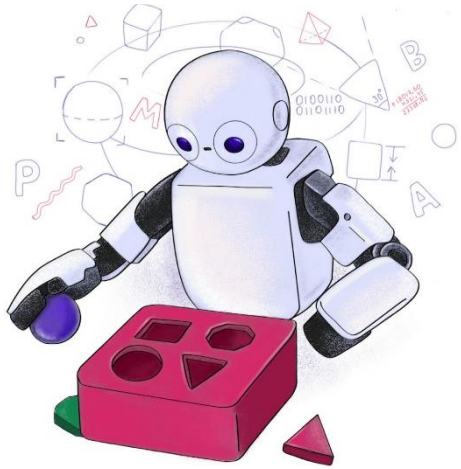
T319 - Introdução ao Aprendizado de Máquina: *Regressão Linear (Parte IV)*



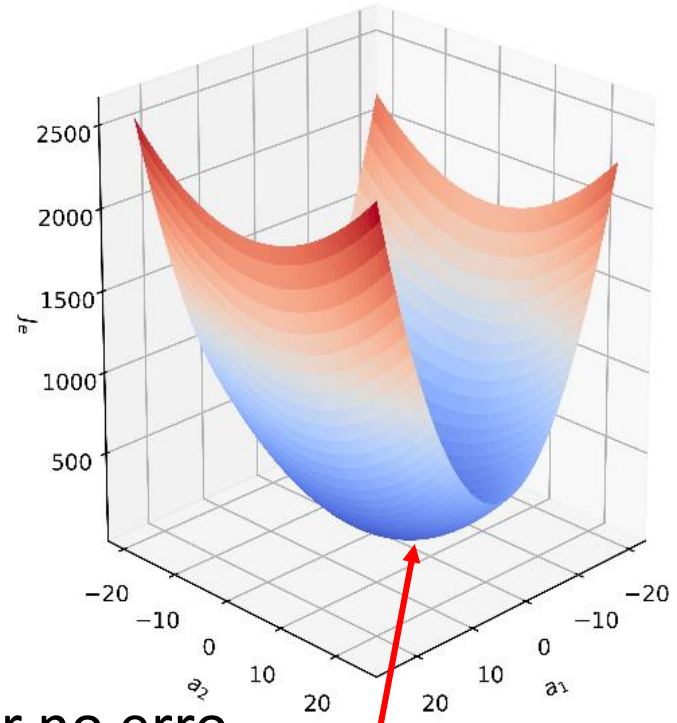
Inatel

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

Recapitulando

- Vimos que ***a escolha do passo de aprendizagem influencia muito no processo aprendizagem*** do gradiente descendente.
  - Valores pequenos fazem com que o algoritmo tenha convergência muito lenta.
  - Valores grandes fazem com que o algoritmo divirja.
- Gráfico do erro em função das iterações nos ajuda a depurar o algoritmo.
- Além do ajuste manual, quando usamos GDE ou GD em mini-batches, precisamos reduzir o valor do passo de aprendizagem ao longo das iterações para "*forçar*" a convergência do GD.
- Hoje, veremos
  - um tipo de ***pré-processamento*** bastante importante para algoritmos de ML que usam métricas de distância como função de erro.
  - como aproximar dados que não são lineares, ou seja, que não podem ser aproximados por uma simples reta.

Escalonamento de Atributos



- Dada a seguinte equação hipótese, $h(\boldsymbol{x})$

$$\hat{y}(n) = h\big(\boldsymbol{x}(n)\big) = \widehat{a_1}x_1(n) + \widehat{a_2}x_2(n).$$

- A função de erro é dada por

$$J_e(\boldsymbol{a}) = \frac{1}{N}\sum_{n=0}^{N-1}\Big[y_{\text{noisy}}(n) - \big(\widehat{a_1}x_1(n) + \widehat{a_2}x_2(n)\big)\Big]^2.$$

