a complexidade dos dados, não seria possível obter-se um modelo que se aproximasse do ***mapeamento verdadeiro***, $y = f(x)$.
 - 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 tanto treinamento quanto de validação.
 - **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 ***erro de 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

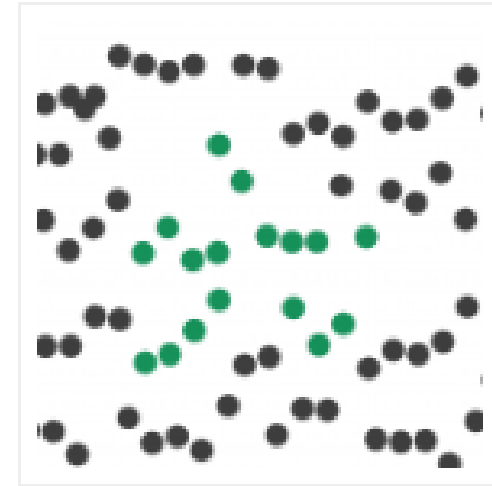
- **Erro irreduzível:** é o erro devido ao ruído contido nos dados.
 - Também conhecido como ***erro de Bayes***.
 - É o erro que não pode ser reduzido através da criação de bons modelos.
 - Por melhor que seja nosso modelo, os dados terão uma certa quantidade de ruído que não pode ser removido.
- Exemplos: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

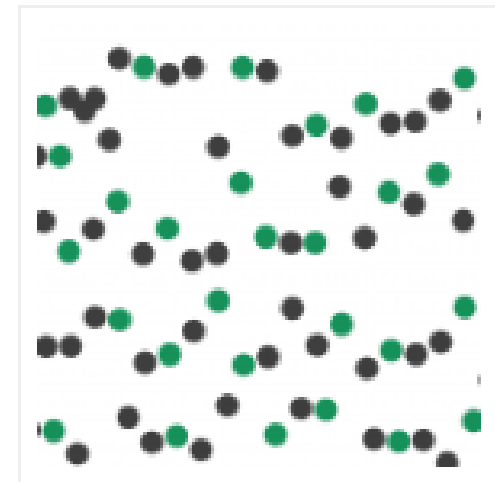
- Após todas essas informações, algumas perguntas surgem a respeito do modelo e seu treinamento.
- Como escolhemos a ***função hipótese*** quando não conhecemos o ***mapeamento verdadeiro***?
 - Ou seja, o **quão complexo** deve ser o modelo?
- Como podemos dizer que o modelo está ajustando demais ou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

Validação cruzada

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



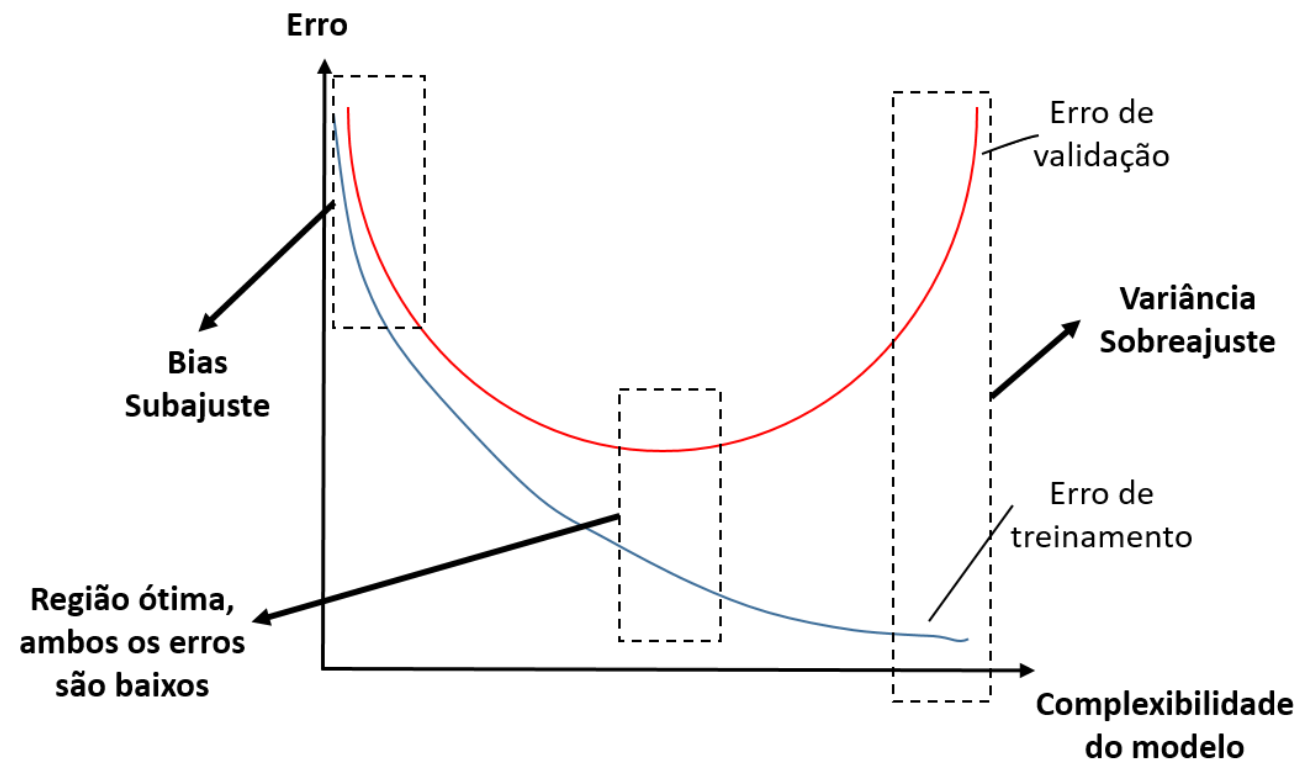
Conjuntos
não-representativos.



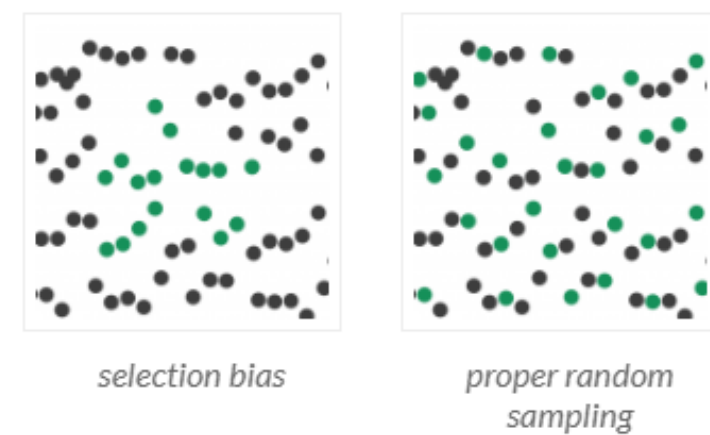
Conjuntos
representativos.

Comportamento “*típico*” dos erros

- A figura mostra o comportamento “*típico*” dos erros de treinamento e validação durante a validação cruzada.
- **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.
- **Região ótima:** ambos os erros são baixos.

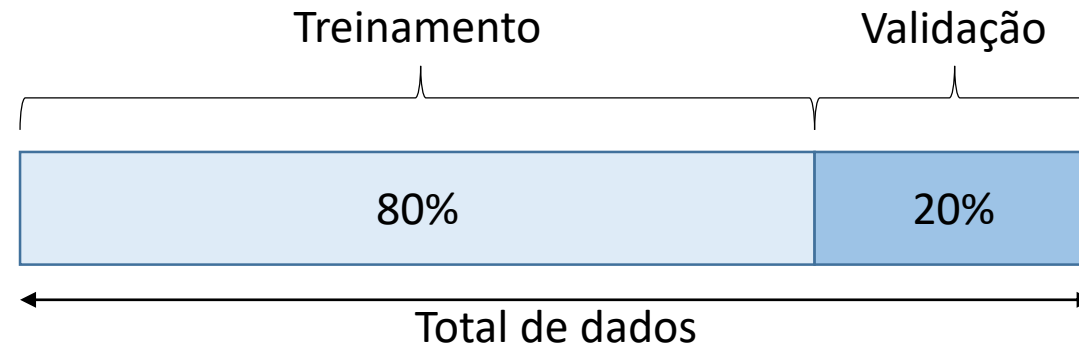


Estratégias para validação cruzada



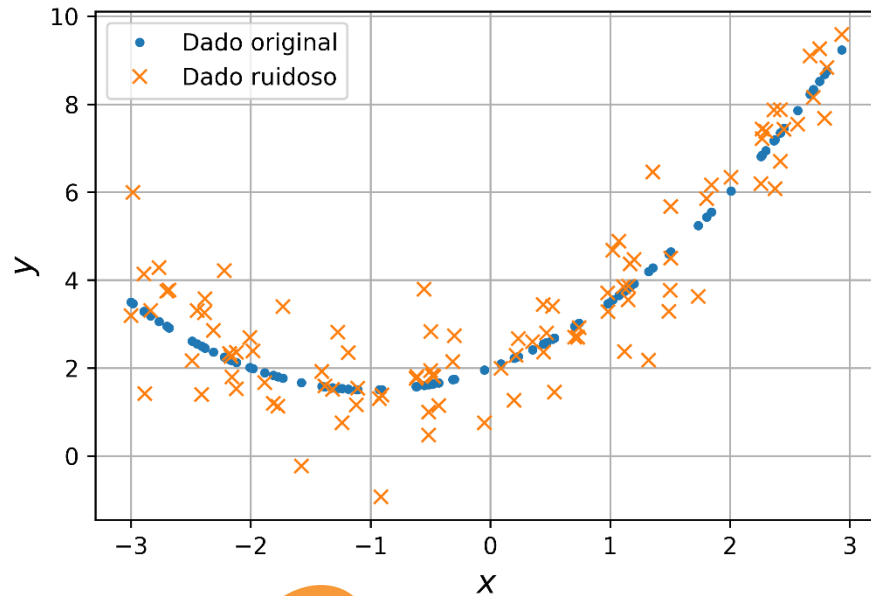
- Estratégias de **validação cruzada** são usadas para validação de modelos.
- Elas fornecem uma visão sobre como o modelo generaliza para um **conjunto de dados independente**.
- Um ponto importante é termos conjuntos **representativos** de modo que não ocorra um **viés de seleção**.
 - O **viés de 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 as mais utilizadas: **holdout**, **k-fold** e **leave-p-out**.
- Existem outras estratégias, mas elas 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 tem-se apenas **um único par treinamento/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 testes.
- **Desvantagem**
 - Sofre com o problema do **viés de seleção** o que 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 testes.
 - Os resultados podem depender da divisão aleatória feita para os conjuntos de treinamento e validação.

