

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



Inatel

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

Motivação

- **Exemplo 1:** Estimar o preço de casas.
- **Exemplo 2:** Estimar as vendas de sorvete.

500 m²



R\$ 1.000.000,00

70 m²



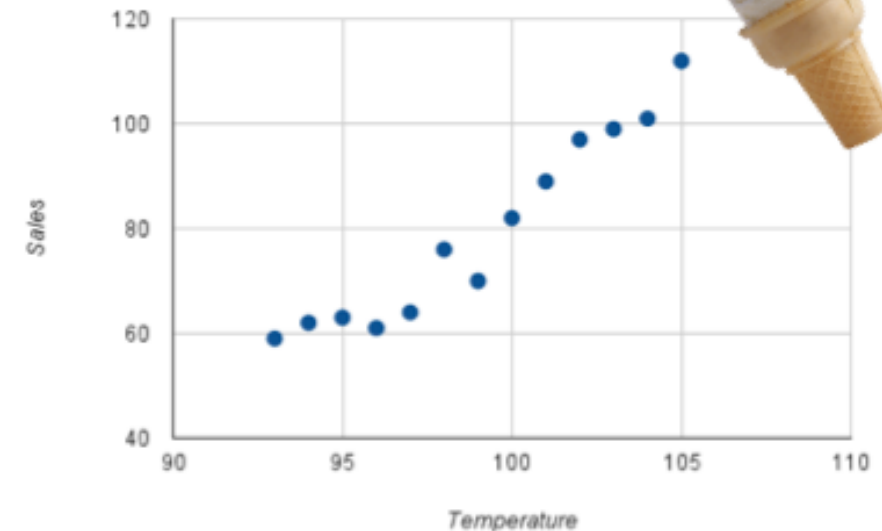
R\$ 200.000,00

200 m²



???

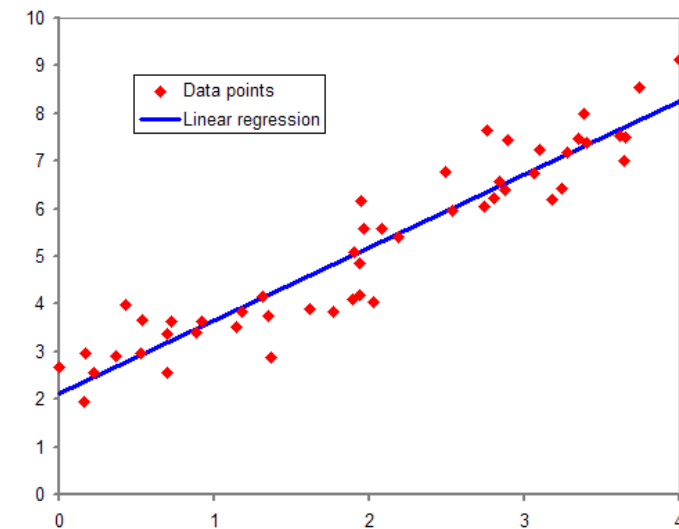
Ice Cream Sales



- Podemos encontrar uma relação matemática entre a área, localização, n° de quartos de uma casa e seu valor?
- Ou entre a temperatura e a quantidade de sorvetes vendidos?

Regressão Linear

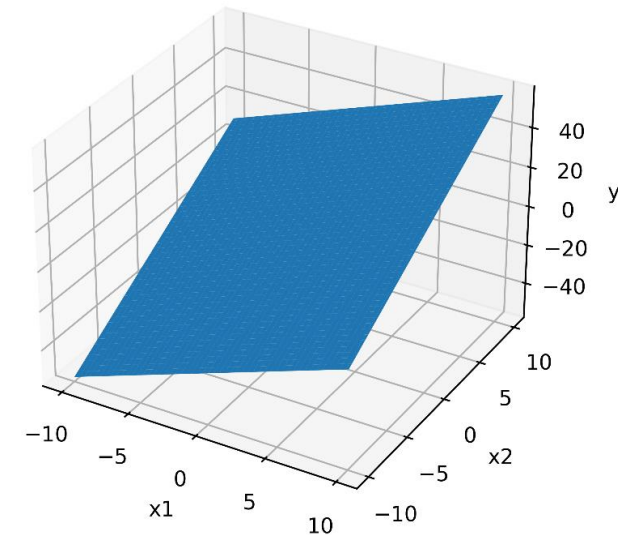
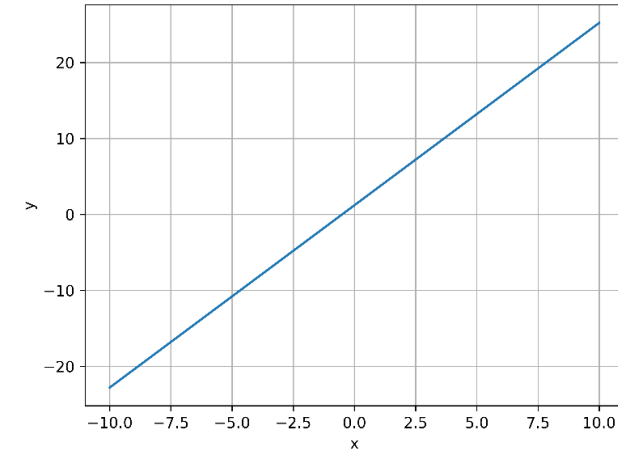
- Um dos mais, se não o mais, conhecido algoritmo de aprendizado de máquina.
- Vai nos dar vários *insights* (i.e., intuições) importantes para o entendimento de outros algoritmos mais complexos, como, por exemplo, classificadores e redes neurais.
- **Objetivo:** encontrar uma função, $h(\mathbf{x})$, que mapeie, de forma ótima, os atributos de entrada \mathbf{x} em uma variável de saída \hat{y} : $\hat{y} = h(\mathbf{x})$, de tal forma que $h(\mathbf{x})$ seja uma boa aproximação da **função verdadeira**, mas muitas vezes desconhecida, chamada de **função objetivo**, $f(\mathbf{x})$.
- Regressão também é conhecida como **aproximação de funções**.
- Como faríamos para encontrar uma função, $h(\mathbf{x})$, que **aproxime** $f(\mathbf{x})$ de forma ótima?



