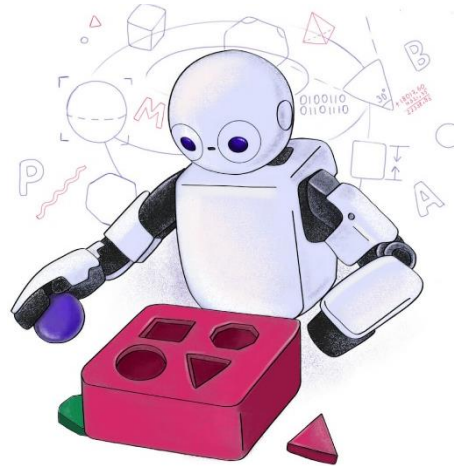


# T319 - Introdução ao Aprendizado de Máquina: *Regressão Linear (Parte IV)*



***Inatel***

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

# Recapitulando

- Vimos que o **valor do passo de aprendizagem influencia no processo aprendizagem** do gradiente descendente.
  - Valores pequenos fazem com que o algoritmo tenha convergência muito lenta.
  - Valores muito grandes fazem com que o algoritmo divirja.
- O gráfico do erro versus iterações nos ajuda a **depurar as versões do GD**.
- Quando usamos as versões estocásticas do GD, podemos **reduzir o valor do passo de aprendizagem ao longo do treinamento** para “**forçar**” a **convergência** do algoritmo.
- Nesse tópico, veremos
  - Técnicas de **pré-processamento** importantes para algoritmos de ML que usam **métricas de distância como função de erro**.
  - Como **polinômios** podem ser usados para **se ajustar a dados que apresentam mapeamento não-linear** entre os atributos e o valor esperado.

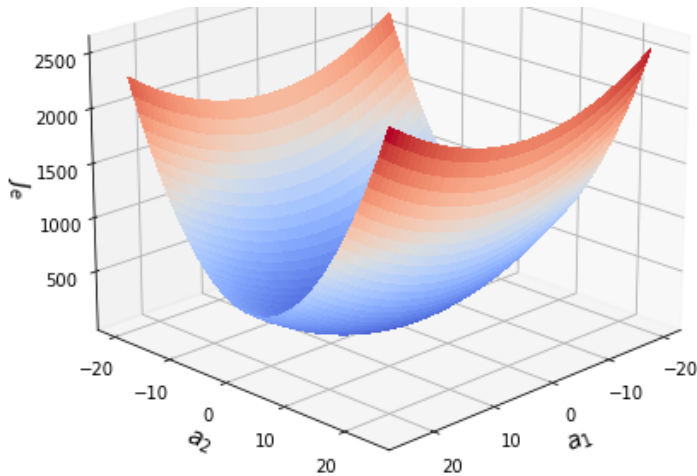
# Variações do formato da superfície de erro

- Como vimos no laboratório #3, nem toda superfície de erro criada a partir da função do EQM tem formato de tigela (i.e., com curvas de erro circulares).
- Dependendo dos *intervalos de variação dos atributos*, podemos ter *superfícies com formato de vale*.
- Por exemplo, se  $x_1 \gg x_2$ , ele *dominará o erro* e fará com que a superfície de erro tenha *formato de vale*.

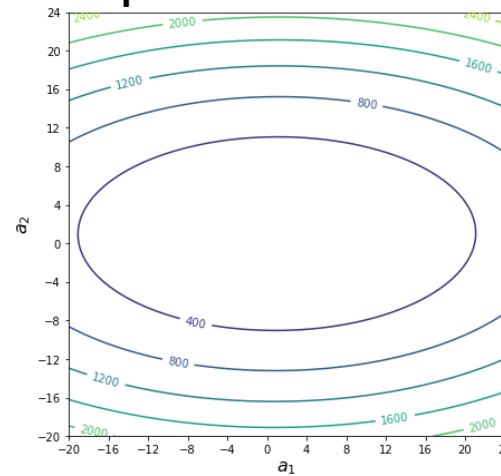
$$J_e(\mathbf{a}) = \frac{1}{N} \sum_{n=0}^{N-1} \left[ y_{\text{noisy}}(n) - \underbrace{(\hat{a}_1 x_1(n) + \hat{a}_2 x_2(n))}_{\text{Função hipótese}} \right]^2 \underset{x_1 \gg x_2}{\approx} \frac{1}{N} \sum_{n=0}^{N-1} [y_{\text{noisy}}(n) - \hat{a}_1 x_1(n)]^2$$

# Superfícies com formato de vale

Superfície de erro



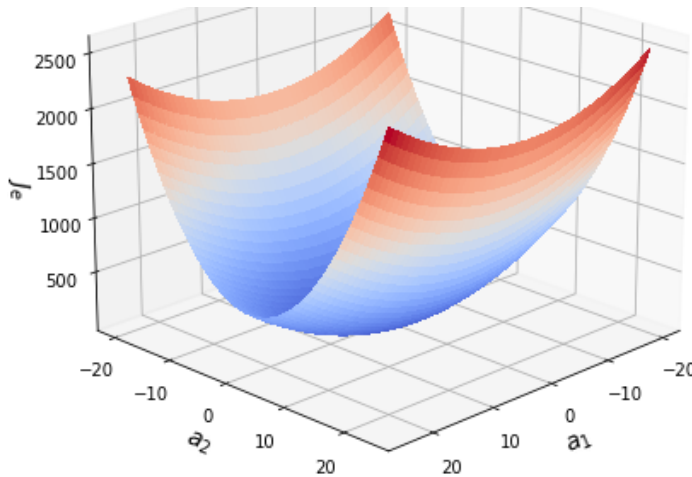
Superfície de contorno



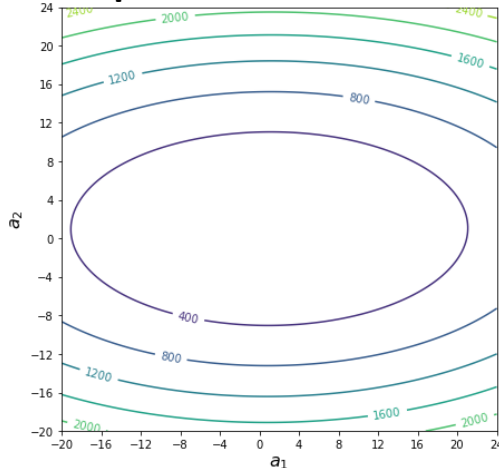
- Superfícies com *formato de vale* fazem com que a *convergência do GD se torne muito lenta*.
- A *convergência* se torna *lenta devido à superfície ser plana ou quase plana* em algumas direções (i.e., inclinação  $\approx 0$ ).
  - Vetor gradiente tende a zero.