Holdout: Exemplo



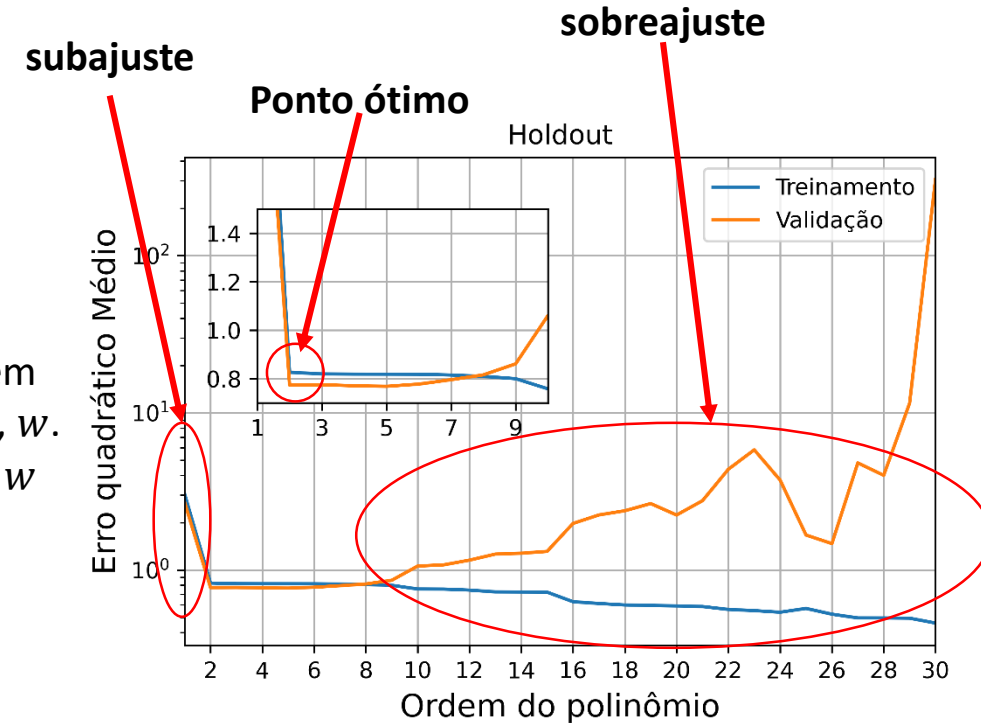
Função observável é um polinômio de segunda ordem com ruído Gaussiano branco, w .
 $y_{noisy} = 2 + x + 0.5x^2 + w$



Split the whole set into random training and validation set.

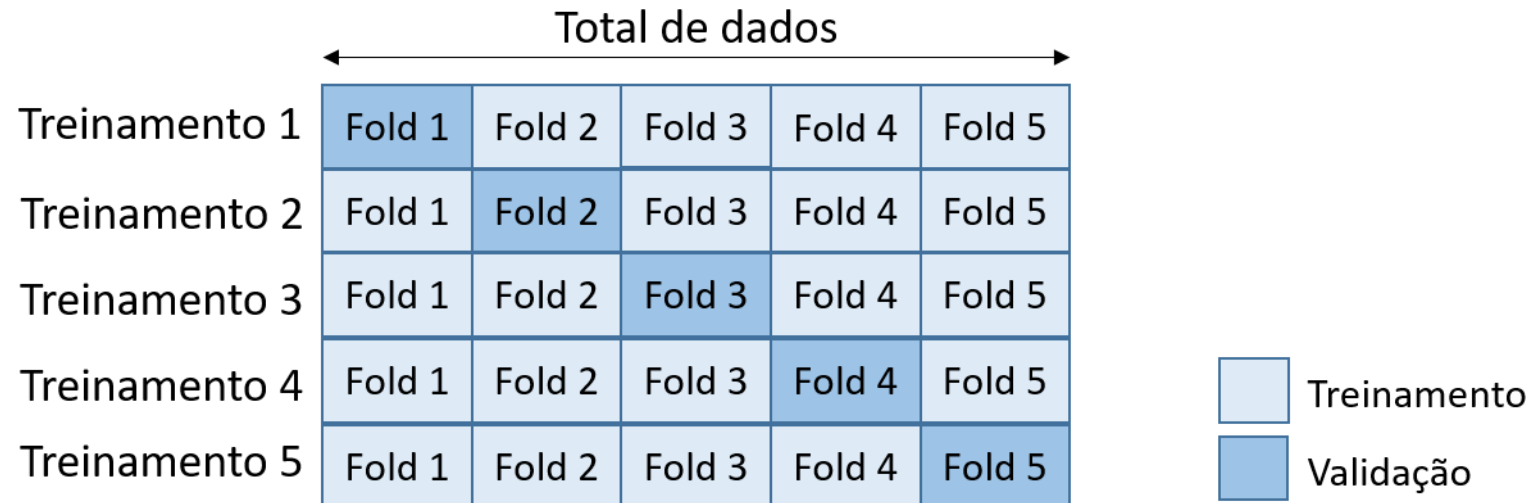
```
x_train, x_val, y_train, y_val = train_test_split(x, y_noisy, test_size=0.3, random_state=10)
```

- 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 (subconjuntos) 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.



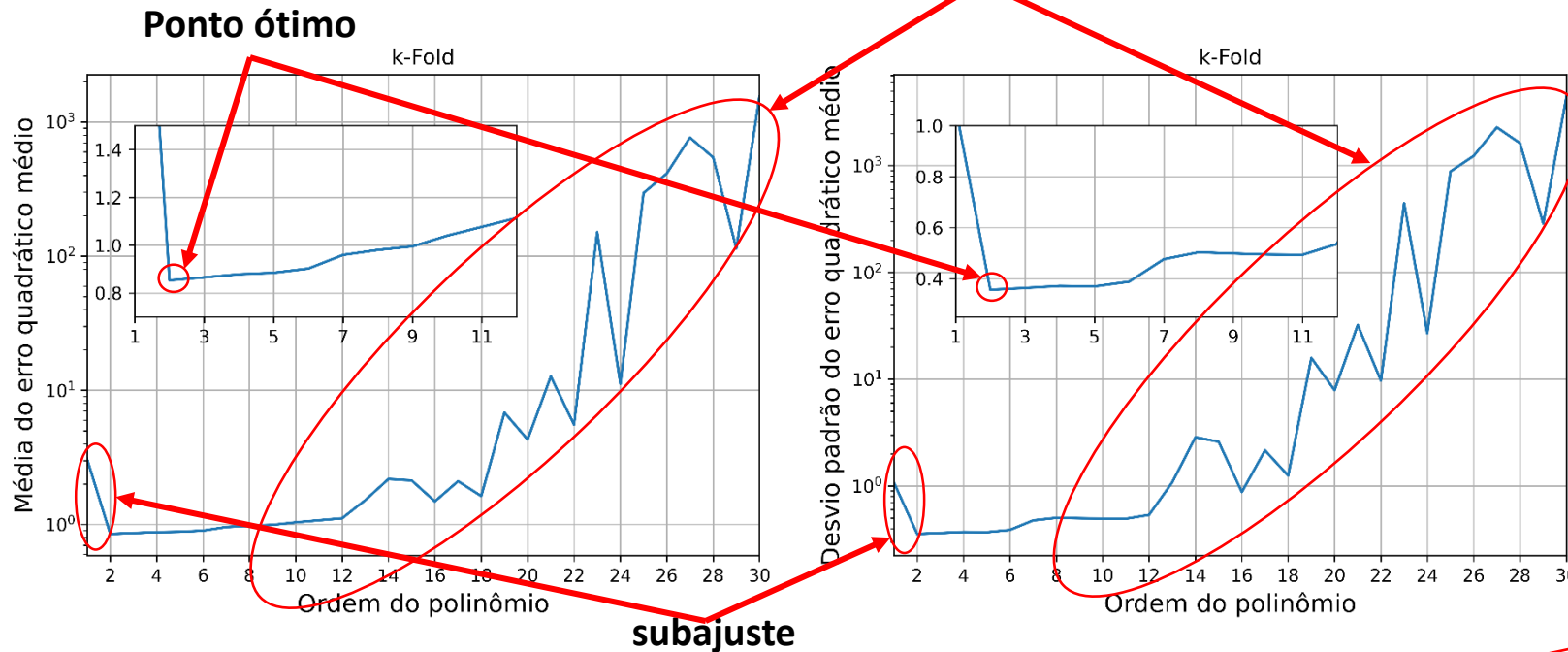
- Cada exemplo entra em um conjunto de validação exatamente **1** vez e em um conjunto de treinamento **k-1** vezes.
- O desempenho do modelo é dado pela **média dos erros de validação** calculados para cada um dos **k** folds.

Exemplo: [holdout_and_kfold_comparison_of_shuffling.ipynb](#)

k-Fold

- 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.
 - A **variância** do modelo resultante é reduzida à medida que **k** é aumentado.
- 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.
- O k-Fold é bastante útil quando se tem conjuntos de dados pequenos.
- **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