- Caso $x_1(n) \gg x_2(n), \forall i$, então $x_1(n)$ tem uma influência maior no erro resultante, o que pode ser expresso de forma aproximada como

$$J_e(\boldsymbol{a}) \approx \frac{1}{N}\sum_{n=0}^{N-1}\Big[y_{\text{noisy}}(n) - \widehat{a_1}x_1(n)\Big]^2.$$

- Portanto, o erro entre y_{noisy} e $h\big(\boldsymbol{x}(n)\big)$ será dominado pelo atributo $x_1(n)$ e, portanto, pequenas variações de $\widehat{a_1}$ fazem com que o erro varie rapidamente.
- A diferença entre as magnitudes dos atributos afeta o desempenho de algoritmos de ML que usam métricas de distância como função de erro.
  - As diferenças entre as magnitudes dos atributos faz com que as superfícies de erro tenham formato de vale (‘U’ ou ‘V’), ***dificultando a convergência de algoritmos iterativos, como o gradiente descendente (todas as versões)***.

Escalonamento de Atributos

- O que pode ser feito?
- Para evitar esse problema, o intervalo de variação de todos os ***atributos*** deve ser ***escalonado*** para que cada ***atributo*** contribua com o mesmo ***peso*** para o cálculo do erro.
- As duas formas mais comuns de escalonamento são:
  - **Normalização Mín-Max**

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

  - **Padronização**

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

- ***Normalização Mín-Max*** faz com que os atributos variem entre 0 e 1.
- ***Padronização*** faz com que os atributos tenham média zero e desvio padrão unitário. Observe que, neste caso, os valores não ficam restritos a um intervalo específico.
- Vantagens do escalonamento
  - Ajuda a acelerar a convergência do ***gradiente descendente*** pois deixa as curvas de nível da superfície de erro mais circulares.
  - Possibilita comparar mais facilmente o peso/influência de cada ***atributo*** no modelo.

Escalonamento de Atributos

Modelo gerador:

$$y(n) = a_1 x_1(n) + a_2 x_2(n),$$

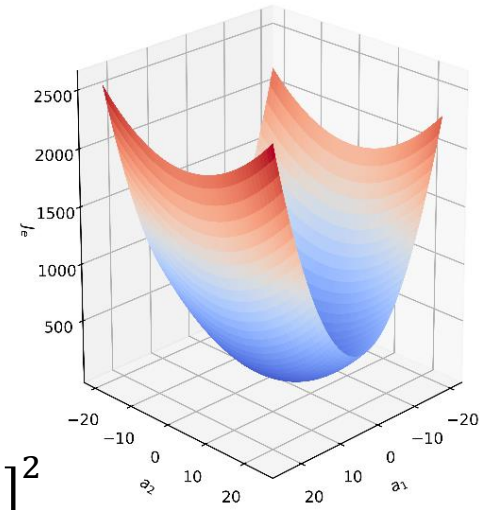
onde $a_1 = 1, a_2 = 1$.

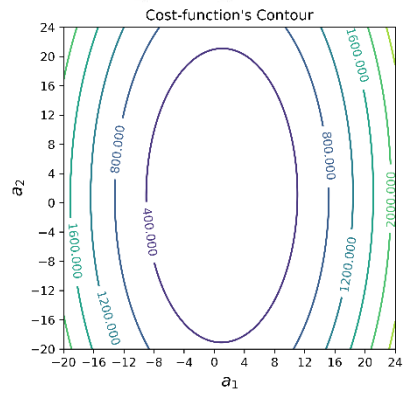
Para plotar a superfície de erro usamos:

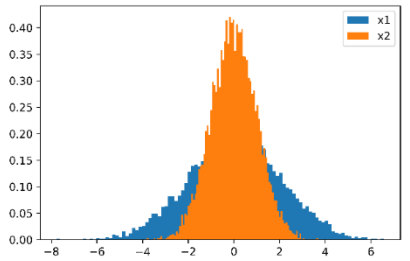
$$J_e(\boldsymbol{a}) = \frac{1}{N}\sum_{n=0}^{N-1}\left[y_{\text{noisy}}(n) - (\widehat{a_1}x_1(n) + \widehat{a_2}x_2(n))\right]^2$$

- $x_1 \gg x_2$: erro varia mais rapidamente com variações de $\widehat{a_1}$, resultando num vale.
- $x_2 \gg x_1$: erro varia mais rapidamente com variações de $\widehat{a_2}$, resultando também em um vale.
- Quando x_1 e x_2 têm intervalos semelhantes, então, a variação tanto de $\widehat{a_1}$ quanto de $\widehat{a_2}$ tem ***peso*** semelhante na variação do erro (tigela).