# Superfícies com formato de vale

Superfície de erro



Superfície de contorno



- Nessas direções, o **gradiente da função de erro é muito pequeno**, tornando as **atualizações dos pesos**, consequentemente, **muito pequenas**.

$$\mathbf{a} \leftarrow \mathbf{a} - \alpha \frac{\partial J_e(\mathbf{a})}{\partial \mathbf{a}}$$

- Na figura ao lado, as derivadas parciais do EQM em relação ao peso  $a_1$  serão muito pequenas devido à **pequena inclinação da superfície nessa direção**.

***O que pode ser feito?***

# Escalonamento de atributos

- Para evitar esse problema, o *intervalo de variação de todos os atributos* pode ser *escalonado*, trazendo-os para uma escala similar.
- Assim, *cada atributo influenciará da mesma forma o cálculo do erro*.

$$J_e(\mathbf{a}) = \frac{1}{N} \sum_{n=0}^{N-1} \left[ y_{\text{noisy}}(n) - \underbrace{(\hat{a}_1 x_1(n) + \hat{a}_2 x_2(n))}_{\text{Função hipótese}} \right]^2 .$$

- Consequentemente, a superfície se tornará mais circular, facilitando o aprendizado.
- As duas formas mais comuns de escalonamento são:
  - Normalização Mín-Max
  - Padronização

# Escalonamento de atributos

## Observações:

- Em geral, aplicamos o escalonamento *apenas aos atributos* e não aos rótulos, pois são os atributos que influenciam o formato da superfície de erro.
- Além disso, ao escalonar os rótulos, perde-se seu significado.
- Por exemplo, a predição do preço de casas deixa de ser em reais.
- O escalonamento (qualquer tipo) altera os valores dos pesos originais.
  - Se a escala dos atributos é alterada, para que o modelo ainda prediga os mesmos valores de saída (i.e., rótulos), os pesos precisam ter seus valores alterados (ver anexo I).
- O escalonamento pode ser usado com qualquer versão do GD.

# Escalonamento de atributos

- Em geral, a **normalização mín-max** faz com que os **atributos variem entre 0 e 1**, mas pode-se definir outros intervalos.
- A equação usada para normalizar os atributos é apresentada abaixo

$$x'_k(n) = \frac{x_k(n) - \min(\mathbf{x}_k)}{\max(\mathbf{x}_k) - \min(\mathbf{x}_k)}, 0 \leq x'_k(n) \leq 1,$$

onde  $x_k$  representa o  $k$ -ésimo atributo,  $n$  é o número da amostra,  $\min(\mathbf{x}_k)$  e  $\max(\mathbf{x}_k)$  são os valores mínimo e máximo, respectivamente, **calculados ao longo de todas as amostras do  $k$ -ésimo vetor de atributos,  $\mathbf{x}_k$ .**

# Escalonamento de atributos

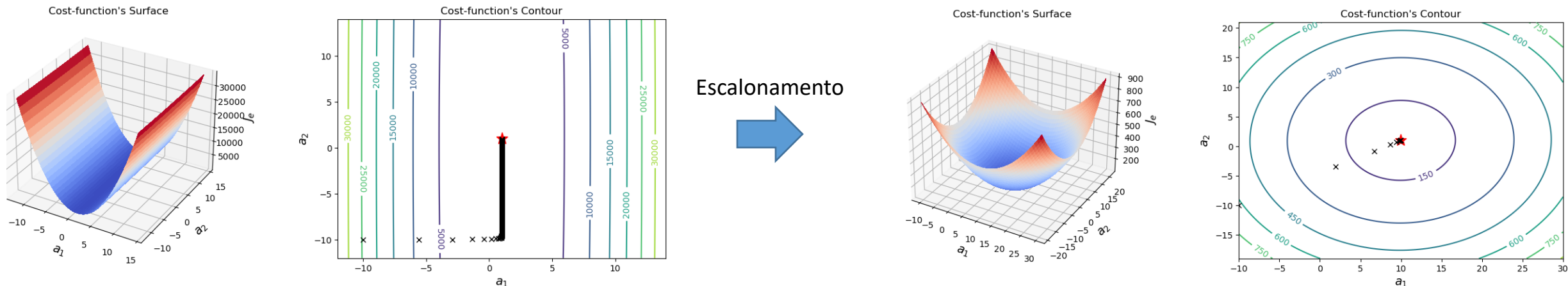
- **Padronização** faz com que os *atributos passem a ter média zero e desvio padrão unitário*.
- Notem que, neste caso, os *valores não ficam restritos a um intervalo específico*.
- A equação usada para padronizar os atributos é apresentada abaixo.

$$x'_k(n) = \frac{x_k(n) - \mu_{x_k}}{\sigma_{x_k}},$$

onde  $x_k$  representa o  $k$ -ésimo atributo,  $n$  é o número da amostra,  $\mu_{x_k}$  e  $\sigma_{x_k}$  são as estimativas da média e do desvio padrão, respectivamente, *calculados ao longo de todas as amostras do  $k$ -ésimo vetor de atributos*,  $x_k$ .

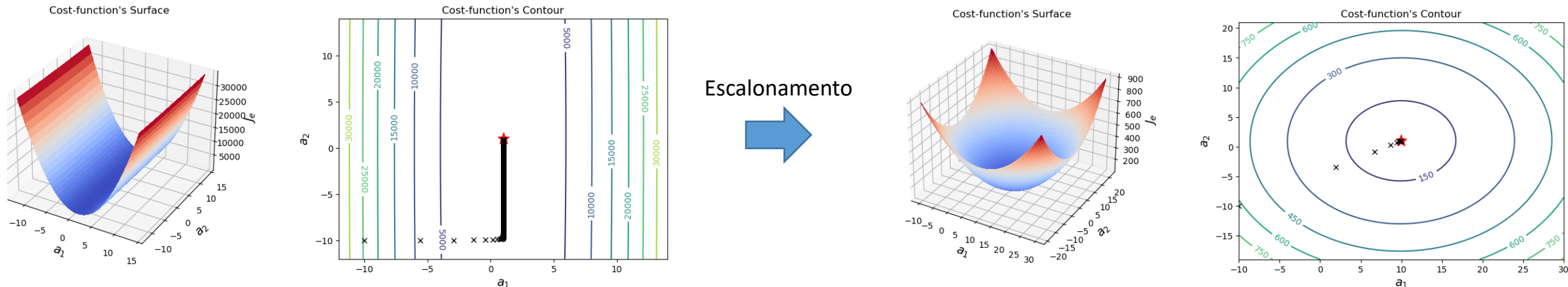
# Vantagens do escalonamento de atributos

- Ajuda a **acelerar a convergência** do gradiente descendente, pois deixa a superfície de erro mais circular
  - Pois a inclinação da superfície se torna similar em todas as direções.
- Reduz a probabilidade de **problemas de precisão numérica**, mantendo a estabilidade do algoritmo durante o treinamento.
  - Por exemplo, **atributos com valores muito grandes podem gerar erros extremamente grandes** que podem não ser representados pelas variáveis.