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 igual a 0 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 validação cruzada.
- **Scoring**: quanto mais positivo, melhor.



```
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.

[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

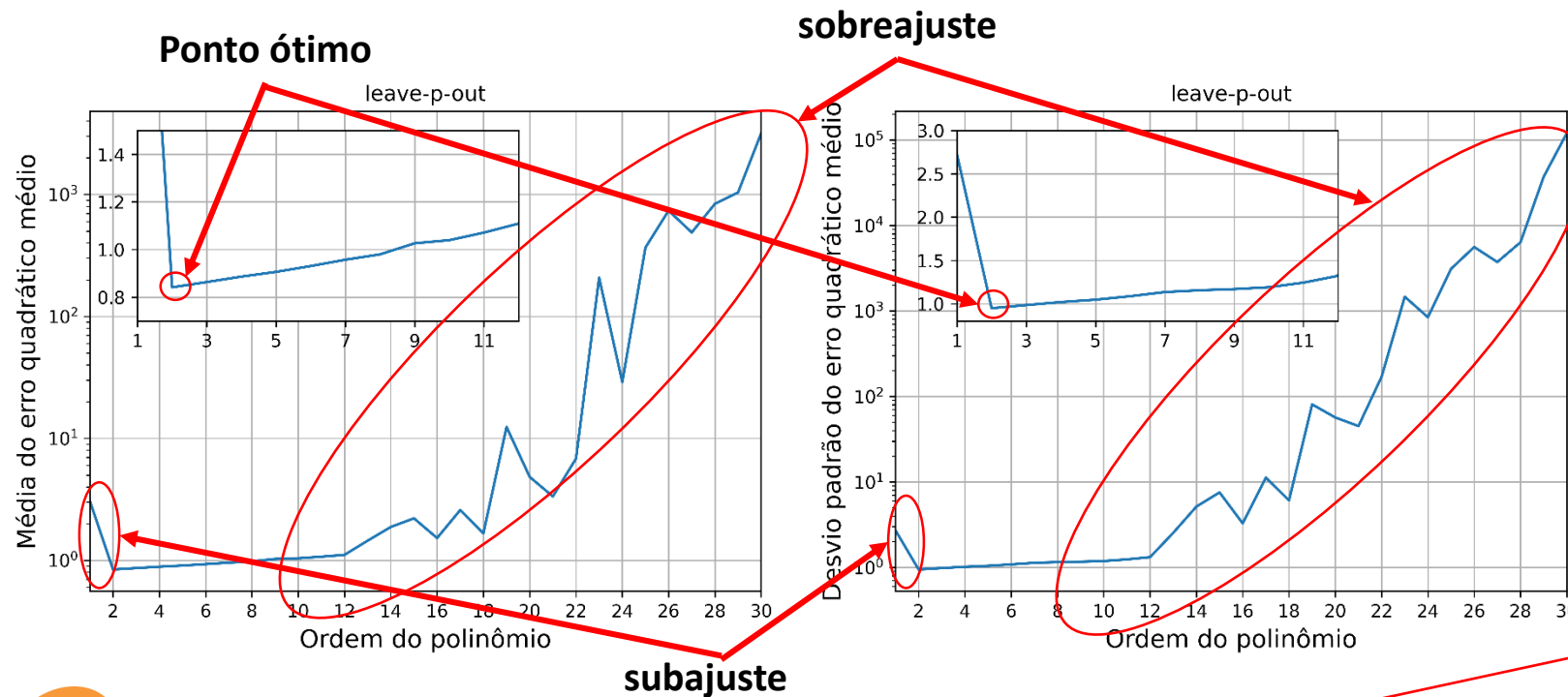
$$\binom{N}{p} = \frac{N!}{p!(N-p)!},$$

Quantos subconjuntos de p exemplos posso 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 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 fazemos ***k=N*** (número ***folds*** igual ao número total de exemplos), então o k-Fold é equivalente à estratégia do **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.

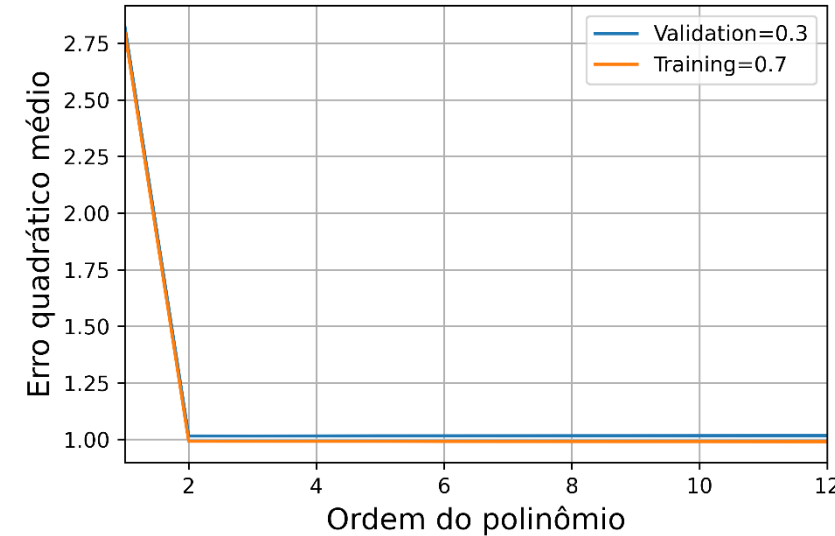
[Exemplo: validacao_cruzada.ipynb](#)

Qual estratégia utilizar?

- O **leave-p-out** 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, ele é bastante custoso 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-lo com **bases relativamente pequenas**.
- Para bases maiores, o **k-fold** é uma opção melhor e mais eficiente do que o **holdout**.
- Para bases muito grandes, o **holdout** já daria boas indicações sobre qual ordem utilizar.

Qual ordem escolher para o modelo?

- E se os erros de treinamento e validação são pequenos, similares 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.
- Portanto, escolhemos modelos usando a **navalha de Occam**: escolhemos a **função hipótese** menos complexa que se ajusta bem aos dados.

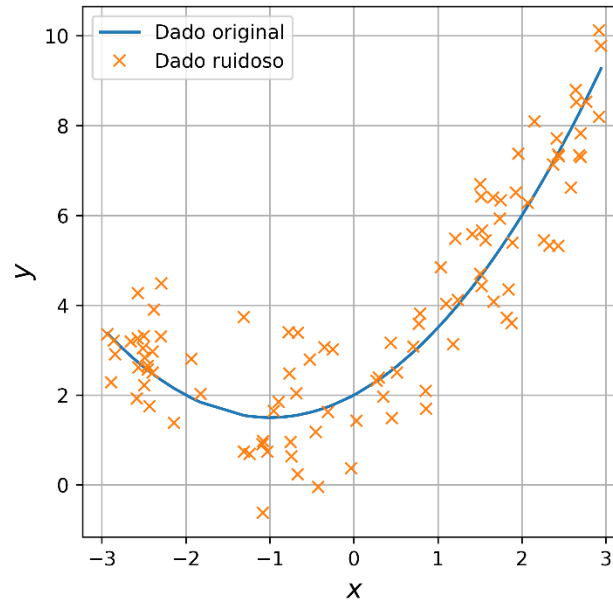


- Mesma função observável dos exemplos anteriores.
- Base de dados com **10000 exemplos**.
- Holdout com 30% para validação.
- Vejam que teoricamente, qualquer ordem maior ou igual a 2 já seria uma boa escolha.
- **Qual ordem escolher?**

Curvas de Aprendizado

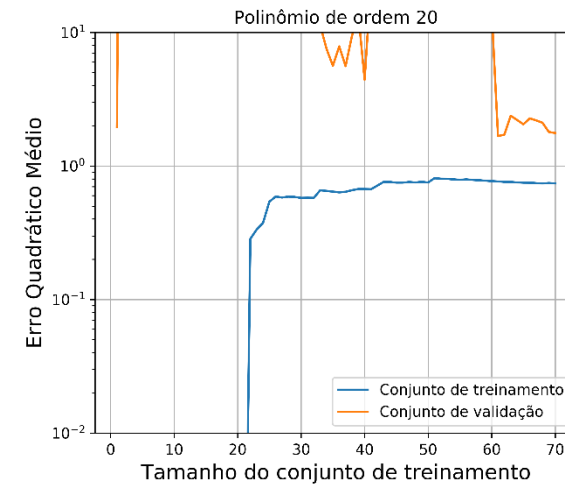
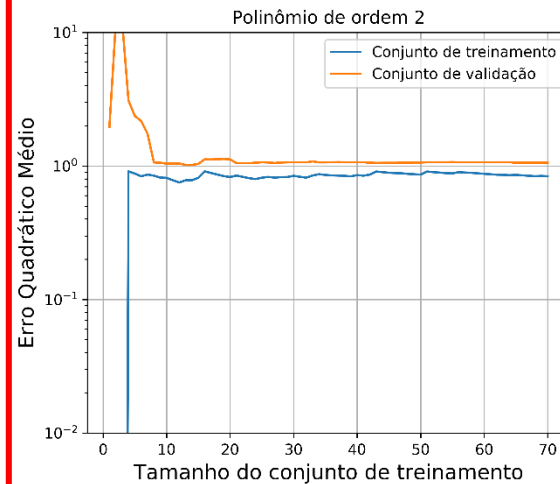
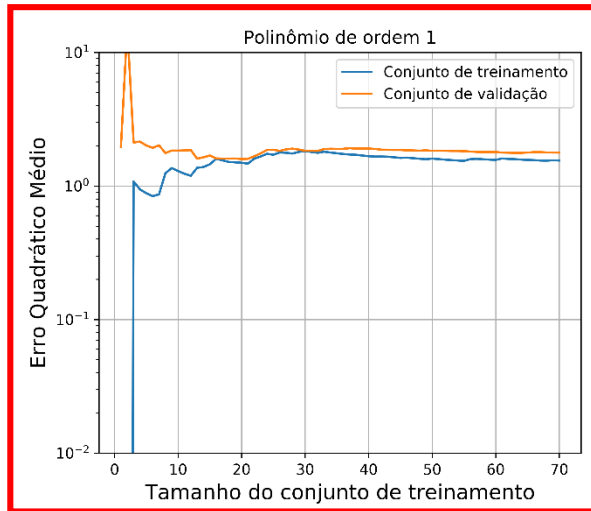
- **Curvas de aprendizado:** são gráficos que comparam o desempenho do modelo no conjunto de treinamento e no conjunto de validação em função do tamanho do conjunto de treinamento.
 - Tamanho do conjunto de validação permanece constante.
 - Caso não tenhamos dois conjuntos, treinamento e validação, normalmente 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



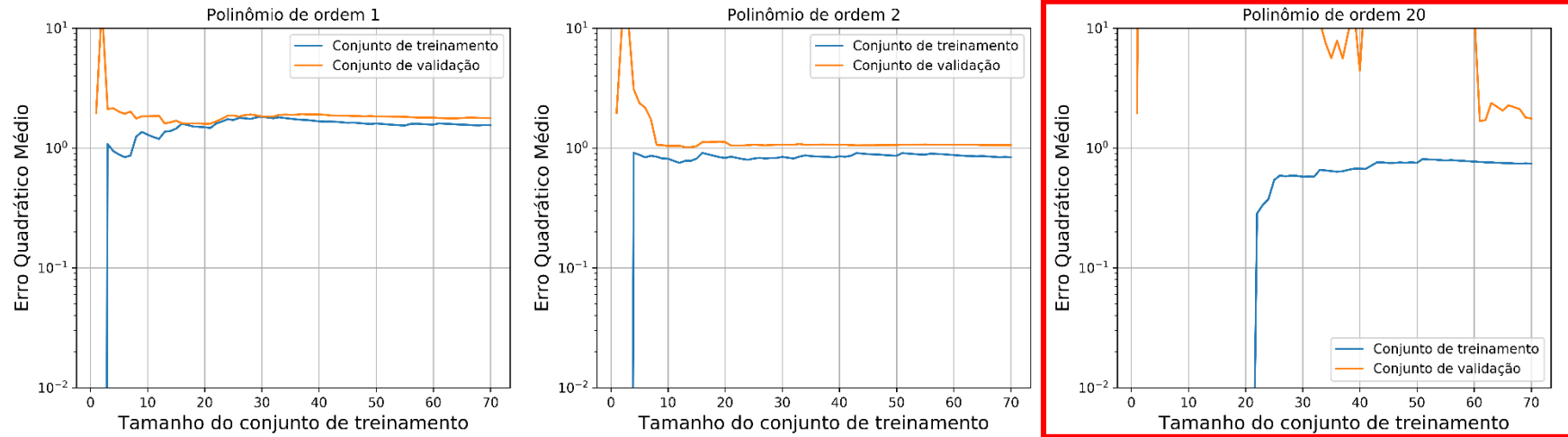