Temos \mathbf{x} (atributos) e \mathbf{y} (rótulos) e queremos encontrar $\hat{y} = h(\mathbf{x})$.
Que tipo de aprendizado?

Regressão Linear

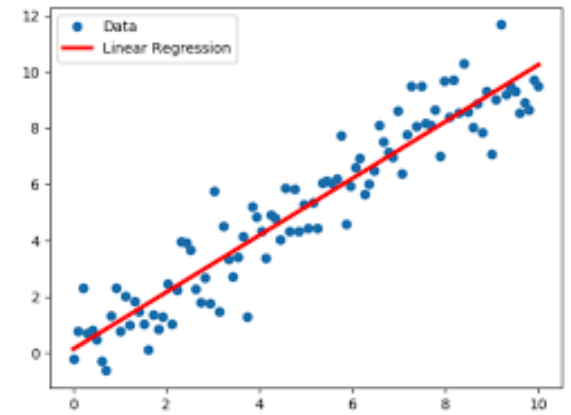
- **Qual forma deve ter a função $h(\mathbf{x})$?** Os modelos mais simples são:
 - Com apenas um atributo, x_1 , $h(\mathbf{x})$ é uma reta, $h(\mathbf{x}) = a_0 + a_1 x_1$.
 - Com dois atributos, x_1 e x_2 , $h(\mathbf{x})$ é uma superfície 2D (ou seja, um plano), $h(\mathbf{x}) = a_0 + a_1 x_1 + a_2 x_2$.
 - E assim por diante.
- **Modelo geral:** Equação de um Hiperplano
$$h(\mathbf{x}) = a_0 + a_1 x_1 + \dots + a_K x_K = a_0 + \sum_{i=1}^K a_i x_i.$$
- Existem outros modelos, os quais veremos mais adiante.
- Na literatura, a função $h(\mathbf{x})$, é chamada de **função hipótese**, pois é uma das possíveis soluções encontradas no **espaço de hipóteses**, H , para aproximar $f(\mathbf{x})$.
- **Espaço de hipóteses:** é conjunto de todas as possíveis **funções hipótese**.
 - Hiperplano formado por todos possíveis valores dos parâmetros, $a_k, \forall k$.



Regressão Linear

- Agora que sabemos a forma de $h(\mathbf{x})$, podemos refinar o objetivo da regressão um pouco mais.
- **Objetivo:** Encontrar os **parâmetros**, também chamados de **pesos**, a_0, a_1, \dots, a_k de tal forma que $h(\mathbf{x})$ seja uma aproximação ótima de $f(\mathbf{x})$.
- **Aprendizado supervisionado:** atributos/exemplos (i.e., \mathbf{x}) mais rótulos/objetivos (i.e., y).
- A **regressão** é chamada de **linear** porque a variável de saída, y , é modelada como uma **combinação linear** dos atributos, \mathbf{x} .
 - **OBS.: Linear**, nesse contexto, significa “*linear com relação aos pesos*” e não com relação aos atributos, i.e., \mathbf{x} . Desta forma, os seguintes modelos também são lineares com relação aos pesos:
 - $y = a_0 + a_1 \log x_1 + a_2 \cos x_2$
 - $y = a_0 + a_1 e^{x_1}$
 - $y = a_0 + a_1 x_1^2$
- Exemplo de modelo não-linear: $y = \frac{a_0 x_1}{a_1 + x_1}$.

Definição do Problema



O problema de **regressão** pode ser enunciado da seguinte forma:

- **Dados disponíveis:**

- Conjunto de N observações (conjunto de pares/exemplos de treinamento) : $\{\mathbf{x}(i), y(i)\}$, $i = 0, \dots, N - 1$, onde
 - $\mathbf{x}(i) \in \mathbb{R}^K$: i -ésimo vetor de entrada com dimensão K , ou sejam K atributos (ou features).
 - $y(i) \in \mathbb{R}$: i -ésimo valor esperado de saída referente ao vetor de entrada $\mathbf{x}(i)$.