# Vantagens do escalonamento de atributos

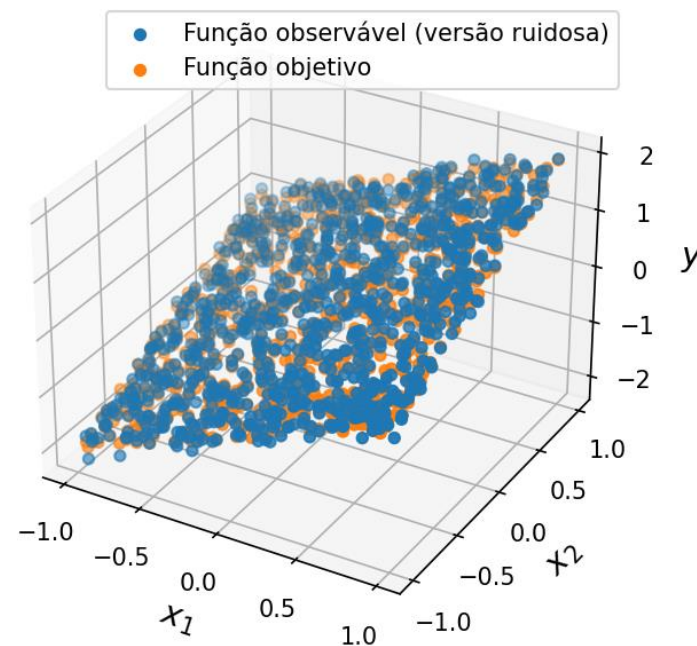
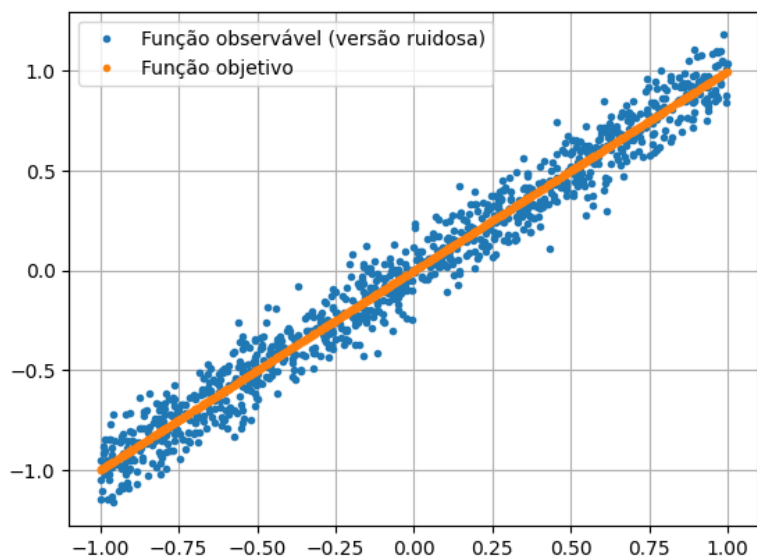
- Possibilita a **comparação justa do peso/influência** de cada **atributo** no modelo.
  - Pois os pesos representam o impacto relativo dos atributos nas previsões.
- Evita que **atributos com escalas muito diferentes dominem o processo de treinamento**.
  - Sem escalonamento, o modelo pode dar mais importância a atributos com intervalos maiores e menos importância a atributos com intervalos menores.



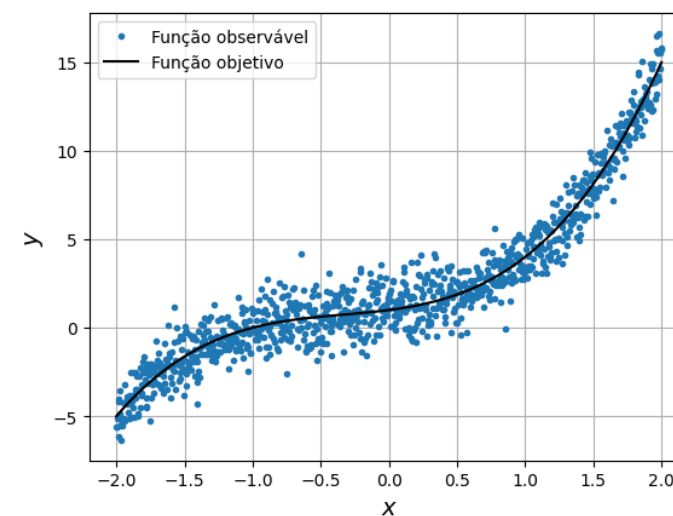
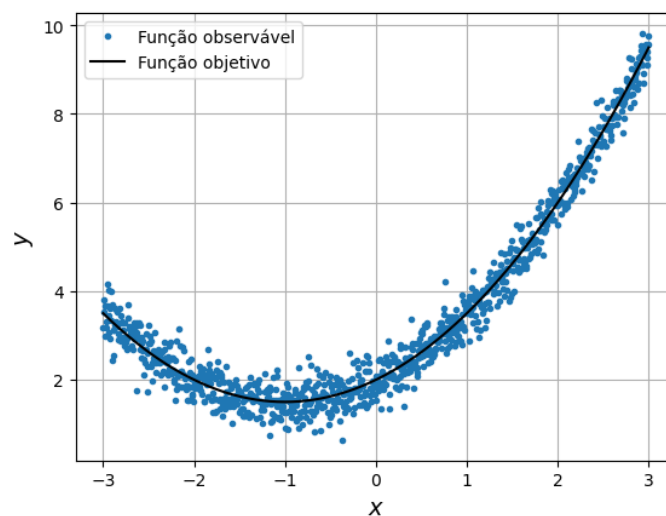
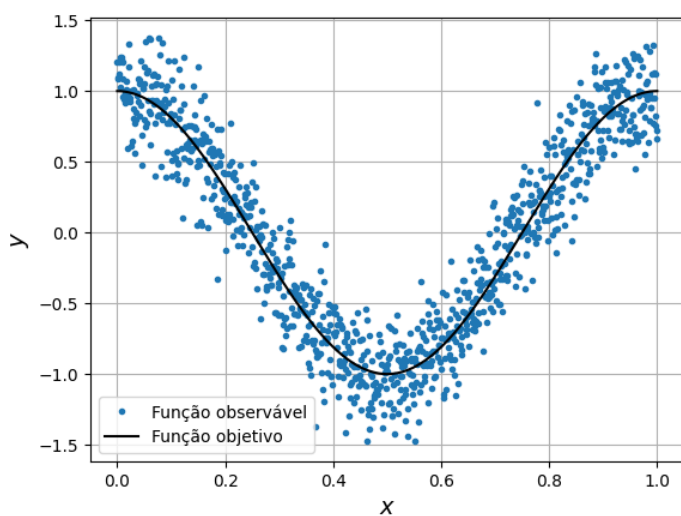
# Avisos