- Caso não conhecêssemos o ***mapeamento verdadeiro*** dos dados ruidosos da figura, qual ordem de polinômio melhor aproximaria o ***mapeamento verdadeiro***?

Curvas de Aprendizado: Polinômio de ordem 1



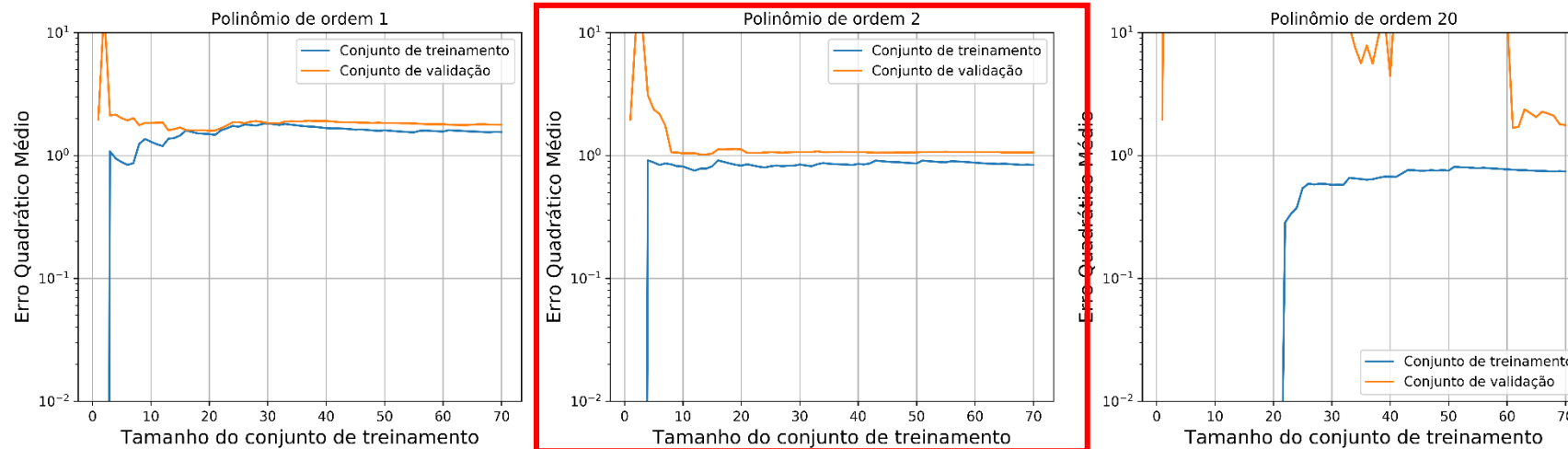
- Com 1 ou 2 exemplos no conjunto de treinamento, o modelo se ajusta perfeitamente (reta), porém, conforme o número de exemplos aumenta, é impossível para ele se ajustar (dados não-lineares e ruído).
- Erro nos exemplos de treinamento aumenta até atingir um platô e não diminui mesmo com o aumento do 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.

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 **sobreaajustando**.
- 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.

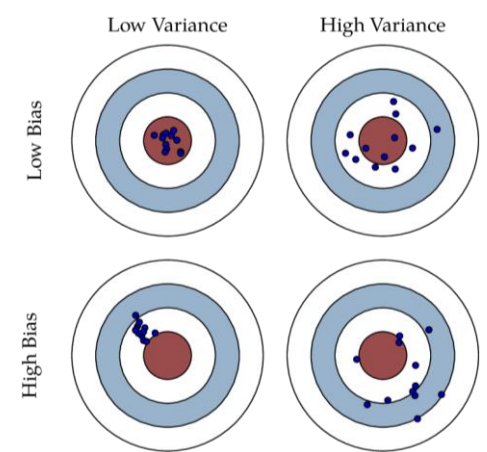
Curvas de Aprendizado: Polinômio de ordem 2



Exemplo:
[curvas_de_aprendizado.ipynb](#)

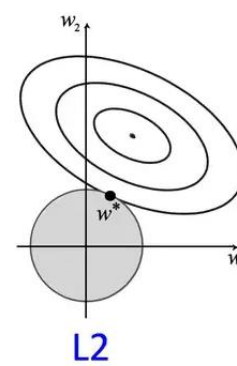
- A diferença entre os erros diminui com o aumento do conjunto de treinamento se tornando pequena.
- Tanto o erro de treinamento quanto o de validação são menores do que com o modelo de regressão linear, i.e., reta.
- Isso é a indicação de um modelo que está se ajustando bem aos dados de treinamento 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**.
- Elas são técnicas que visam se 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 **penalizações** proporcionais à alguma **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: *ridge 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).
- Essas técnicas permitem diminuir a **variância** do modelo ao custo de introduzir algum **viés**. Encontrar um bom balanço entre viés e variância permite minimizar o erro total do modelo.
- É importante aplicar o escalonamento de atributos quando se utiliza regularização. Desta forma, os pesos são penalizados de forma adequada.

Ridge regression



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

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

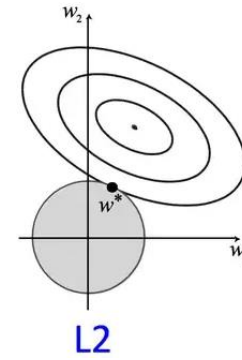
onde $\lambda \geq 0$ é o **fator de regularização**, Φ é a matriz de atributos e \mathbf{a} é o vetor de pesos.

- 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.:** o peso a_0 não é considerado no cálculo da **norma L2**, pois a **complexidade** se deve à ordem do modelo.
 - a_0 apenas dita o **deslocamento** em relação ao eixo y e não influencia na complexidade da **hipótese** pois não é multiplicado por nenhum atributo.