- **Modelo:**

$$\hat{y}(i) = h(\mathbf{x}(i)) = a_0 + a_1 x_1(i) + \dots + a_K x_K(i) = \mathbf{a}^T \Phi(i),$$

onde $\mathbf{a} = [a_0, \dots, a_K]^T$ e $\Phi(i) = [1, x_1(i), \dots, x_K(i)]^T$.

- \mathbf{a} é o vetor $(K + 1 \times 1)$ contendo os parâmetros/pesos que definem a **função hipótese**, ou seja, o mapeamento $h: \mathbf{x}(i) \rightarrow \hat{y}(i)$ e $\Phi(i)$ é um vetor $(K + 1 \times 1)$ contendo os i -ésimos valores dos atributos.
- a_0 é o **coeficiente linear**, ou seja, é o valor de $h(\mathbf{x})$ para o ponto do hiperplano que intercepta o eixo y , a_0 é conhecido também como **intercept** ou **bias**.
- Como a_0 não tem um **atributo** relacionado a ele, para facilitar o modelamento matemático, criamos um atributo falso, x_0 , com valor constante sempre igual a 1, i.e., $x_0 = 1$.
- **Objetivo do modelo:** encontrar o vetor de pesos \mathbf{a} que minimize o **erro**, dado por uma **função de erro**, $J_e(\mathbf{a})$, entre a aproximação $\hat{y}(i)$ e o valor desejado $y(i)$ para todo i .

$$\min_{\mathbf{a}} J_e(\mathbf{a})$$

Ou seja, **o treinamento do modelo envolve a minimização de uma função de erro.**

- Portanto, precisamos definir uma **função de erro**.

Função de Erro

- **Função de erro:** existem várias possibilidades para se definir a **função de erro** a ser minimizada, porém, geralmente, utiliza-se a medida do **erro quadrático médio**

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

que nada mais é do que a **média aritmética do quadrado dos erros**.

- Nós veremos mais adiante a razão pela qual o **erro quadrático médio** é utilizado.
- A **função de erro** pode ser reescrita em forma matricial como

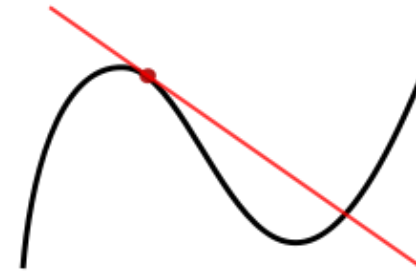
$$J_e(\mathbf{a}) = \frac{1}{N} \|\mathbf{y} - \Phi \mathbf{a}\|^2,$$

onde $\mathbf{y} = [y(0), \dots, y(N-1)]^T$ é um vetor $(N \times 1)$, $\Phi = [\Phi(0), \dots, \Phi(N-1)]^T$ é uma matriz $(N \times K+1)$ e N é o número de amostras, exemplos ou observações.

- Então, para encontrar o vetor de parâmetros \mathbf{a} devemos minimizar

$$\min_{\mathbf{a} \in \mathbb{R}^{K+1}} \|\mathbf{y} - \Phi \mathbf{a}\|^2.$$

Minimizando a Função de Erro



A linha vermelha é tangencial à curva no ponto marcado pelo ponto vermelho. Sua inclinação é a derivada naquele ponto.

Como encontramos o mínimo da função de erro em relação aos pesos?

- Da disciplina de cálculo, sabemos que derivando $\|\mathbf{y} - \Phi \mathbf{a}\|^2$ com relação a \mathbf{a} e igualando a 0 nós encontramos o **ponto** onde a inclinação de uma reta tangente à **função de erro** é nula:

$$\frac{\partial \|\mathbf{y} - \Phi \mathbf{a}\|^2}{\partial \mathbf{a}} = 2\mathbf{a}^T \Phi^T \Phi - 2\mathbf{y}^T \Phi = 0,$$

porém, esse ponto pode ser tanto um mínimo quanto um máximo da **função de erro**, pois em ambos os pontos a inclinação da tangente é nula.

- **Então, como sabemos se o ponto encontrado é um mínimo ou um máximo?**

Se a inclinação da tangente é nula e a **derivada de segunda ordem** for positiva, então o ponto nos dá o mínimo da função,

$$\frac{\partial^2 \|\mathbf{e}\|^2}{\partial^2 \mathbf{a}} = 2\Phi^T \Phi.$$

Se a matriz Φ tiver **posto** igual a $K + 1$, então a matriz $\Phi^T \Phi$ é **positiva semi-definida** e, portanto, o ponto encontrado acima é realmente o ponto de mínimo da **função de erro**.

- **Posto de uma matriz:** corresponde ao número de linhas ou colunas linearmente independentes da matriz.
- Uma matriz quadrada A é **positiva semi-definida** se $\mathbf{x}^* A \mathbf{x} \geq 0, \forall \mathbf{x} \neq 0$.

Minimizando a Função de Erro

- Portanto, voltando à equação da derivada parcial de primeira ordem igual a 0, temos

$$\mathbf{a}^T \Phi^T \Phi = \mathbf{y}^T \Phi.$$

- Após aplicarmos o transposto a ambos os lados e isolando \mathbf{a} temos

$$\mathbf{a} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y}.$$

- Essa equação é conhecida como a **equação normal** e nos dá a **solução ótima** em relação a minimização do **erro quadrático médio** para esse sistema de equações.