- **Exercício Prático:** Laboratório #5 (**Apenas o exercício #1**)
  - Pode ser acessado através do link acima (Google Colab) ou no GitHub.
  - Vídeo explicando o laboratório: Arquivos -> Material de Aula -> Laboratório #5
  - Se atentem aos prazos de entrega.
  - [Instruções para resolução e entrega dos laboratórios.](#)
- **Avaliação Presencial:** **07/11/2025 – Sala I-4**
  - Avaliação e projeto podem ser feitos em grupos de no máximo 3 alunos.
  - **Presencialmente, faremos apenas o exercício 1 do projeto final.**
  - Os outros exercícios devem ser entregues até **28/11/2024.**

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



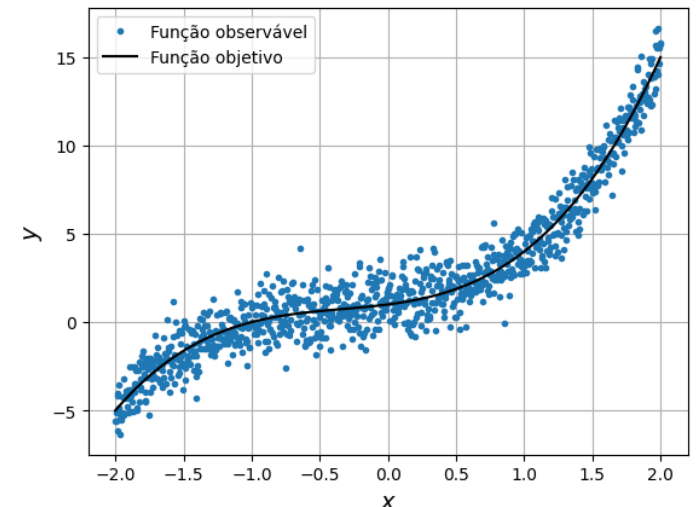
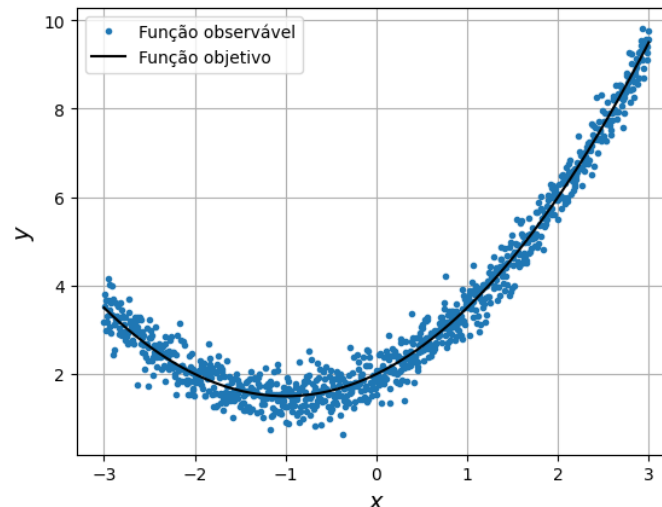
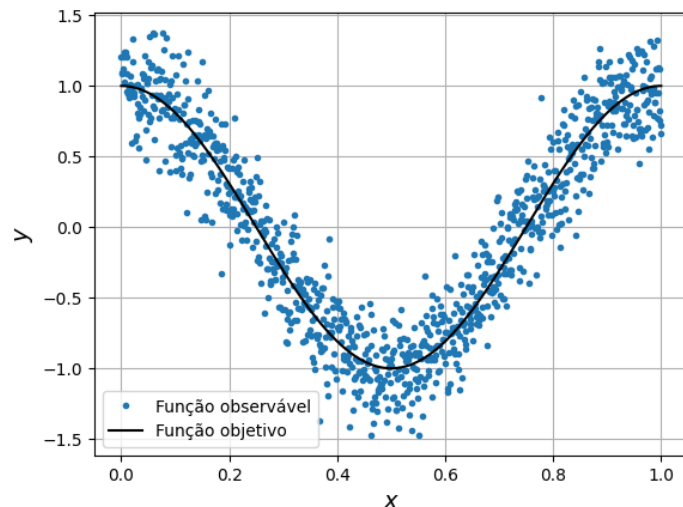
*Mas e se o mapeamento entre os atributos e o valor esperado for **não linear**?*



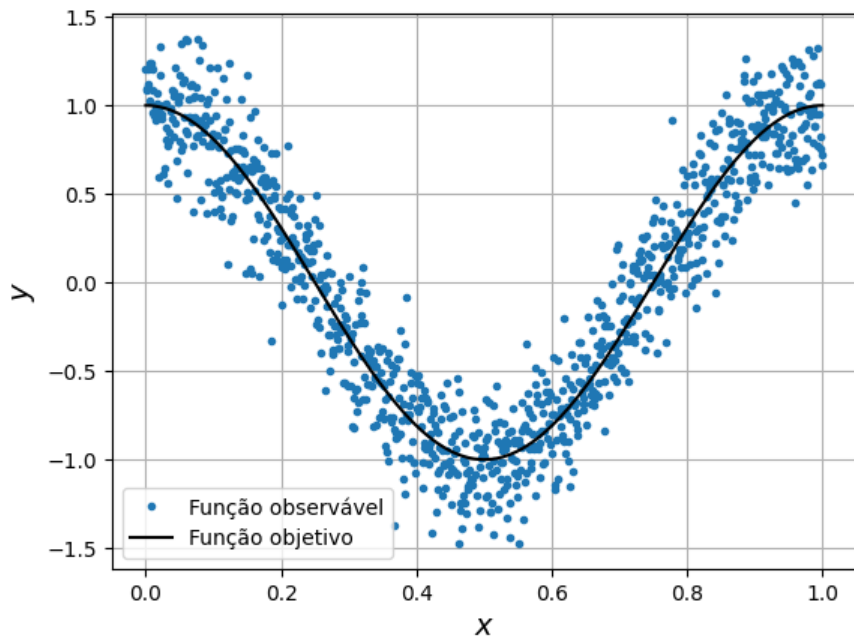
O que podemos fazer quando *hiperplanos não se ajustam bem aos dados*?