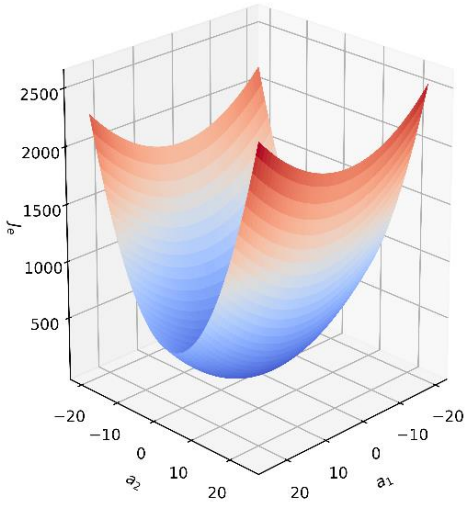
Após padronização

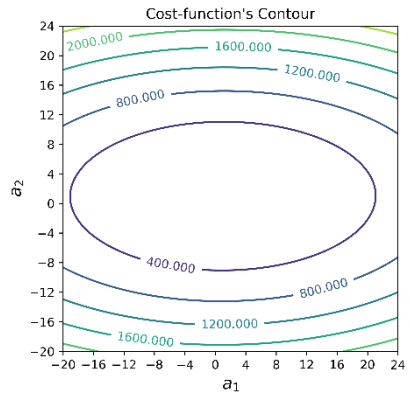


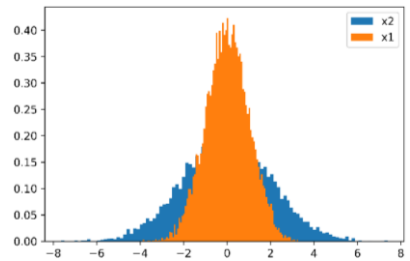




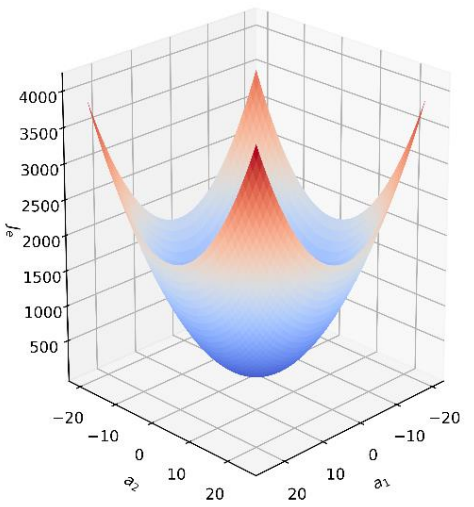
$x_1 = 2 * \text{randn}(\text{M}, 1)$
$x_2 = \text{randn}(\text{M}, 1)$

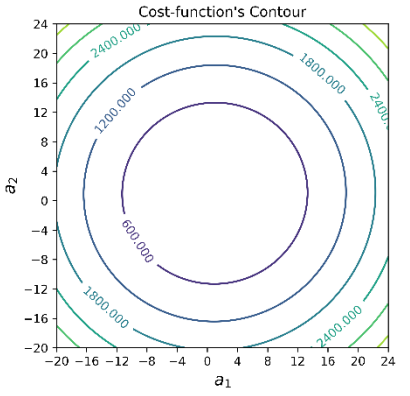


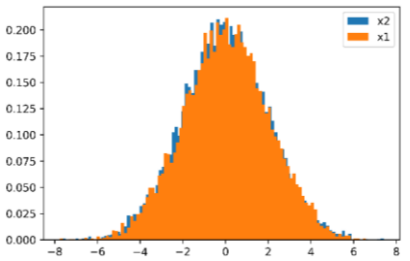




$x_1 = \text{randn}(\text{M}, 1)$
$x_2 = 2 * \text{randn}(\text{M}, 1)$





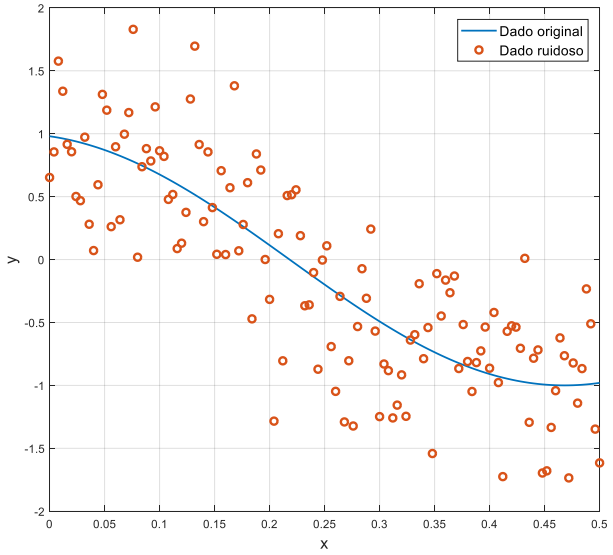


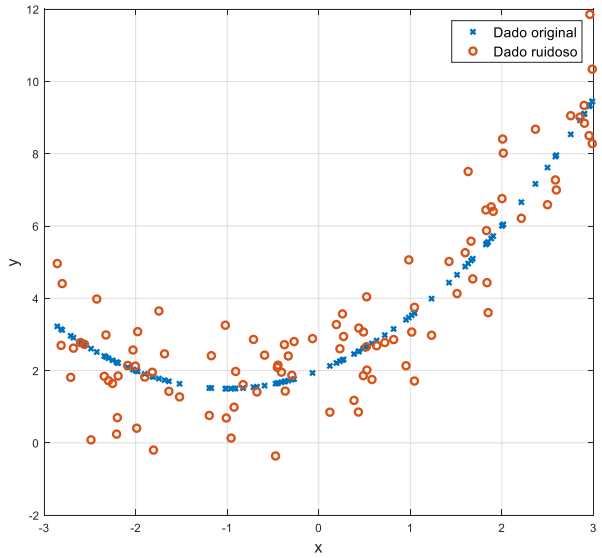
$x_1 = \text{randn}(\text{M}, 1)$
$x_2 = \text{randn}(\text{M}, 1)$

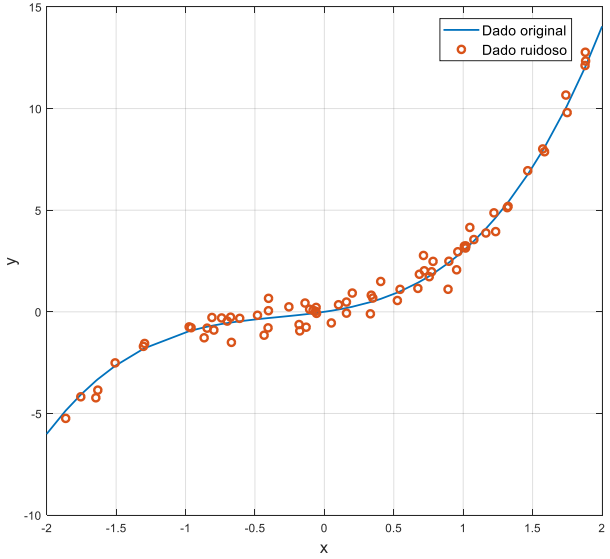
[Exemplo: escalonamento_de_atributos.ipynb]

Regressão Polinomial: Motivação

- Até agora, usamos funções hipóteses com formato de hiperplanos, e.g., retas ou planos, mas e se os dados tiverem um formato mais complexo do que uma simples linha reta ou plano?
- Como encontraríamos um modelo que aproxime as funções abaixo?
- Uma reta claramente não seria uma boa escolha.
  - Uma reta não capturaria o comportamento das funções abaixo, pois ela não tem complexidade (i.e., graus de liberdade) o suficiente para isso.







Regressão Polinomial

- Através do teorema de ***Weierstrass***, sabemos que dados deste tipo podem ser aproximados através de ***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***.
- Portanto, podemos aproximar dados de qualquer formato com polinômios:

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

- Por simplicidade, para nossas análises, nós vamos considerar ***funções hipóteses polinomiais 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 M é a ordem do polinômio.
- Todos resultados encontrados anteriormente (equação normal, vetor gradiente para o algoritmo do gradiente descendente, escalonamento) são diretamente estendidos para ***funções hipótese polinomiais***.
- Porém, precisamos encontrar a ordem do polinômio que melhor aproxime os dados.

Regressão Polinomial: Exemplo

- Geramos 30 exemplos do seguinte ***mapeamento verdadeiro (i.e., função objetivo)***:

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

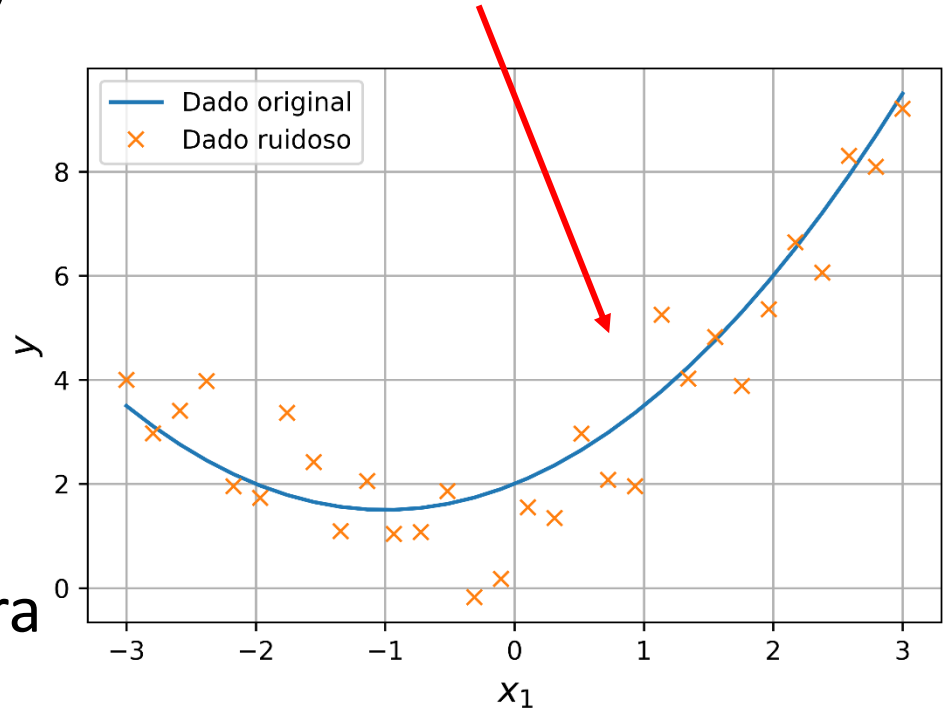
e adicionamos ruído Gaussiano branco, $w(n)$

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

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

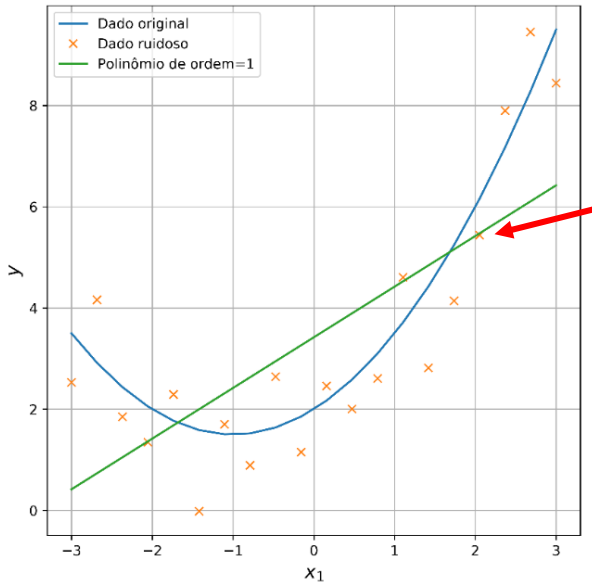
- Vamos usar uma ***função hipótese polinomial*** para aproximar a função objetivo.
- Porém, surge uma dúvida, ***e se não soubéssemos a ordem por trás do modelo gerador, qual ordem deveríamos utilizar***?

Função objetivo: polinômio de ordem 2.



Regressão Polinomial: Qual ordem usar?

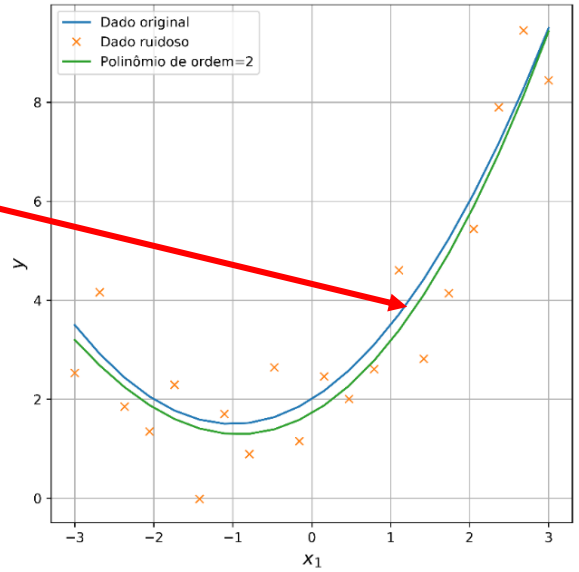
Polinômio de ordem 1.



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

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

Polinômio de ordem 2.



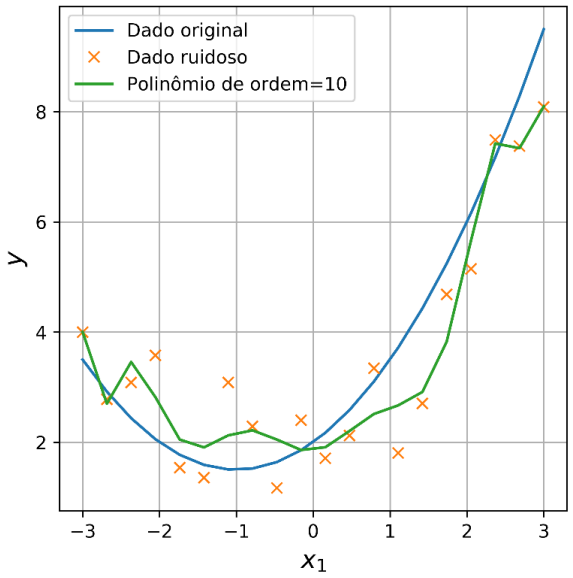
Ordem ótima pois é a mesma do modelo gerador.

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

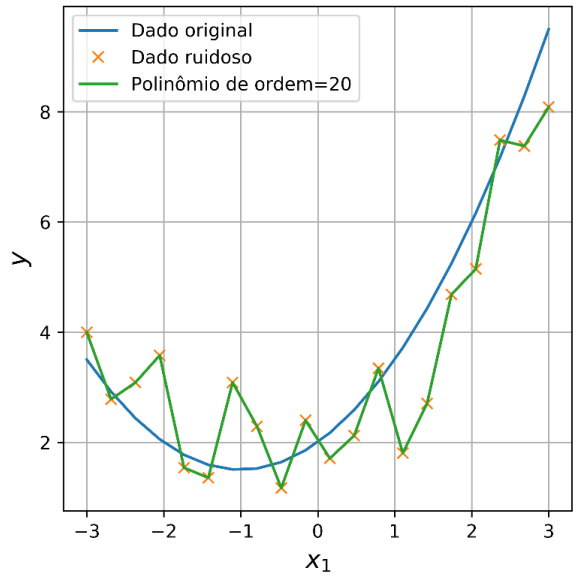
- Polinômio de ordem 1 não tem flexibilidade o suficiente para aproximar bem os dados.
- O modelo erra muito tanto para predição dos exemplos de treinamento quanto para exemplos de validação (ou seja, exemplos não vistos durante o treinamento).
- Efeito conhecido como ***subajuste*** ou ***underfitting***: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