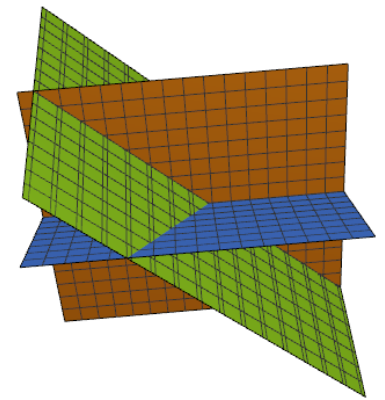
Observações:

1. O método encontra uma **solução única** se e somente se a matriz quadrada, $\Phi^T \Phi$, for **invertível** (se ela for **não-singular**), ou seja, com **posto** igual a $K + 1$.
2. O método só funciona para sistemas **determinados** ou **sobredeterminados**, ou seja, quando o número de equações (i.e., pares \mathbf{x} e \mathbf{y}) é igual ou maior do que o número de incógnitas (i.e., pesos), ou seja, $N \geq K + 1$.
3. Para sistemas **subdeterminados**, ou seja, que têm menos equações do que incógnitas, a matriz $\Phi^T \Phi$ tem **posto** menor do que $K + 1$ e portanto é **singular** (ou seja, a matriz não tem uma inversa). Neste caso, não existe solução ou ela não é única.

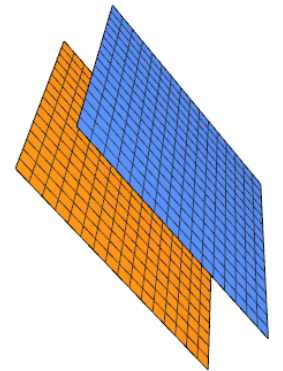
Causas para singularidade de $\Phi^T \Phi$

Algumas causas para a singularidade são:

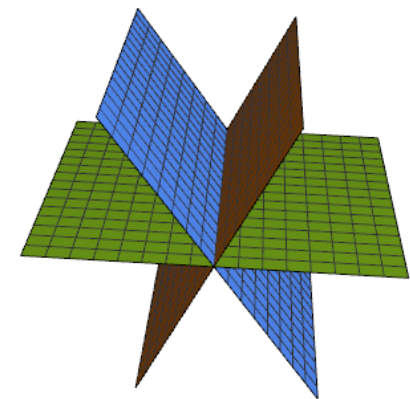
- Atributos, x , redundantes: pelo menos um dos atributos x_1, x_2, \dots, x_K é linearmente dependente.
 - **Ex.:** $x_1 = \text{área em } m^2$ e $x_2 = \text{área em } pés^2 \Rightarrow 1 m^2 = 10.7639 pés^2$.
 - **Solução:** remover um dos atributos que são linearmente dependentes.
 - **Técnicas:** seleção de atributos (LASSO) e redução de dimensionalidade (PCA).
- Número de atributos (x_1, x_2, \dots, x_K) maior do que o de observações ($y(0), y(1), \dots, y(N - 1)$), i.e., $K > N$ (**sistema subdeterminado**):
 - **Sistema subdeterminado:** sistema não tem solução ou existe um número infinito delas (Ex.: quantos planos 2D podem passar por uma reta, 1D?).
 - **Solução:** coletar mais exemplos, remover atributos ou replicar exemplos adicionando ruído à eles.



Sistema com uma única solução



Sistema sem solução



Sistema com infinitas soluções

Regressão Linear em Python

```
# Import all the necessary libraries.
import numpy as np

# Generate input/output (features/labels) values.
N = 100 # Number of observations.

x = 2 * np.random.rand(N, 1)
y = 4 + 3 * x
y_noisy = y + np.random.randn(N, 1)

# Solve by applying the least-Squares method.
# We use the inv() function from NumPy's Linear Algebra module (np.linalg) to
compute the inverse of a matrix.
# We use dot() method for matrix multiplication.
X_b = np.c_[np.ones((N, 1)), x] # add x0 = 1 to each instance
a_optimum = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y_noisy)

# Print best solution.
print('a0: %1.4f' % (a_optimum[0][0]))
print('a1: %1.4f' % (a_optimum[1][0]))

a0: 4.0763
a1: 2.9413
```

```
# The equivalent solution using the Scikit-Learn library is given below
# Import the linear regression module from the library.
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression() # instantiate it.
lin_reg.fit(x, y_noisy)

# Value that crosses the y-axis when all features are equal to 0.
print('a0: %1.4f' % (lin_reg.intercept_[0]))
# parameters associated with the features.
print('a1: %1.4f' % (lin_reg.coef_[0][0]))

a0: 4.0763
a1: 2.9413
```



<https://scikit-learn.org>

[Exemplo: normal equation example1.ipynb](#)

- Percebam que com apenas 100 exemplos, os valores obtidos são bem próximos dos exatos, porém, o ruído torna impossível recuperar os parâmetros exatos do modelo gerador.
- Se aumentarmos o número de exemplos conseguimos melhorar a estimação, porém, o ruído limitará essa melhoria.
- O argumento x passado para o método **fit** da classe **LinearRegression** é uma matriz com $N \times K$. Porém, lembrem-se que se existe um peso a_0 , e portanto x deveria ter dimensão $N \times K+1$, entretanto, por padrão, a classe **LinearRegression** já faz isso pra vocês automaticamente. Caso sua função hipótese não considere o peso a_0 , então, durante a instanciação da classe vocês devem configurar o parâmetro **fit_intercept=False**.