 - Assim, a norma L2 é dada por

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

↖ Início em 1 e não em 0.

Ridge regression



c limita os possíveis valores que os pesos podem assumir.

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

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

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 $c \rightarrow 0$, quando $a_i \rightarrow 0$.
 - Por outro lado, conforme c aumenta, maior será a magnitude dos pesos (i.e., eles podem assumir valores maiores), até que $c \rightarrow \infty$, quando $a_i \rightarrow \infty$.
 - 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

- A equação de erro regularizado, $\|\mathbf{y} - \Phi\mathbf{a}\|_2^2 + \lambda\|\mathbf{a}\|_2^2$, 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**:

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

- **OBS.1:** mesmo que a matriz Φ não possua **posto completo** (i.e., matriz singular), a inversa na equação acima sempre existirá por conta da adição do **termo de regularização** à diagonal principal da matriz quadrada $\Phi^T \Phi$.
- **OBS.2:** como a **norma L2** é diferenciável, os problemas de aprendizagem usando a regularização de Ridge também podem ser resolvidos iterativamente através do **algoritmo do gradiente descendente**.
- **OBS.3:** 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

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

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

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

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

← Gradiente com relação ao peso de bias

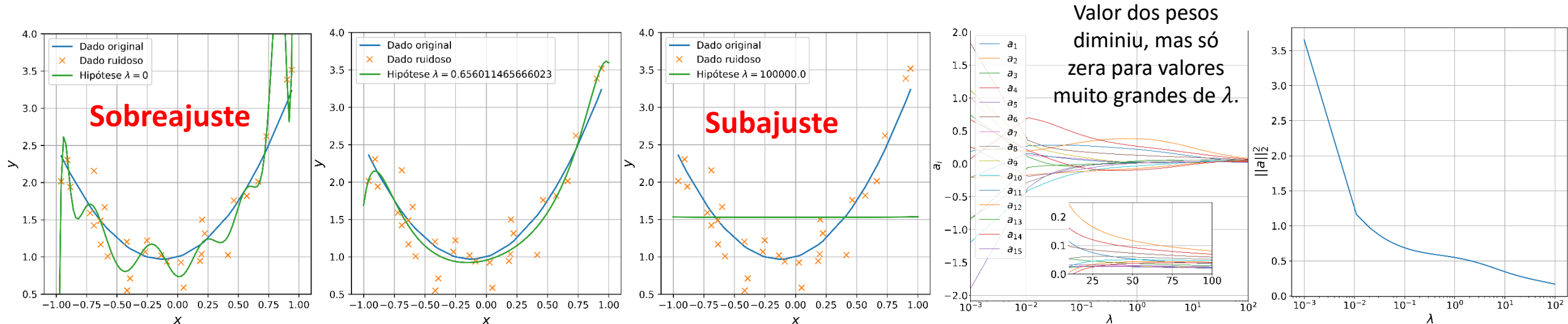
$$\frac{\partial J_e(\mathbf{a})}{\partial \mathbf{a}} = \frac{-2}{N} \Phi^T (\mathbf{y} - \Phi \mathbf{a}) + 2\lambda \mathbf{I}' \mathbf{a},$$

← Equação geral do vetor gradiente

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.

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

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 λ aumenta, o modelo não se “contorce” tanto e passa a se ajustar aos dados de treinamento.
- Se λ 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**.
- **O aumento de λ leva a hipóteses menos complexas.** Isso reduz a variância do modelo, mas aumenta seu bias. Ou seja, ele tende a **subajustar**.