# Mapeamentos não lineares

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



# Regressão polinomial



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

# Regressão polinomial

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

$$y(x_1, x_2, x_3) = a_0 + a_1x_1 + a_2x_2 + a_3x_3 + a_4x_1x_3 + a_5x_1^3 + a_6x_1x_2x_3^2$$

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

# Regressão polinomial

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

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

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

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

# Regressão polinomial

- Só precisamos nos lembrar que o **vetor de atributos**,  $\mathbf{x}$ , e consequentemente, a **matriz de atributos**,  $\mathbf{X}$ , são **compostos pelos atributos originais e pelos atributos formados através de suas combinações**.

- Por exemplo, para a seguinte função hipótese polinomial
$$h(x_1(n)) = a_0 + a_1x_1(n) + a_2x_1^2(n) + \dots + a_Mx_1^M(n),$$

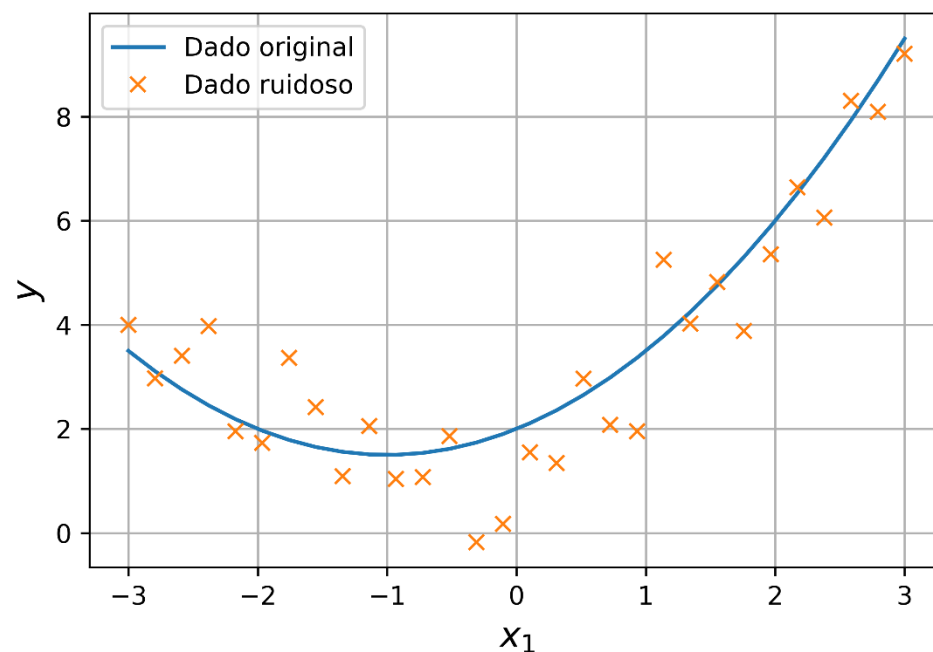
a **matriz de atributos polinomial**,  $\mathbf{X}$ , fica da seguinte forma

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

onde cada coluna contém um atributo (original ou combinação).

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

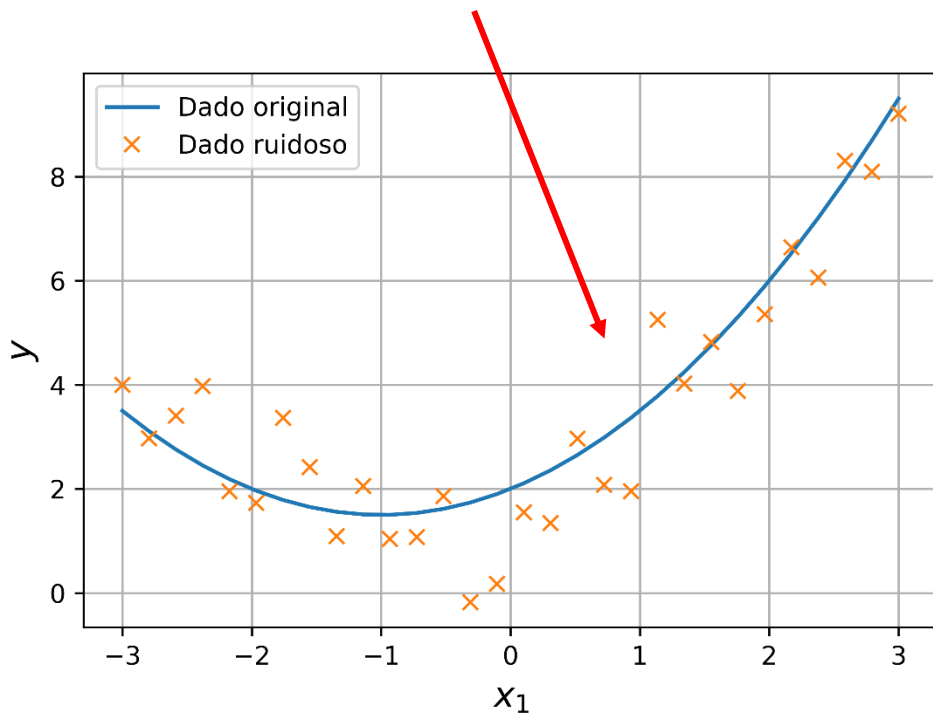
# Exemplo de regressão usando polinômio



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

# Exemplo de regressão usando polinômio

Função objetivo: polinômio de ordem 2.



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

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

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

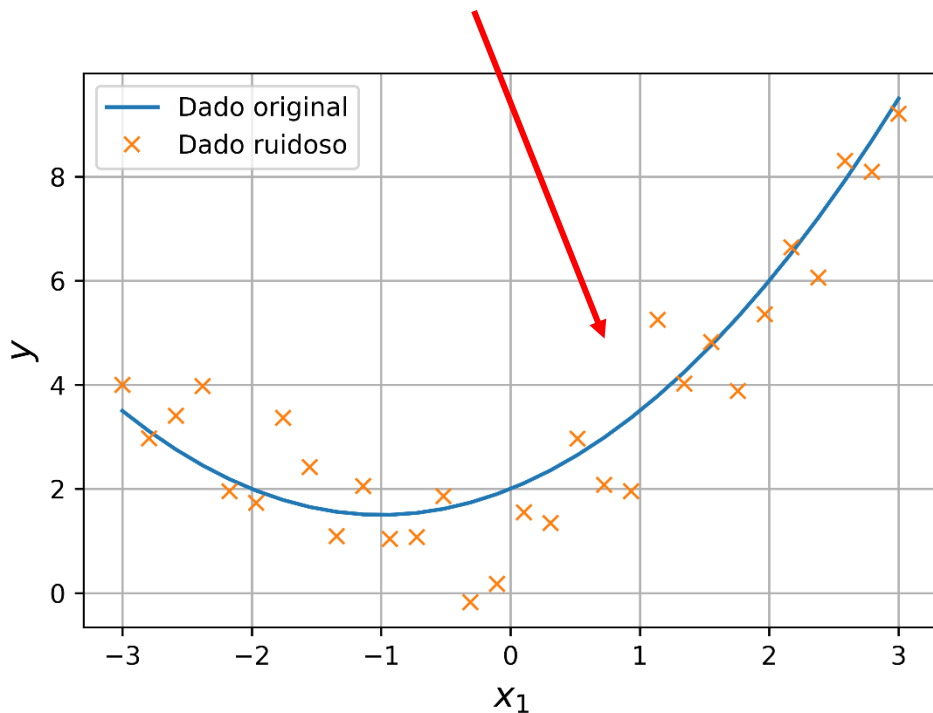
e adicionamos ruído Gaussiano branco,  $w(n)$

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

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

# Exemplo de regressão usando polinômio

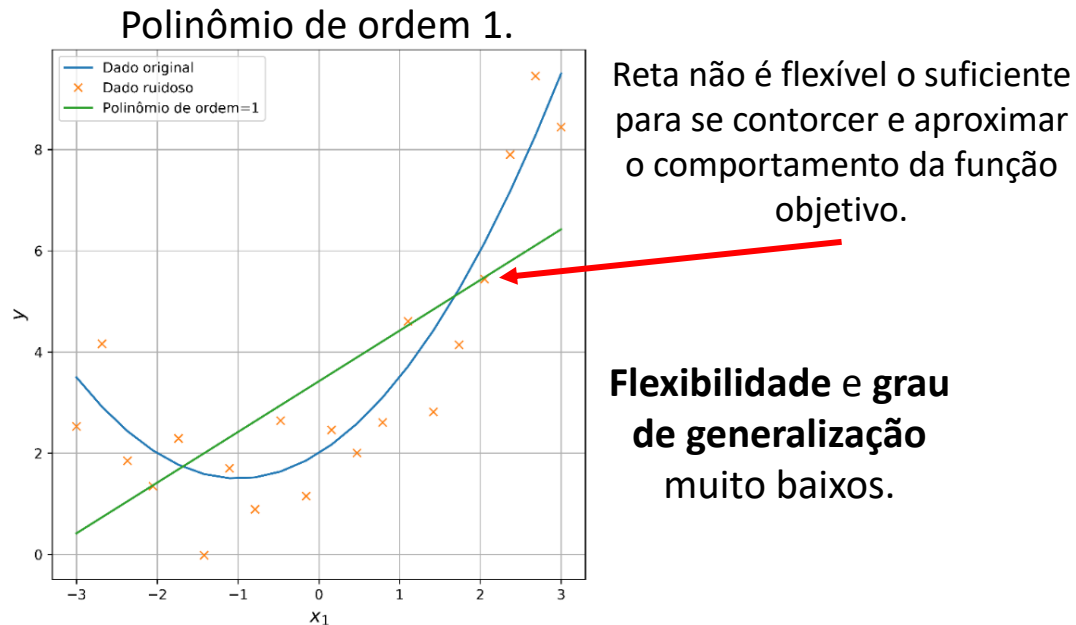
Função objetivo: polinômio de ordem 2.



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

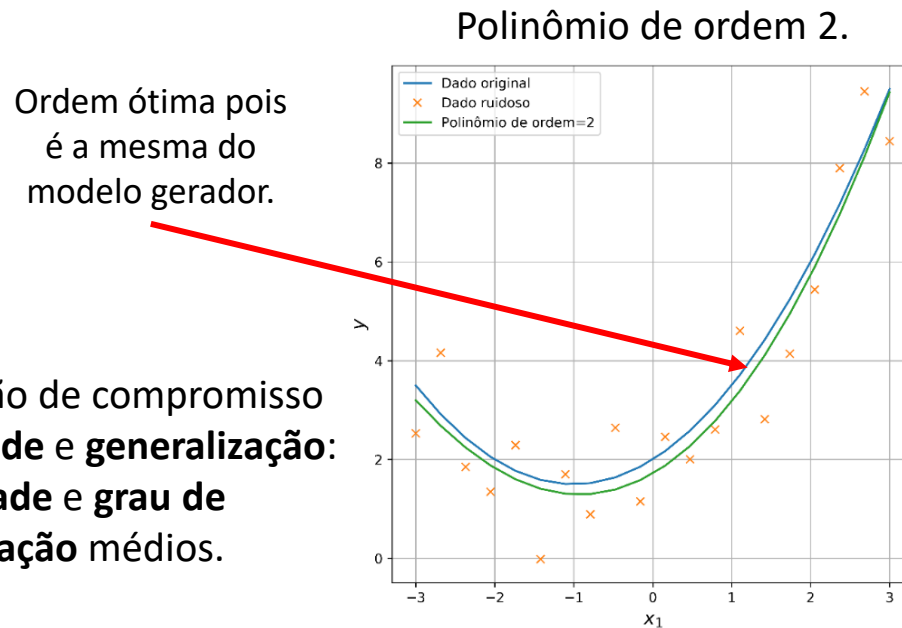
- Vamos usar uma **função hipótese polinomial** para **aproximar a função objetivo a partir dos dados ruidosos**.
- Porém, surge uma dúvida, **e se não soubéssemos a ordem por trás do modelo gerador, qual grau deveríamos utilizar?**

# Regressão polinomial: Qual ordem usar?



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

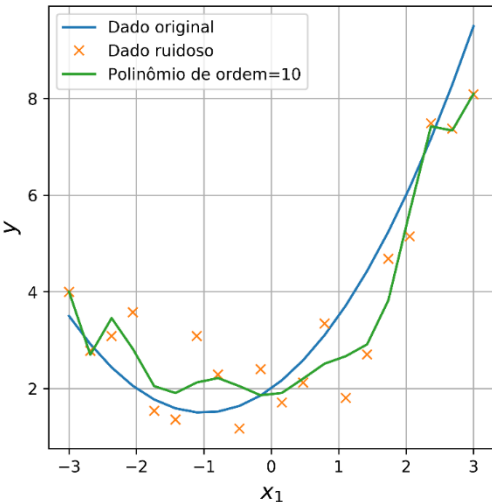
# Regressão polinomial: Qual ordem usar?



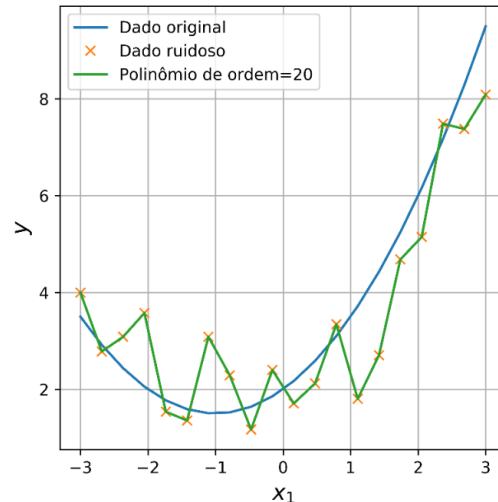
- Porém, como esperado, o polinômio de **ordem 2** produz a **melhor aproximação** da função objetivo, **errando pouco para exemplos dos conjuntos de treinamento e validação**.
  - Esse modelo encontra uma **relação de compromisso** entre **flexibilidade** e **grau de generalização**.
  - Essa aproximação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

# Regressão polinomial: Qual ordem usar?

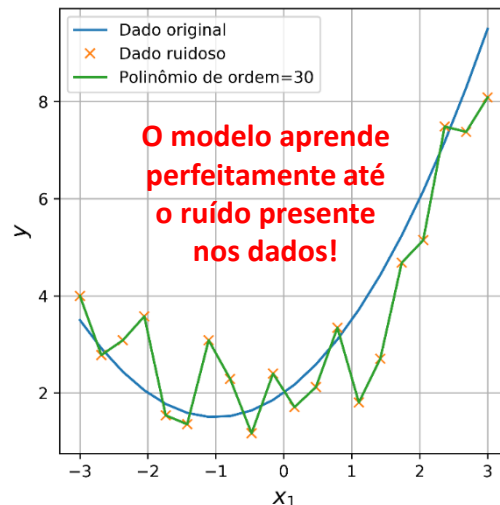
Polinômio de ordem 10.



Polinômio de ordem 20.



Polinômio de ordem 30.



- Polinômios com ordem *maior do que 2* tendem a produzir *aproximações perfeitas* dos exemplos disponíveis, i.e., o modelo *memoriza o conjunto de treinamento*.
- O *erro* no *conjunto de treinamento é muito baixo*.
- Porém, essa *aproximação se distancia bastante do modelo gerador*.
- Portanto, esses modelos apresentarão *erros significativamente maiores* quando forem apresentados a *exemplos de validação*.
- Efeito conhecido como *sobreajuste* ou *overfitting*: *flexibilidade* muito alta e *grau de generalização* muito baixo.

# Resumo sobre subajuste e sobreajuste

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

# Resumo sobre subajuste e sobreajuste


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

# Tarefas

- **Quiz:** “*T319 - Quiz - Regressão: Parte IV*” que se encontra no MS Teams.
- **Exercício Prático:** [Laboratório #5](#).
  - Pode ser acessado através do link acima (Google Colab) ou no GitHub.
  - Vídeo explicando o laboratório: Arquivos -> Recordings -> Laboratório #5
  - Se atentem aos prazos de entrega.
  - [Instruções para resolução e entrega dos laboratórios](#).
- **Entrega do projeto final: 28/11/2025**
  - Projeto final já se encontra no github.
  - Pode ser feito em grupos de no máximo 3 alunos.

Laboratório 5 

 Launch on Google Colab

 launch binder

Projeto Final 

 Launch on Google Colab

 launch binder

Obrigado!

**Anexo I:** O escalonamento altera o  
valor dos pesos originais

# Mudança dos pesos originais após a padronização

- Considerando a seguinte função hipótese

$$\hat{y}(n) = \hat{a}_1 x_1(n).$$

- Se padronizarmos o atributo  $x_1$ , teremos

$$x'_1(n) = \frac{x_1(n) - \mu_{x_1}}{\sigma_{x_1}}, \forall n,$$

onde  $\mu$  e  $\sigma$  são as estimativas da média e do desvio padrão, respectivamente, calculados ao longo de todas as amostras do vetor de atributos,  $\mathbf{x}_1$ .

# Mudança dos pesos originais após a padronização

- Isolando-se  $x_1(n)$  na equação da padronização, temos

$$x_1(n) = x'_1(n)\sigma_{x_1} + \mu_{x_1}.$$

- Na sequência, substituindo-se  $x_1(n)$  na função hipótese, tem-se

$$\hat{y}(n) = \hat{a}_1(x'_1(n)\sigma_{x_1} + \mu_{x_1}) = \hat{a}_1\sigma_{x_1}x'_1(n) + \hat{a}_1\mu_{x_1}.$$

- Perceba que na equação acima há o surgimento de um termo de bias,  $\hat{a}_1\mu_{x_1}$ , além da alteração do peso original para  $\hat{a}_1\sigma_{x_1}$ .