Superfície de Erro

- E se plotarmos a função de erro, $J_e(\mathbf{a})$? Que forma vocês acham que ela terá?

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

- $J_e(\mathbf{a})$ faz o mapeamento entre cada possível valor dos pesos do modelo e o erro correspondente:

- $J_e(\mathbf{a}): \mathbb{R}^{K+1} \rightarrow \mathbb{R}$. Esse mapeamento define o que conhecemos como **superfície de erro**.

- $J_e(\mathbf{a})$ assume uma forma **quadrática** com respeito ao **vetor de pesos, \mathbf{a}** .

$$J_e(\mathbf{a}) = \|\mathbf{y} - \Phi\mathbf{a}\|^2 = \mathbf{y}^T \mathbf{y} - \mathbf{y}^T \Phi\mathbf{a}^T - \mathbf{a}^T \Phi^T \mathbf{y} + \mathbf{a}^T \Phi^T \Phi \mathbf{a}.$$

- Consequentemente, a superfície é **convexa** (ou seja, tem forma de tigela), e portanto possui um único **mínimo global**, que pode ser encontrado, por exemplo, pela **equação normal**.

- Isso é provado mostrando-se que $\frac{\partial^2 J_e(\mathbf{a})}{\partial^2 \mathbf{a}} = 2\Phi^T \Phi$ é uma **matriz positiva semi-definida**, e portanto, $J_e(\mathbf{a})$ sempre será **convexa** com relação ao vetor de pesos, \mathbf{a} .

Superfície de Erro: Exemplo #1

- A figura ao lado mostra a **superfície de erro** para a seguinte **função observável**:

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

onde $w(n) \sim N(0,1)$ e $y(n)$ é a **função objetivo**.

- Neste exemplo, a **função objetivo** (ou **modelo gerador**) é dada por:

$$y(n) = a_1 x_1(n),$$

onde $x_1(n) \sim N(0,1)$ e $a_1 = 1$.

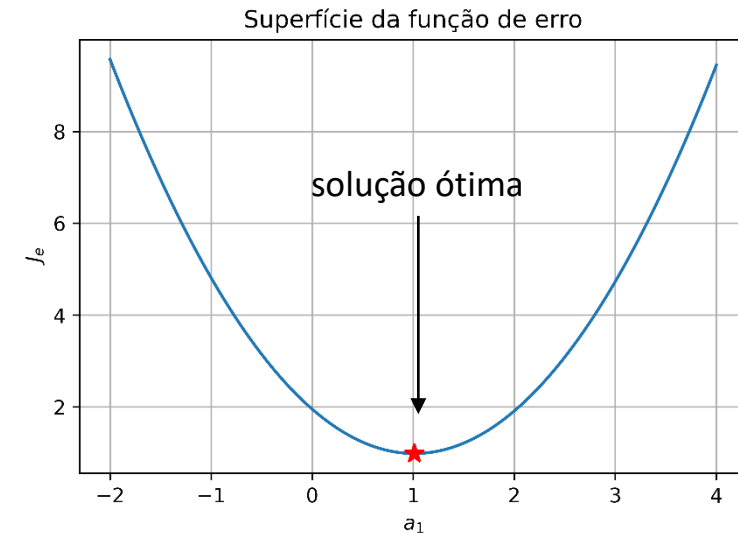
- A **função hipótese**, $h(x)$, é dada por

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

- O erro **erro quadrático médio** é calculado como

$$J_e(a_1) = \frac{1}{N} \sum_{n=0}^{N-1} \left(y_{\text{noisy}}(n) - \hat{y}(n) \right)^2 = \frac{1}{N} \sum_{n=0}^{N-1} \left(y_{\text{noisy}}(n) - \hat{a}_1 x_1(n) \right)^2.$$

[Exemplo: error_surface_example1.ipynb](#)



O erro é calculado variando-se a_1 na equação do EQM.

Superfície de Erro: Exemplo #2

[Exemplo: error surface example2.ipynb](#)

Superfície da função de erro

- A figura ao lado mostra a **superfície de erro** para a seguinte **função observável**:

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

onde $w(n) \sim N(0,1)$ e $y(n)$ é a **função objetivo**.

- Neste exemplo, a **função objetivo** (ou **modelo gerador**) é dada por:

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

onde $x_1(n)$ e $x_2(n) \sim N(0,1)$ e $a_1 = a_2 = 1$.