- Conforme λ aumenta, os pesos e a norma L2 do vetor de pesos diminuem.
- Utiliza-se técnicas de **validação cruzada** para encontrar o valor ideal de λ .

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")

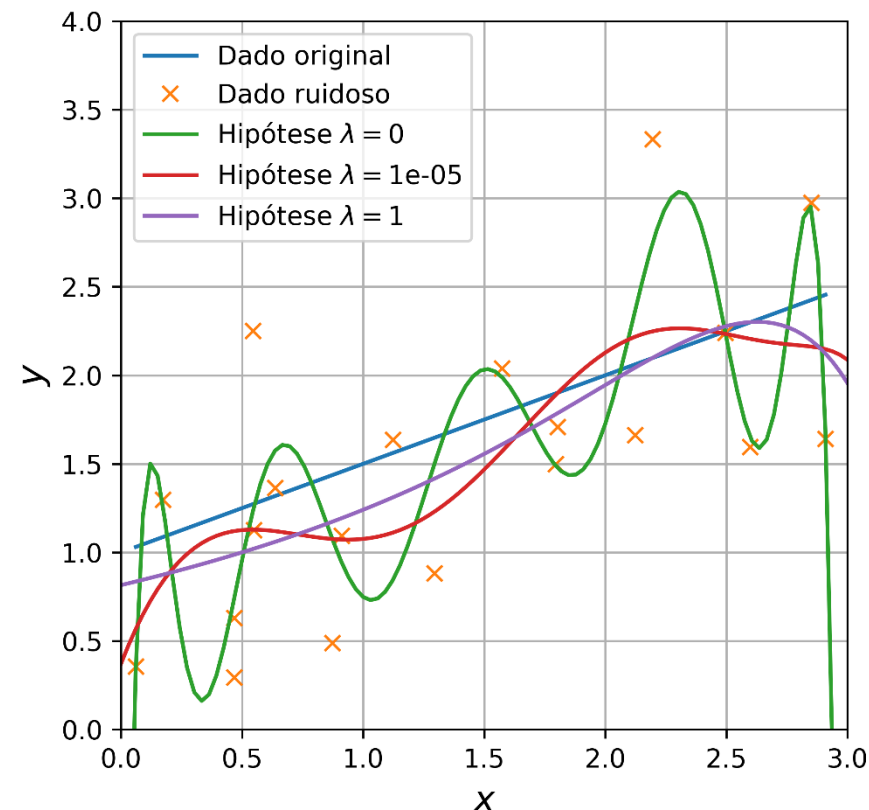
# Create a pipeline of actions.
model = Pipeline([("poly", poly_features), ("scaler", std_scaler), ("reg", reg)])

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

Importa classe

Polinômio de ordem 10

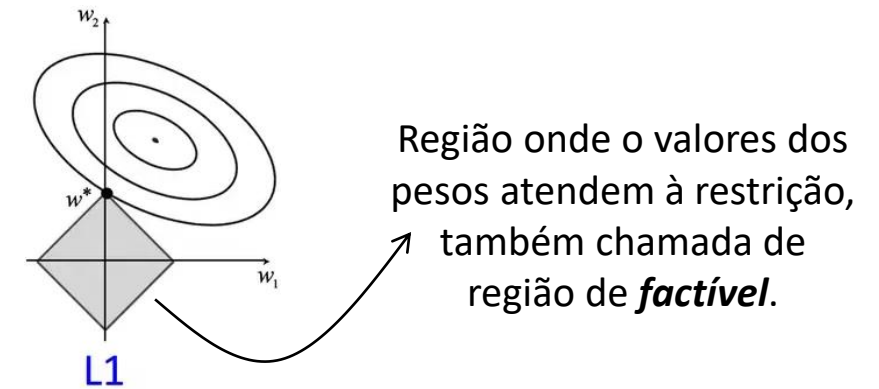
Instancia objeto



[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.

LASSO regression



- 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.

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

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

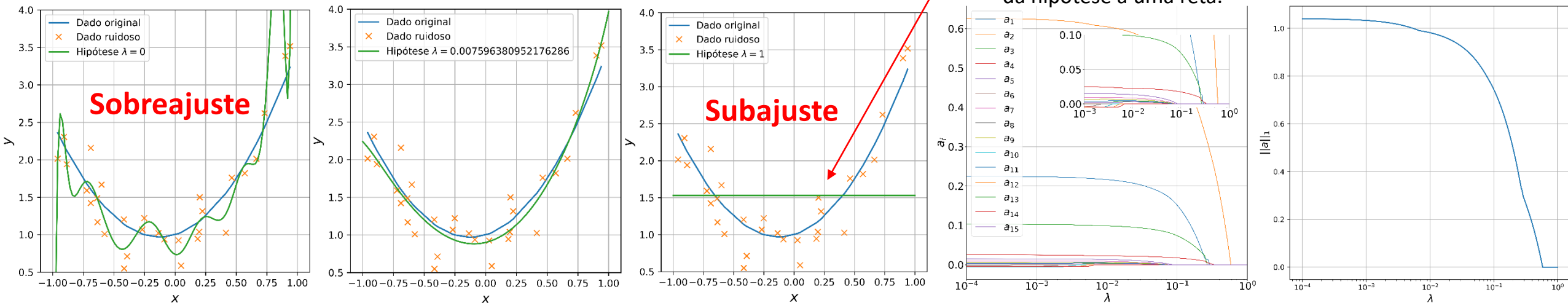
- Podemos re-escrever o **problema de regularização** acima como um **problema de otimização** com restrição da seguinte forma

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

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



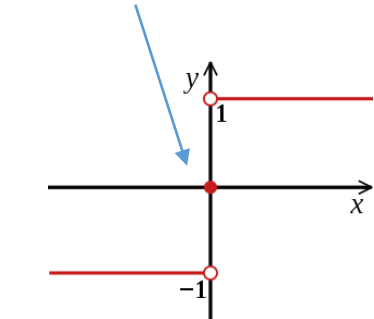
Valor dos pesos se torna igual a zero, restringindo a flexibilidade da hipótese a uma reta.

- Mesmas funções geradora e hipótese do exemplo anterior.
- Valores pequenos de λ 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.
- 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.
- Utiliza-se técnicas de validação cruzada para encontrar o valor ideal de λ .

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

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.

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

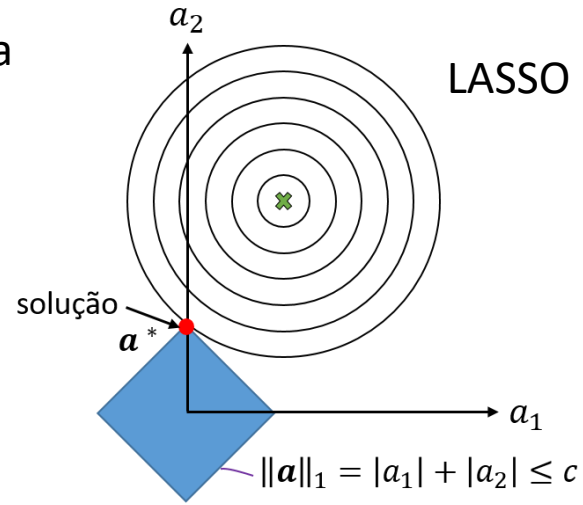
← Implementações práticas consideram que $\text{sign}(0) = 0$.

Vantagem do LASSO sobre Ridge

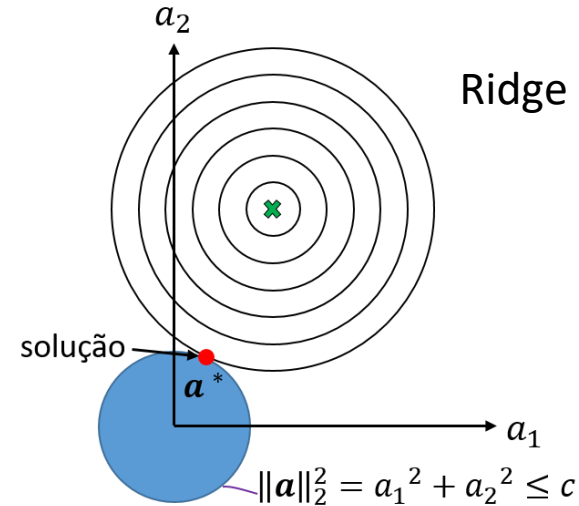
- A vantagem da regressão LASSO sobre a Ridge está na forma de diamante/losango da **região de factibilidade** criada pela penalidade da norma L1, o que leva à eliminação de alguns dos pesos (i.e., os pesos são zerados) rapidamente.
- Isso significa que a regressão LASSO realiza **seleção automática de atributos**, enquanto a regressão Ridge não apresenta esta característica, pois os pesos nunca são totalmente anulados.
- 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 (mais uniformemente) até mesmo pesos pequenos.

Vantagem do LASSO sobre Ridge

- 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 em algum lugar dentro do quadrado.