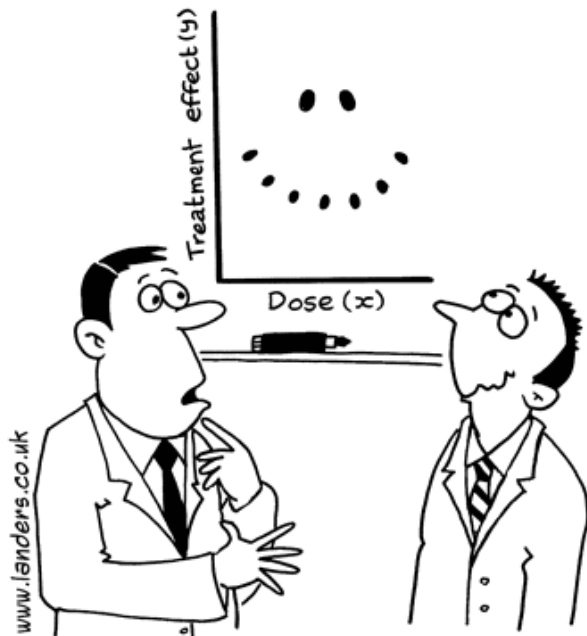
# Mudança dos pesos originais após a padronização

- Assim, podemos reescrever a equação acima como

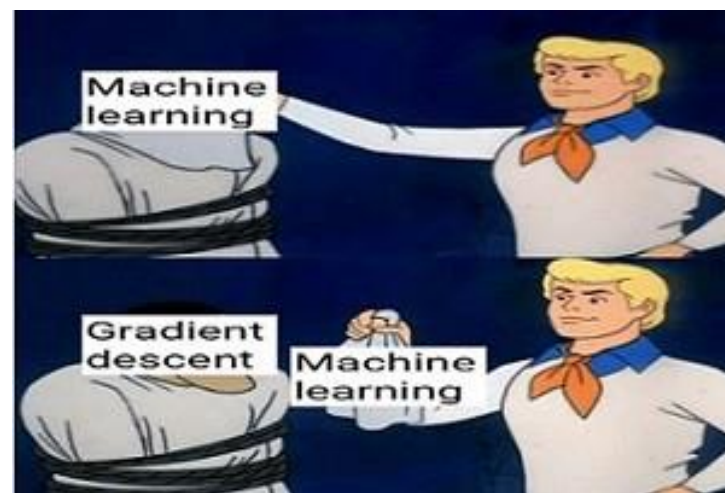
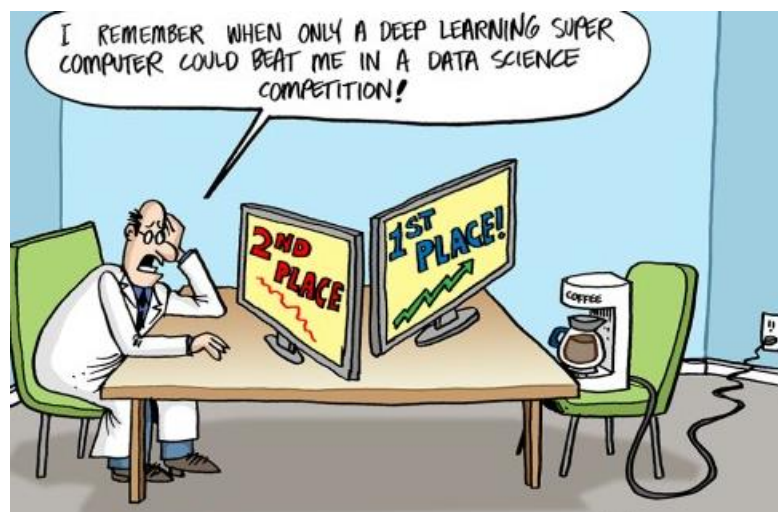
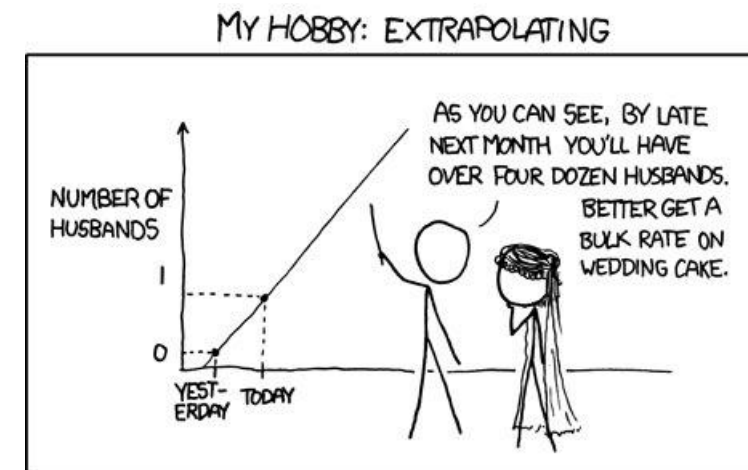
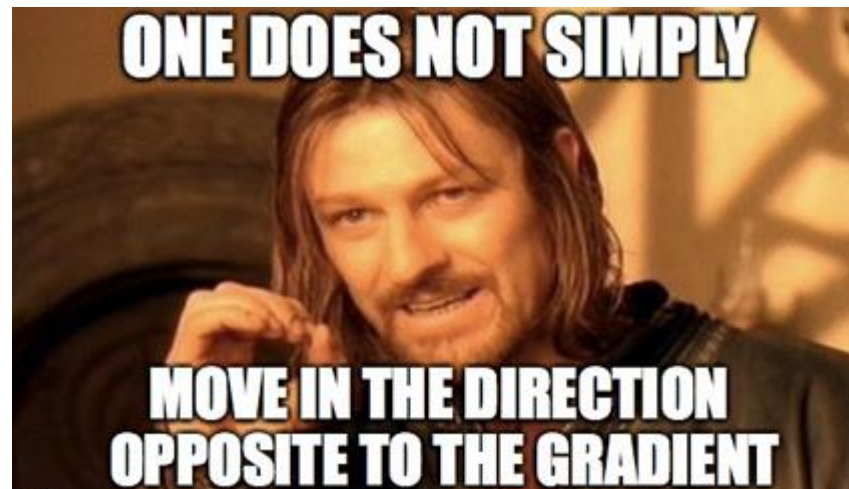
$$\hat{y}(n) = \hat{a}'_0 + \hat{a}'_1 x'_1(n),$$

onde  $\hat{a}'_0 = \hat{a}_1 \mu_{x_1}$  e  $\hat{a}'_1 = \hat{a}_1 \sigma_{x_1}$ .

- Note que a padronização de  $x_1(n)$  fez com que  $\hat{a}_1$  fosse modificado de forma que a função hipótese ainda produza em sua saídas previsões condizentes com os valores esperados,  $y(n)$ .
- Ou seja, mesmo com a padronização dos atributos, a função hipótese ainda fará previsões alinhadas aos valores dos rótulos,  $y(n)$ .
- O mesmo procedimento pode ser diretamente aplicado à normalização e também resultará em mudança dos pesos originais.



"It's a non-linear pattern with outliers.....but for some reason I'm very happy with the data."



Figuras