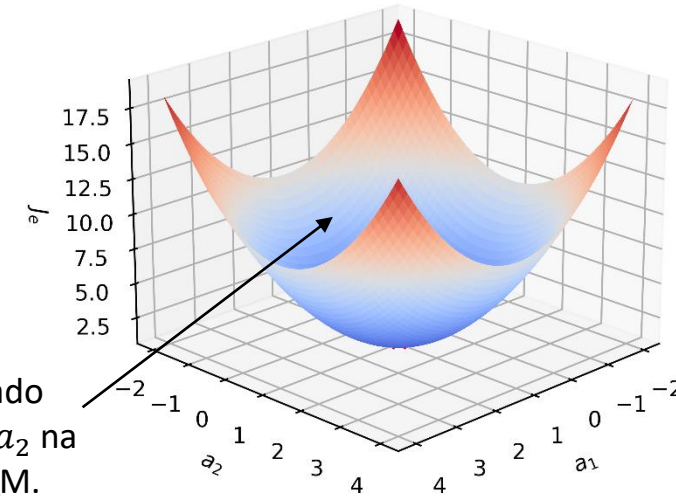
- A **função hipótese**, $h(x)$, é dada por

$$h(x) = \hat{y} = \hat{a}_1 x_1(n) + \hat{a}_2 x_2(n).$$

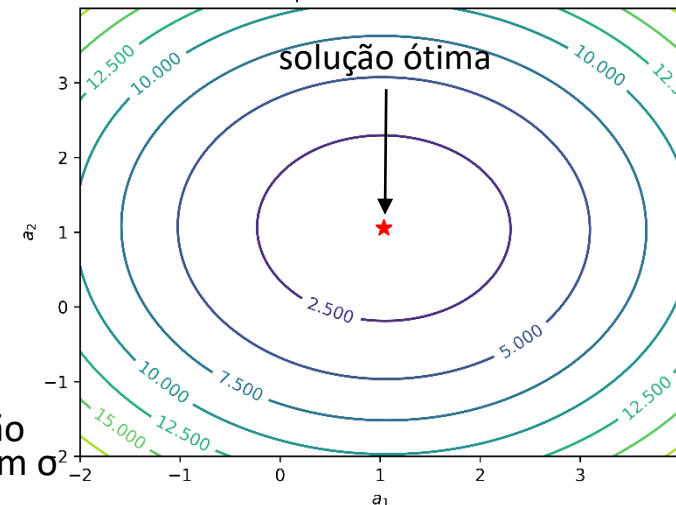
- O erro **erro quadrático médio** é calculado como

$$\begin{aligned} J_e(a_1, a_2) &= \frac{1}{N} \sum_{n=0}^{N-1} (y_{\text{noisy}}(n) - \hat{y}(n))^2 \\ &= \frac{1}{N} \sum_{n=0}^{N-1} (y_{\text{noisy}}(n) - (\hat{a}_1 x_1(n) + \hat{a}_2 x_2(n)))^2 \end{aligned}$$

O erro é calculado variando-se a_1 e a_2 na equação do EQM.



Superfície de contorno



- A segunda figura mostra a **superfície de contorno**.

- Uma linha de contorno de uma função de duas variáveis é uma curva ao longo da qual a função tem um valor constante. Ou seja, no nosso caso, cada uma das linhas indica uma curva que têm o mesmo erro.

Formatos diferentes para a superfície de erro

Função objetivo:

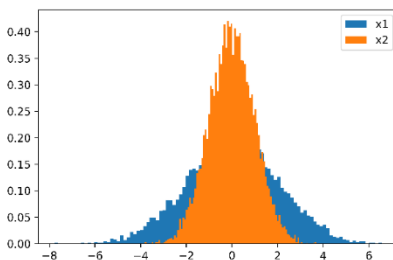
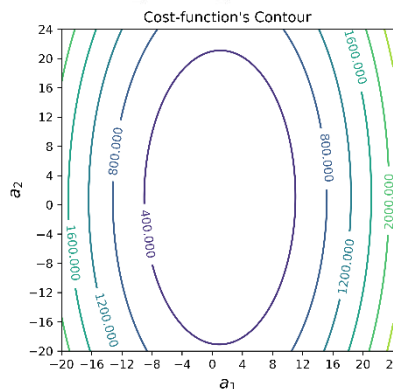
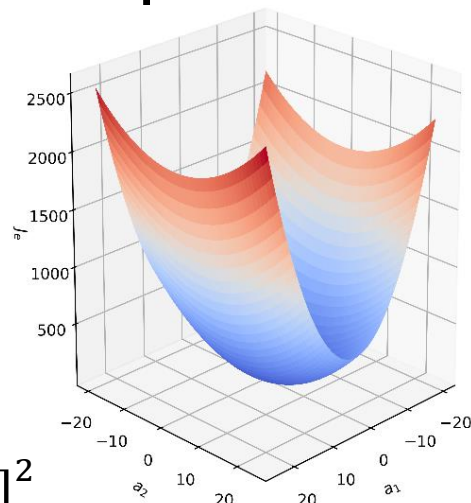
$$y(n) = a_1 x_1(n) + a_2 x_2(n),$$

onde $a_1 = 1, a_2 = 1$.