LASSO



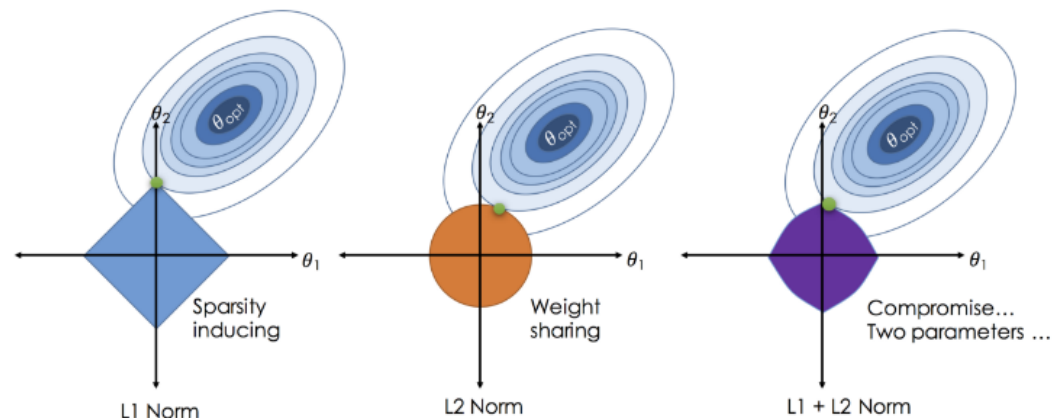
Ridge

- 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 em algum lugar dentro do círculo.

• Por que a regressão LASSO tem como vantagem a produção de modelos esparsos?

- A 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 L1 (esquerda) e L2 (direita) são válidas, considerando o caso em que dois pesos (a_1 e a_2) estão sujeitos a regularização.
- A solução para ambos os métodos corresponde ao ponto, dentro da **região de factibilidade** (área em azul), mais próximo do ponto de mínimo da função de erro.
- É fácil ver que para uma posição arbitrária do mínimo, será comum que um **canto** (ou ponta) do quadrado seja o ponto mais próximo do ponto de mínimo da função de erro.
- Os **cantos** na **região de factibilidade** da restrição L1 aumenta as chances de alguns pesos assumirem o valor zero.
- E claro, os **cantos** são os pontos que possuem um valor igual a 0 em alguma das dimensões (i.e., pesos).

Elastic-net



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

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

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

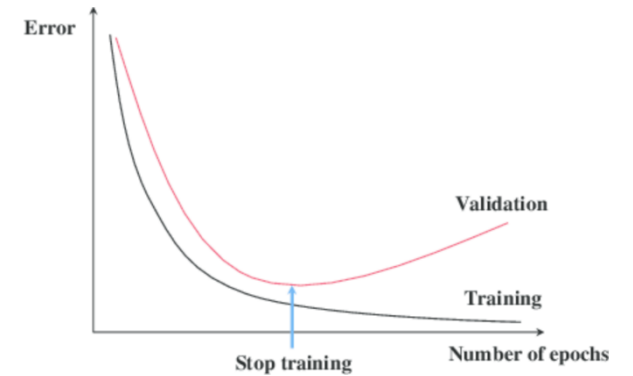
onde $\kappa \in [0, 1]$ é o **parâmetro de elasticidade** entre as duas normas.

- Quando $\kappa = 0$, é equivalente a Ridge regression, e quando $\kappa = 1$, 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 aos dois outros métodos anteriores.

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 erráticamente se o *número de atributos, K , for maior que o número de exemplos, N , de treinamento*. 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, ele seleciona um deles aleatoriamente, o que não é bom para a interpretação do modelo.
 - 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.
 - Quando há vários atributos correlacionados entre si, LASSO provavelmente escolherá um deles aleatoriamente, enquanto o Elastic-Net provavelmente *escolherá ambos*, facilitando a *interpretação do modelo*.
 - Uma proporção de 50% entre as penalizações L1 e L2 é uma boa escolha para esse parâmetro.

Early-stop



- 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-stop** ou **parada antecipada**.
- Pode ser vista como uma **regularização temporal**.
- 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 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-stop

- Existem basicamente duas estratégias para se definir o critério de parada:
 - Interromper o treinamento quando o valor do erro de validação aumentar 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 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 a construção do modelo.

Exemplo: Early-stop

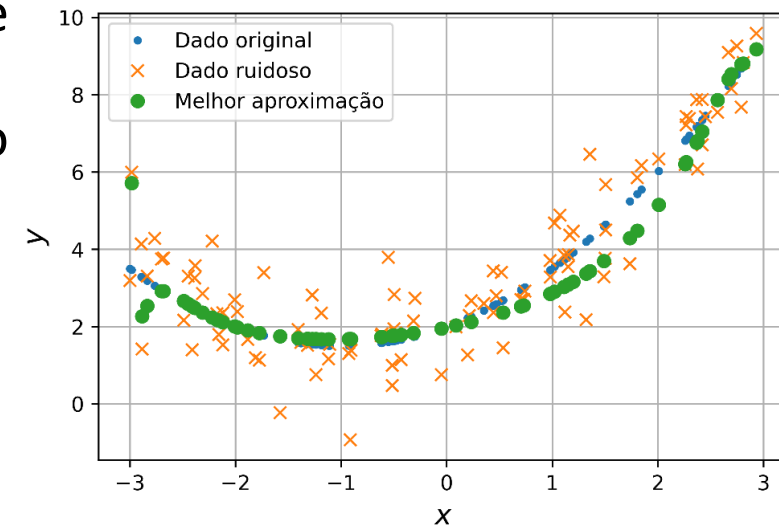
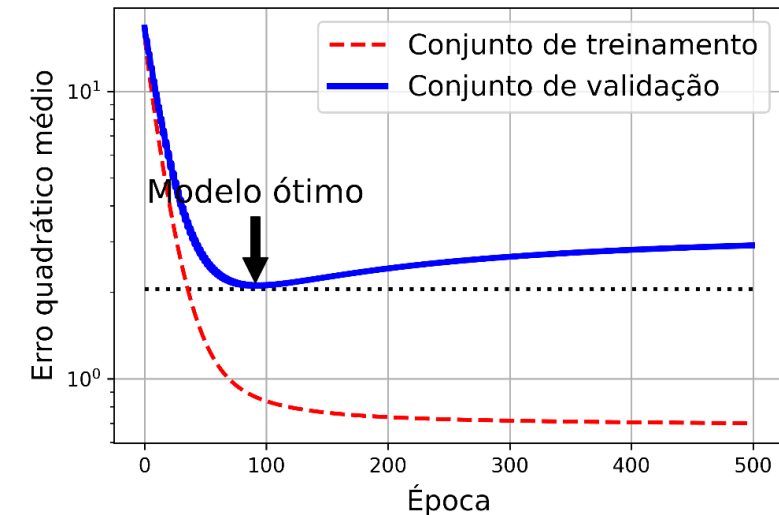
- A figura mostra a curva de aprendizado de modelo de regressão polinomial com grau igual a 90 treinado usando-se o gradiente descendente estocástico com apenas 100 amostras.

- A função observável é dada por

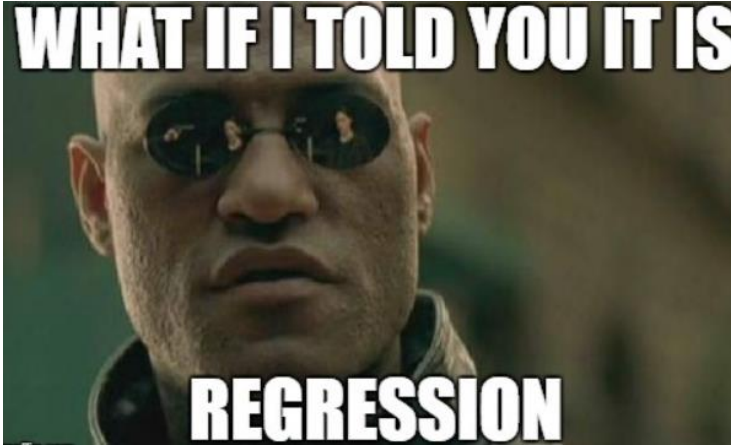
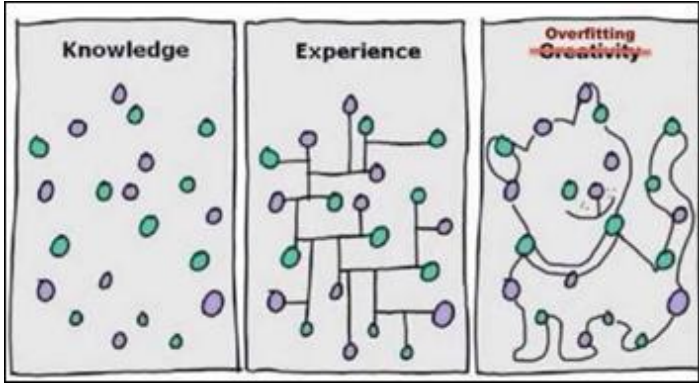
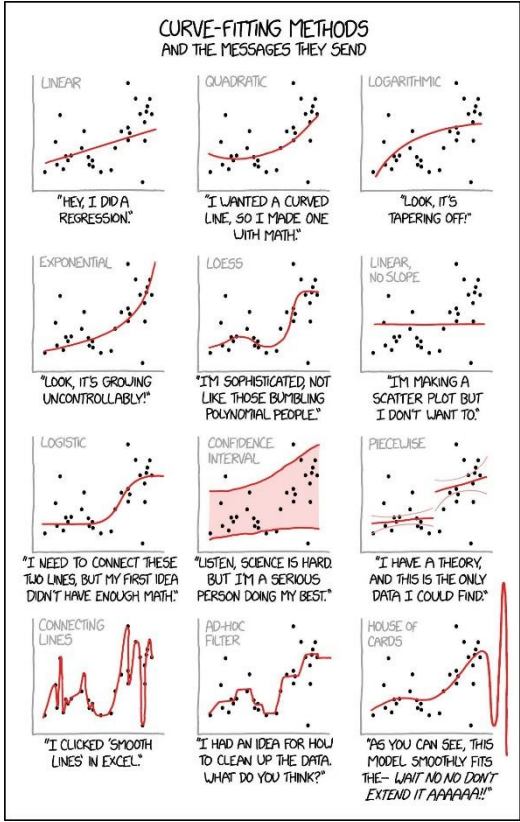
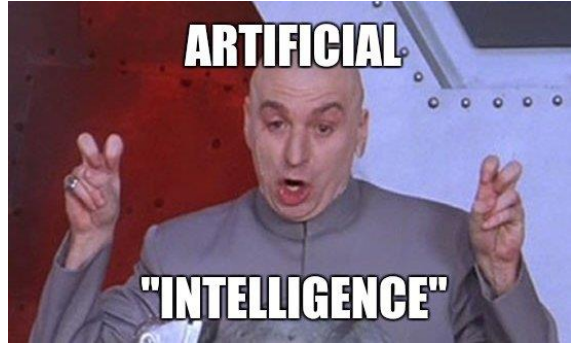
$$y_{\text{noisy}} = 2 + x + 0.5x^2 + w,$$

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

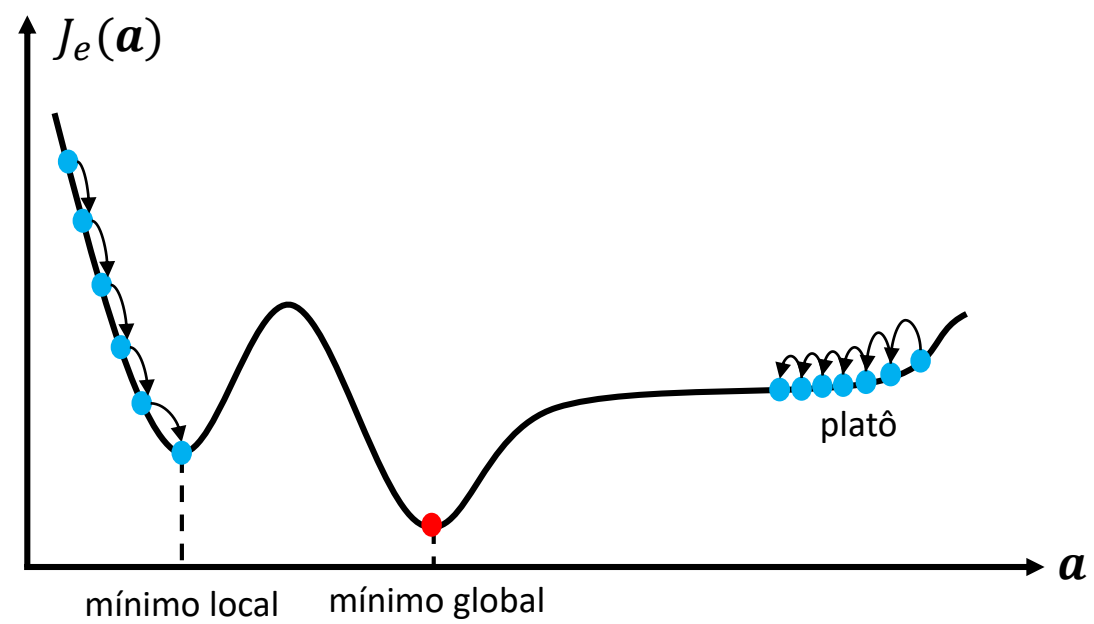
- A ordem do modelo é muito maior (muito flexível) do que a ordem da função verdadeira, além de ser próxima do número de amostras de treinamento (alta probabilidade de sobreajuste).
- À medida que as épocas passam, o algoritmo aprende e seu erro de treinamento diminui, juntamente com o erro de validação.
- No entanto, após 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, usa-se os pesos que resultaram no menor erro de validação, garantindo que o modelo que tenha uma boa **generalização**.

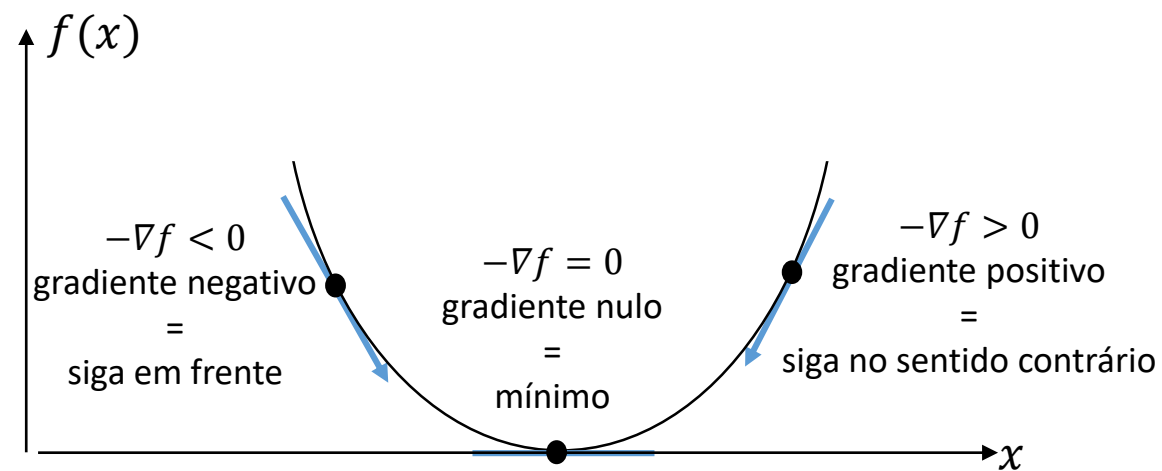
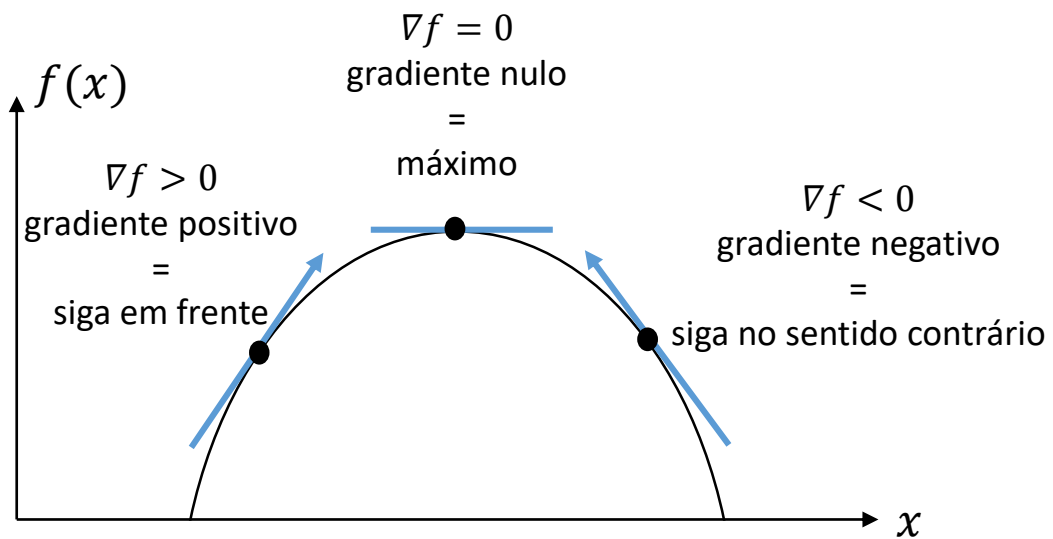


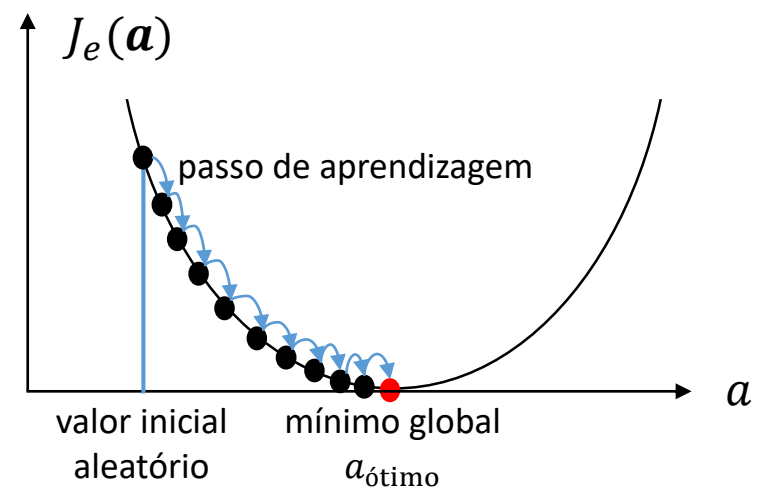
Obrigado!

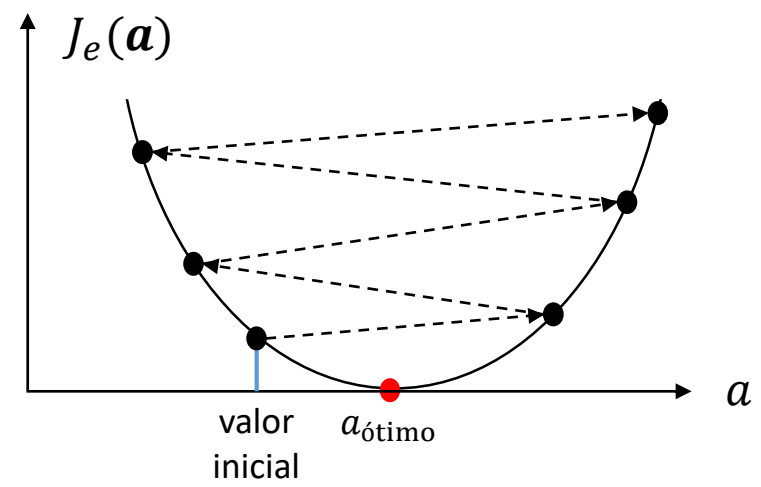


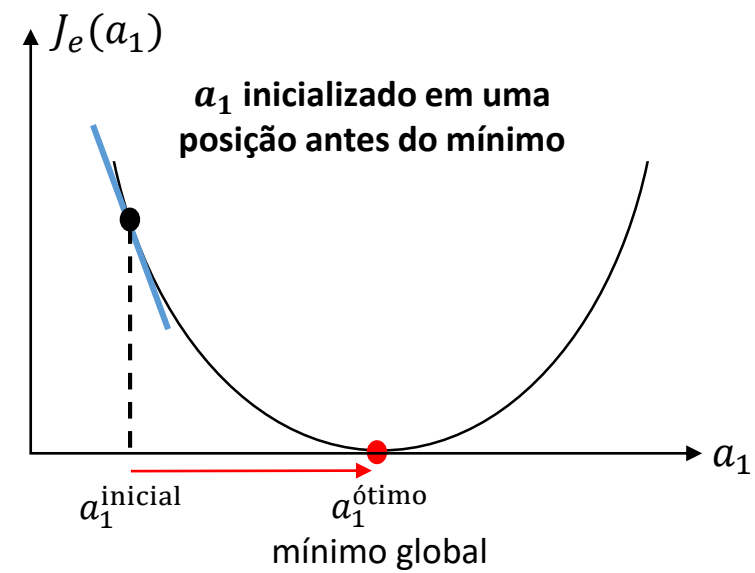
FIGURAS



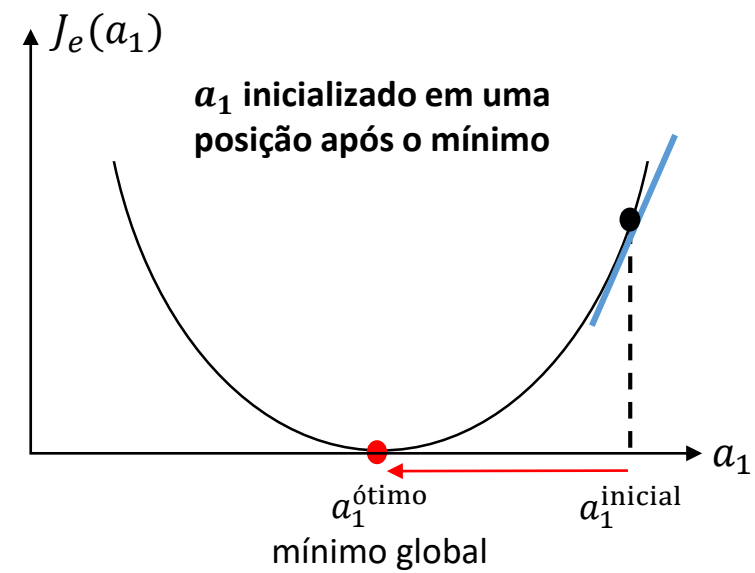




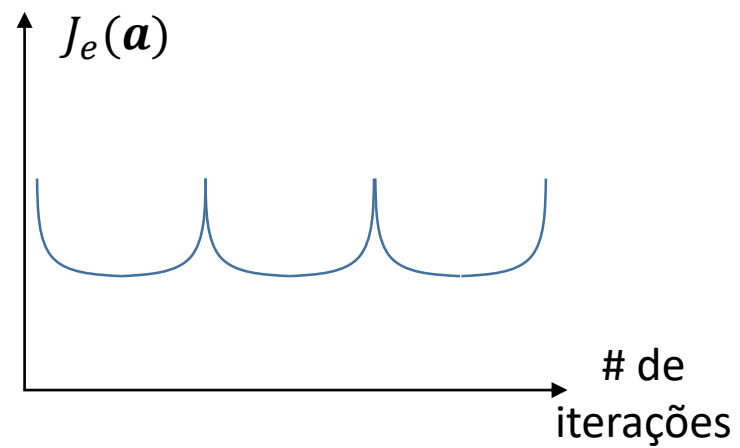
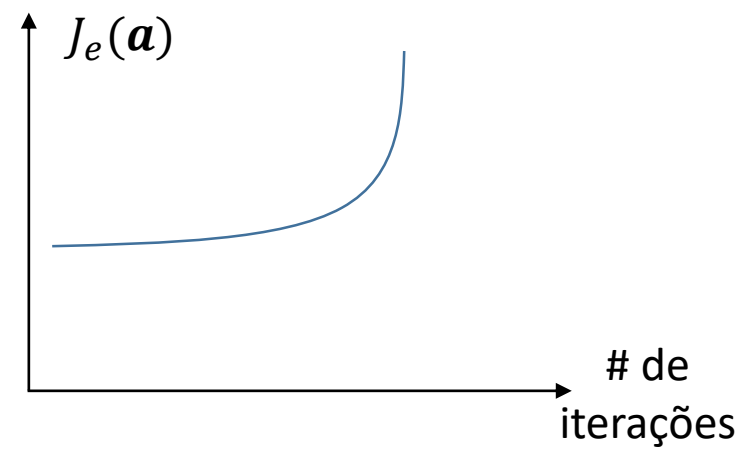
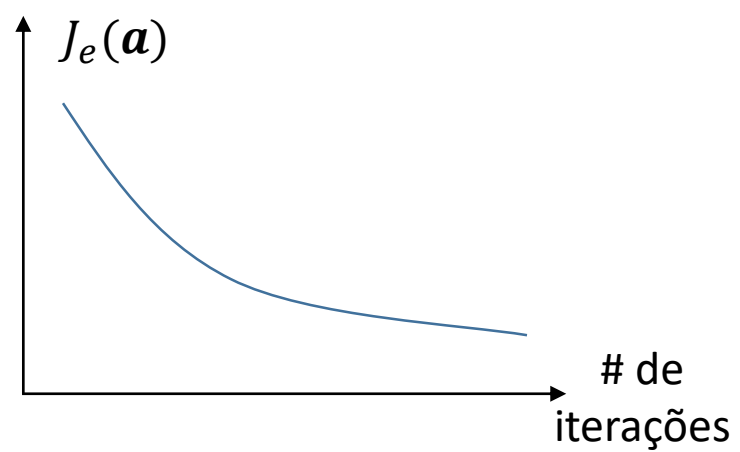
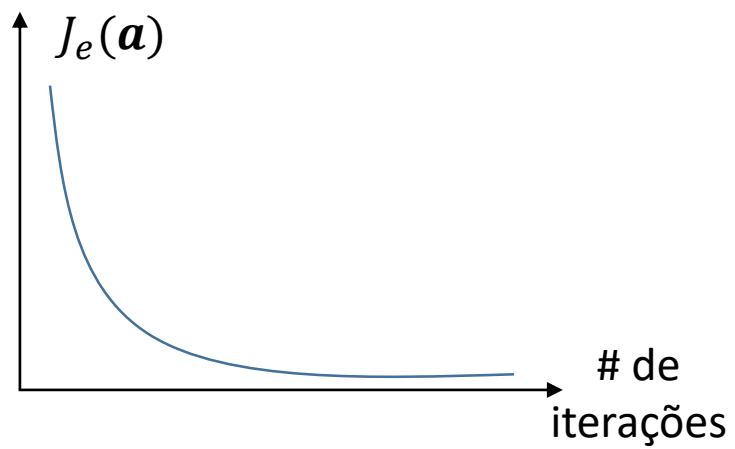


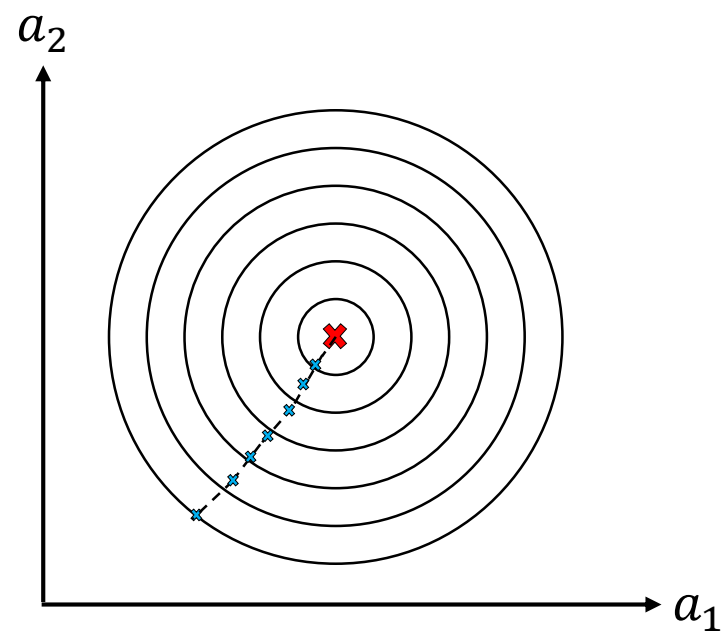
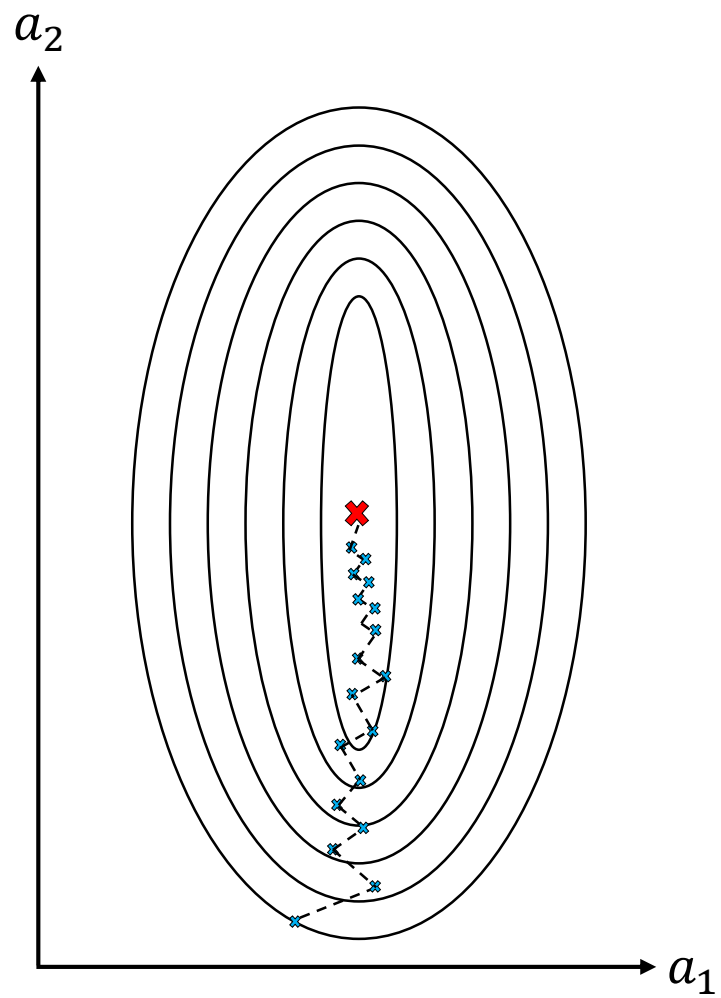


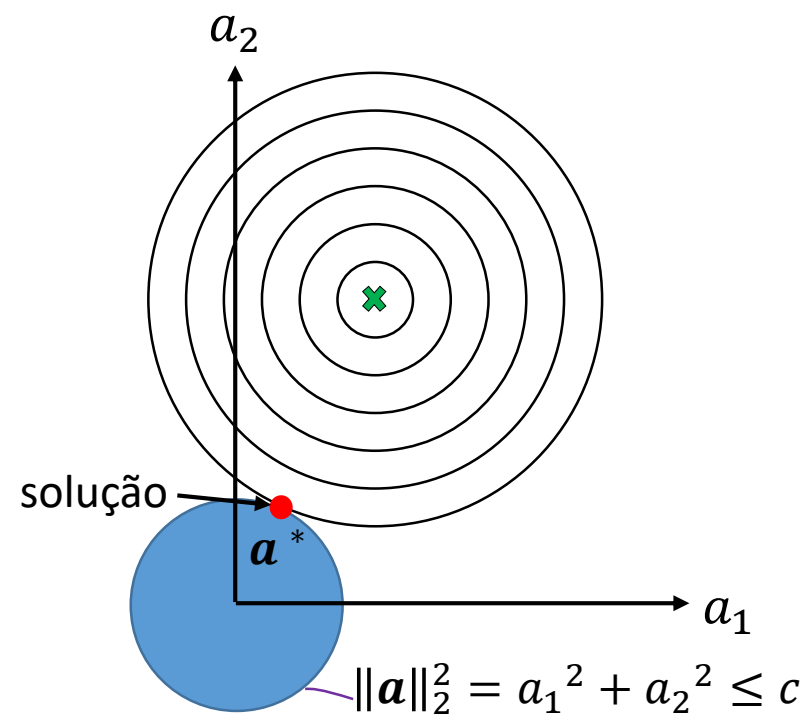
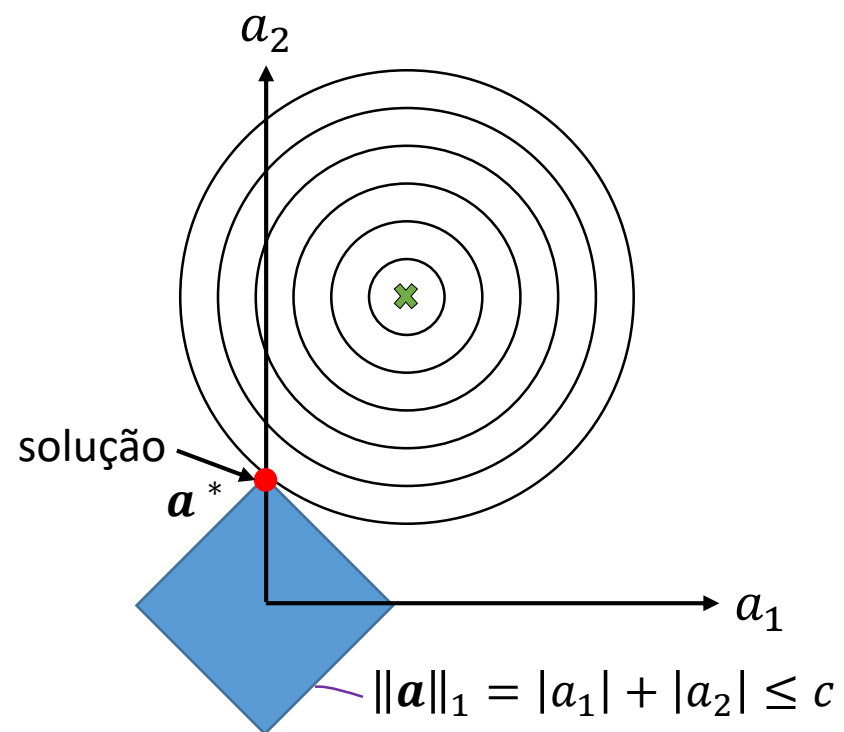
gradiente negativo: $a_1 = a_1^{\text{inicial}} + \alpha \nabla J_e(a_1)$
 a_1 aumenta e se aproxima do mínimo

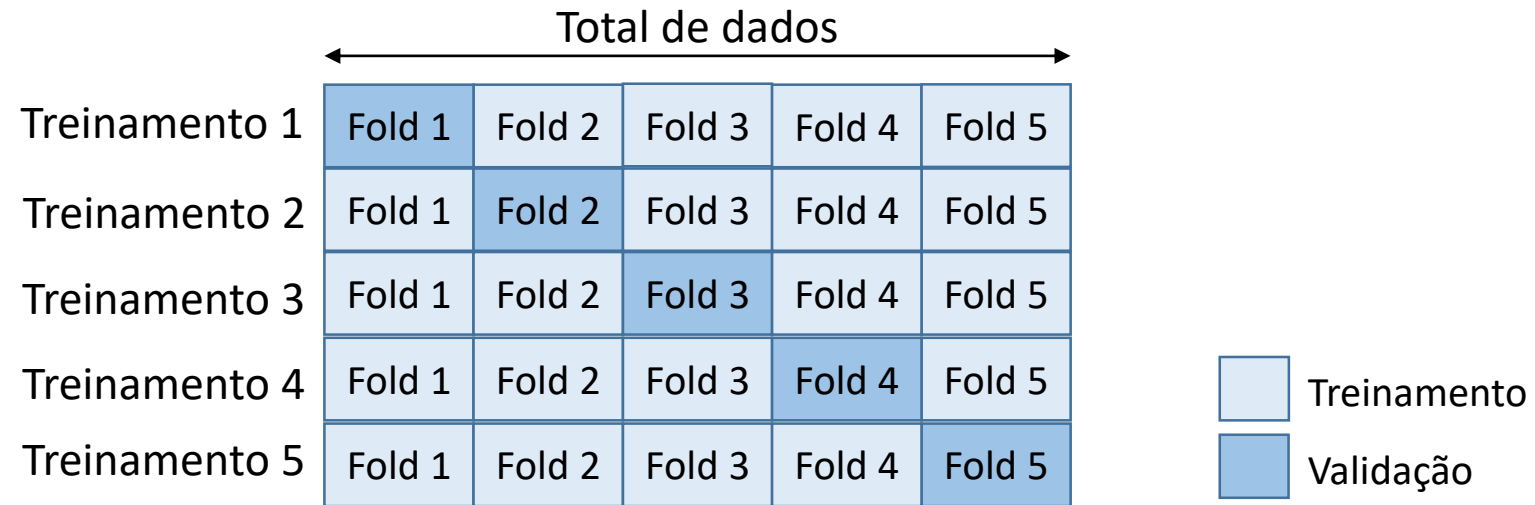


gradiente positivo: $a_1 = a_1^{\text{inicial}} - \alpha \nabla J_e(a_1)$
 a_1 diminuiu e se aproxima do mínimo









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