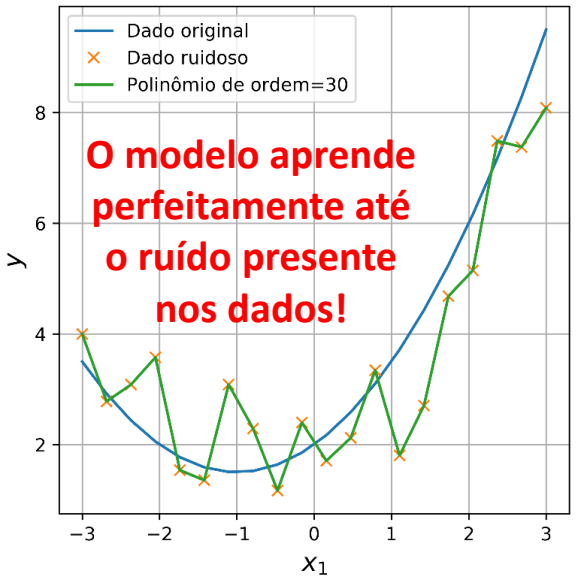
Polinômio de ordem 10.



Polinômio de ordem 20.



Polinômio de ordem 30.



- Polinômios com ordem > 2 tendem a produzir ***aproximações perfeitas*** dos exemplos disponíveis, ou seja, o modelo acaba ***memorizando*** os exemplos de treinamento.
- 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.

Exemplo: polynomial_regression.ipynb

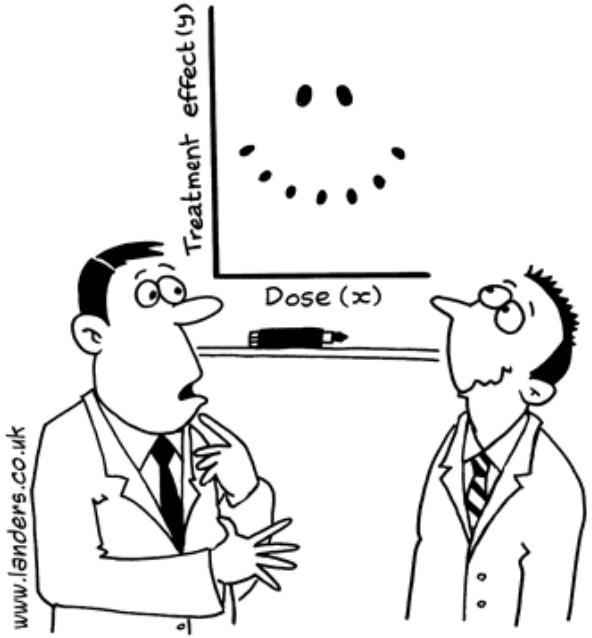
Subajuste e sobreajuste: Resumo

- **Subajuste**: situação em que o modelo falha em aproximar o ***mapeamento verdadeiro***.
  - Ocorre devido ao baixo grau de flexibilidade do modelo.
  - 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.
- **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 uma relação de compromisso entre **flexibilidade** e **generalização** do modelo: **flexibilidade** e **grau de generalização** médios.

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 -> Material de Aula -> Laboratório #5
  - Se atentem aos prazos de entrega.
  - Instruções para resolução e entrega dos laboratórios.
  - **Laboratórios podem ser resolvidos em grupo, mas as entregas devem ser individuais.**

Obrigado!



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