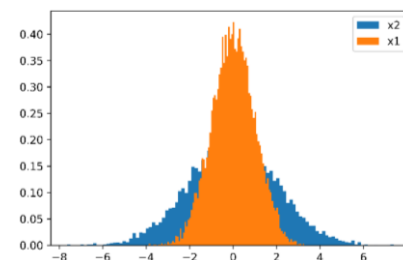
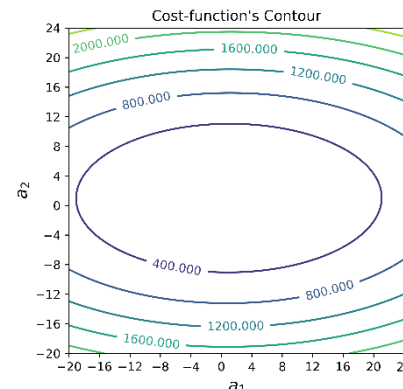
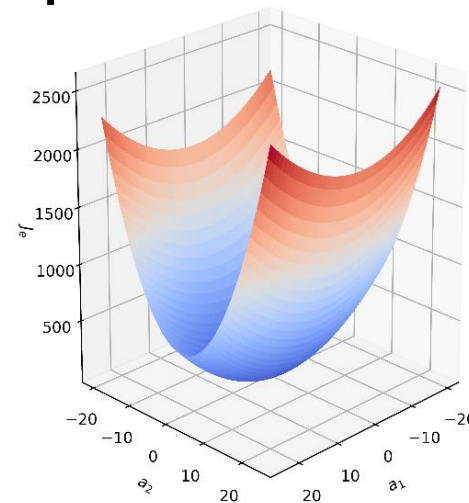
Para plotar a superfície de erro usamos:

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

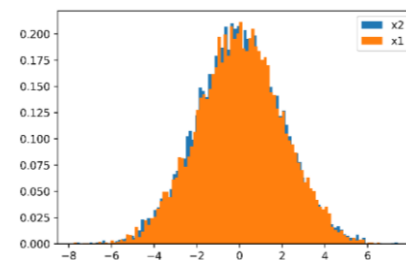
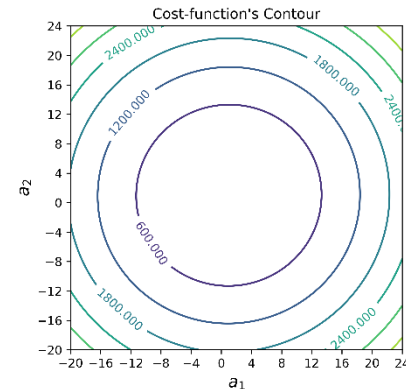
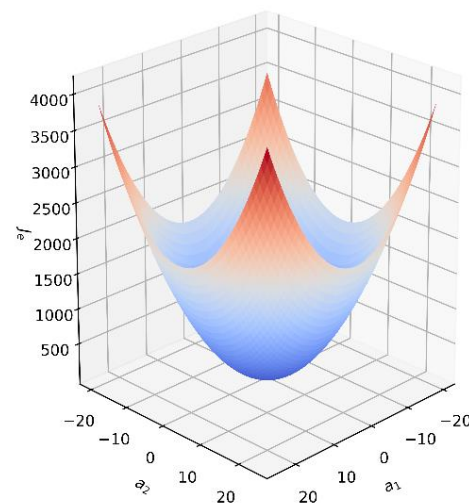
- Se x_1 varia entre um intervalo muito grande de valores, então o **peso** da variação de \hat{a}_1 no erro é maior, ou seja, o erro varia mais rapidamente com variações de \hat{a}_1 , resultando num vale.
- A mesma coisa pode ser dita para x_2 e \hat{a}_2 (vale).
- Quando x_1 e x_2 variam entre um intervalo semelhante, então, a variação tanto de \hat{a}_1 quanto de \hat{a}_2 tem **pesos** semelhantes na variação do erro (tigela).



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



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



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

[Exemplo: formatos diferentes da superfície de erro.ipynb](#)

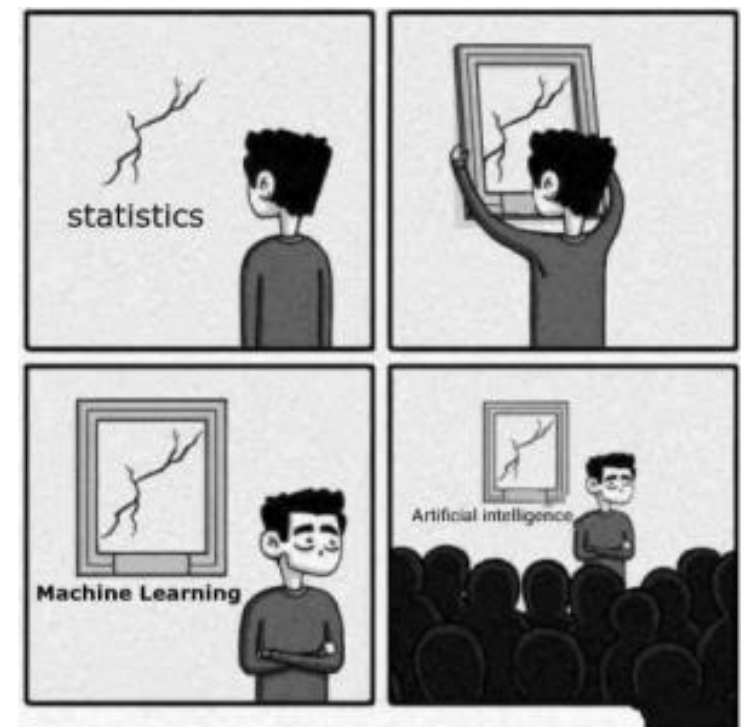
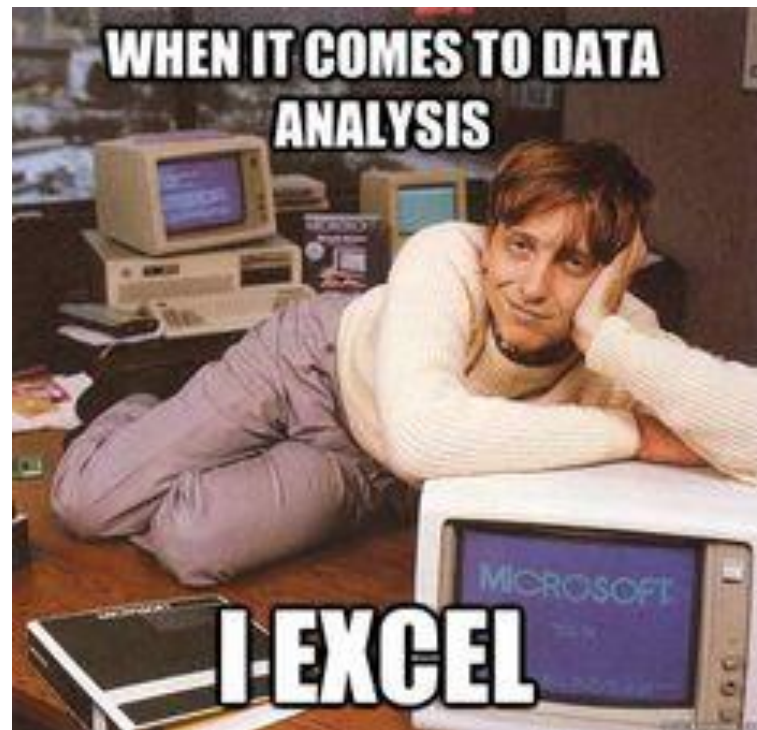
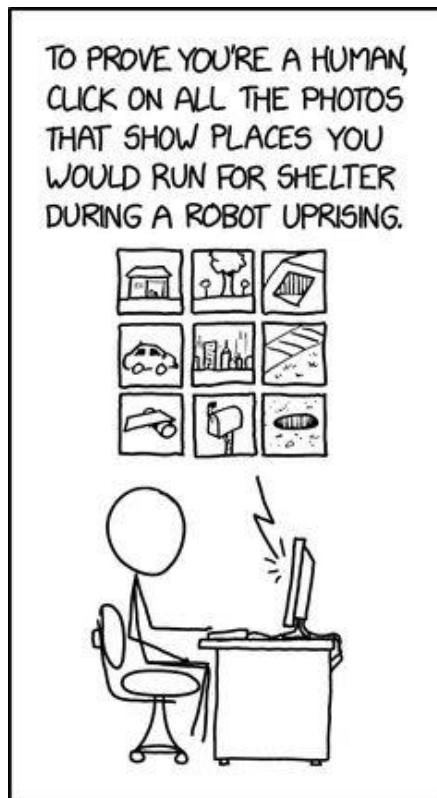
Desvantagens da forma fechada (Eq. Normal)

- **Alta complexidade computacional:** a solução da **equação normal** envolve o cálculo da inversa de $\Phi^T \Phi$, o qual tem complexidade computacional que varia de $O(K^{2.4})$ a $O(K^3)$.
 - **Exemplo:** Se o número de **atributos**, K , dobrar, o tempo para cálculo aumenta de $2^{2.4} = 5.3$ a $2^3 = 8$ vezes.
- Dependendo do número de **exemplos**, N , e **atributos**, x , a matriz Φ pode consumir muita memória.
- **Portanto, essa abordagem não é escalonável!**
- Adicionalmente, para irmos além dos modelos lineares (i.e., regressão não-linear) precisamos lidar com o fato de que não existem formas fechadas como a **equação normal**.
- **Solução:** abordagens iterativas
 - Métodos iterativos de busca que façam a atualização dos parâmetros, \mathbf{a} , à medida que os dados são apresentados ao modelo.
 - Por exemplo, o algoritmo do **gradiente descendente**, o qual veremos a seguir.

Tarefas

- **Quiz:** “*T319 - Quiz - Regressão: Parte I (1S2021)*” que se encontra no MS Teams.
- **Exercício Prático:** [Laboratório #2](#).
 - Pode ser baixado do MS Teams ou do GitHub.
 - Pode ser respondido através do link acima (na nuvem) ou localmente.
 - [Instruções para resolução e entrega dos laboratórios](#).
 - **Laboratórios podem ser feitos em grupo.**

Obrigado!



Albert Einstein: Insanity Is Doing
the Same Thing Over and Over Again
and Expecting Different Results

Machine learning:

