Regressão Linear Múltipla (Parte 1)

Onde estamos e para onde vamos?

Nas aula anteriores, aprendemos a fazer regressão linear com uma **única variável**:

Área da casa [m²] (x)	Custo em R$ (y)
32	51.000
149	265.000
78	110.000
...	...
220	315.000

Variável 1 (característica 1): Área da casa

Modelo Utilizado:

$$f_{w,b} = wx + b$$

Onde estamos e para onde vamos?

Agora estudaremos regressão linear com **múltiplas variáveis**:

Área da casa [m²] (x_1)	Número de quartos (x_2)	Andares (x_3)	Idade [anos] (x_4)	Custo em R$ (y)
32	1	1	2	51.000
149	3	2	10	265.000
78	2	1	30	110.000
...	...	...	...	...
220	4	2	5	315.000

Variável 1 (característica 1): Área da casa
Variável 2 (característica 2): Número de quartos
Variável 3 (característica 3): Número de andares
Variável 4 (característica 4): idade da construção

Levando em conta mais características, é esperado que o modelo seja capaz de prever melhor o valor de uma casa?

Regressão Linear Múltipla

Área da casa [m²] (x_1)	Número de quartos (x_2)	Andares (x_3)	Idade [anos] (x_4)	Custo em R$ (y)
32	1	1	2	51.000
149	3	2	10	265.000
78	2	1	30	110.000
...	...	...	...	...
220	4	2	5	315.000

Notação:

x_j = j-ésima característica $\rightarrow$ $j = 1, 2, \cdots, 4$

n = número total de características $\rightarrow$ $n = 4$

$\vec{x}^{(i)}$ = características do i-ésimo exemplo de treinamento $\rightarrow$ $\vec{x}^{(2)} = \begin{bmatrix} 149 & 3 & 2 & 10 \end{bmatrix}$

$\vec{x}_j^{(i)}$ = valor da característica j do i-ésimo exemplo de treinamento $\rightarrow$ $\vec{x}_2^{(2)} = 3$

Observação:

Para simplificar a notação, vamos tratar a sobre-barra como um elemento opcional de notação, tal que $\vec{x}_j^{(2)} = x_j^{(2)}$, por exemplo. $\rightarrow$ (serve apenas para enfatizar que trata-se de um vetor).

Seja o conjunto de dados abaixo. Quanto vale $\overrightarrow{x}_4^{(3)}$?

Área da casa [m²] (x_1)	Número de quartos (x_2)	Andares (x_3)	Idade [anos] (x_4)	Custo em R$ (y)
32	1	1	2	51.000
149	3	2	10	265.000
78	2	1	30	110.000
...	...	...	...	...
220	4	2	5	315.000

Regressão Linear Múltipla

Antes, na regressão linear com uma única variável, tínhamos o seguinte modelo:

$$f_{w,b}(x) = wx + b$$

Agora, na regressão linear com múltiplas variáveis, teremos:

$$f_{w,b}(x) = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$

Exemplo:

$$f_{w,b}(x) = 0.1x_1 + 4x_2 + 10x_3 - 2x_4 + 80 \quad \rightarrow \quad \text{valor da casa em milhares de R\$}$$

x_1: Área da casa
x_2: Número de quartos
x_3: Número de andares
x_4: idade da construção

Olhando com detalhes o modelo

Área da casa [m²] (x_1)	Número de quartos (x_2)	Andares (x_3)	Idade [anos] (x_4)	Custo em R$ (y)
32	1	1	2	51.000
149	3	2	10	265.000
78	2	1	30	110.000
...	...	...	...	...
220	4	2	5	315.000

O que significam os coeficientes do modelo?

$$f_{w,b}(x) = 0.1x_1 + 4x_2 + 10x_3 - 2x_4 + 80 \quad \rightarrow \quad \text{valor da casa em milhares de R\$}$$

- Note que cada m^2 adiciona R$ 100 ao valor da casa
- Cada quarto adiciona R$ 4000 ao valor da casa
- Cada andar adiciona R$ 10000 ao valor da casa
- Cada ano **reduz** em R$ 2000 o valor da casa
- R$ 80000 seria o "valor base" de qualquer casa

Olhando com detalhes o modelo

Área da casa [m²] (x_1)	Número de quartos (x_2)	Andares (x_3)	Idade [anos] (x_4)	Custo em R$ (y)
32	1	1	2	51.000
149	3	2	10	265.000
78	2	1	30	110.000
...	...	...	...	...
220	4	2	5	315.000

O modelo abaixo é bom para estimar o preço das casas?

$$f_{w,b}(x) = 0.1x_1 + 4x_2 + 10x_3 - 2x_4 + 80 \quad \rightarrow \quad \text{valor da casa em milhares de R\$}$$

- Para a primeira casa do conjunto de dados, temos:

$$f_{w,b}(x) = 0.1 \cdot 32 + 4 \cdot 1 + 10 \cdot 1 - 2 \cdot 2 + 80 = 93.2$$

- Para as demais casas, temos

$$f_{w,b}(x) = 0.1 \cdot 149 + 4 \cdot 3 + 10 \cdot 2 - 2 \cdot 10 + 80 = 106.9$$

$$f_{w,b}(x) = 0.1 \cdot 78 + 4 \cdot 2 + 10 \cdot 1 - 2 \cdot 30 + 80 = 45.8$$

$$f_{w,b}(x) = 0.1 \cdot 220 + 4 \cdot 4 + 10 \cdot 2 - 2 \cdot 5 + 80 = 128.0$$

Olhando com detalhes o modelo

Área da casa [m²] (x_1)	Número de quartos (x_2)	Andares (x_3)	Idade [anos] (x_4)	Custo em R$ (y)
32	1	1	2	51.000
149	3	2	10	265.000
78	2	1	30	110.000
...	...	...	...	...
220	4	2	5	315.000

Conclusão

Observando como o modelo em tela se comporta para os dados que temos, parece que um modelo mais assertivo poderia ter sido obtido.

Pergunta:

Como obter um modelo mais preciso?

Um modelo com n características

Um modelo com n características é dado por

$$f_{w,b}(x) = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$

Parâmetros do modelo:

$$\vec{w} = \begin{bmatrix} w_1 & w_2 & \cdots & w_n \end{bmatrix}$$

$b \quad \rightarrow \quad b$ não é um vetor, mas sim um escalar!

Características presentes no modelo:

$$\vec{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}$$

Utilizando essa notação, note que podemos reescrever $f_{w,b}(x)$ na seguinte forma compacta:

$$f_{w,b}(x) = \vec{w} \cdot \vec{x} + b$$

onde $\cdot$ denota o produto escalar, tal que

$$\vec{w} \cdot \vec{x} = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

Vetorização

Agora que começamos utilizar vetores, antes de continuarmos nossos estudos sobre a Regressão Linear Múltipla, falaremos sobre o conceito de **Vetorização**.

Importância da Vetorização

Ao implementar um algoritmo de Aprendizado de Máquina, a vetorização provê dois benefícios principais:

1. Seu código torna-se mais compacto.
2. Seu algoritmo é capaz de rodar de forma mais rápida e eficiente.

Um código escrito de forma vetorizada é também capaz de extrair máximo proveito de:

- bibliotecas numéricas modernas e otimizadas para cálculo matemático, como a biblioteca de Álgebra Linear **NumPy**, por exemplo.
- elementos de hardware voltados ao processamento intenso de dados, como GPUs (*Graphical Processing Units*), por exemplo.

Exemplo de Vetorização

Sejam os seguintes parâmetros e características de um modelo:

$$\vec{w} = \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \rightarrow (n = 3)$$

b (escalar)

$$\vec{x} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}$$

Em Álgebra Linear, a indexação (contagem de elementos) começa em 1.

Em código escrito em Python (NumPy), a indexação começa em 0.

Exemplo:

```
w = np.array([1.0, 2.5, -3.3])
b = 4
x = np.array([10, 20, 30])
```

Para acessar o primeiro elemento de w, usamos w[0].
Para acessar o segundo elemento de w, usamos w[1].
Para acessarmos o terceiro elemento de w, usamos w[2].

Idem para os elementos do vetor x.

Exemplo de Vetorização

Suponha que você quer implementar uma previsão feita pelo modelo

$$f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + w_3x_3 + b$$

Código sem vetorização:

```
f = w[0]*x[0] + w[1]*x[1] + w[2]*x[2] + b
```

Pergunta:

Seria fácil programar a linha de código acima se tivéssemos $n = 1000$?

Exemplo de Vetorização

Código ainda sem vetorização, mas usando loop for:

Sabemos que

$$f_{\vec{w},b}(\vec{x}) = \left(\sum_{j=1}^{n} w_j x_j \right) + b$$

Portanto poderíamos usar o seguinte código:

```
f = 0
for j in range(0,n):
        f = f + w[j] * x[j]
f = f+b
```

Observação 1: Em Python, `j in range(0,n)` significa que j será $0, 1, 2, \cdots, n-1$.

Observação 2: Em Python, o comando `range(0,n)` faz a mesma coisa que `range(n)`

Pergunta:

Apesar de ser melhor que a nossa primeira implementação, essa segunda seria ainda a forma mais otimizada para a realização dos cálculos?

Código COM vetorização:

Sabendo que

$$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$$

podemos implementar tal operação matemática usando uma **única linha de código**:

```
f = np.dot(w,x)+b
```

Exemplo de Vetorização

Observação:

Ao utilizarmos comandos da biblioteca NumPy, como f = np.dot(w,x)+b , por exemplo, estamos otimizando o nosso código, especialmente para os casos em que n é grande.

Isso acontece pois, por trás, a biblioteca **NumPy** utiliza **paralelismo de hardware** para a realização das operações matemáticas, mesmo que você esteja usando uma CPU comum ao invés de uma GPU.

Um segundo exemplo de Vetorização

Suponha que você deseja implementar o Método do Gradiente para um modelo com $b = 0$, $\overrightarrow{w} = (w_1, w_2, \cdots, w_{16})$, derivadas $\overrightarrow{d} = (d_1, d_2, \cdots, d_{16})$ e $\alpha = 0.1$, tal que

$$w_j = w_j - 0.1 d_j$$

Código sem vetorização:

```
for j in range(0,16):
        w[j] = w[j] - 0.1*d[j]
```

Código COM vetorização:

```
w = w - 0.1*d
```

De olho no código!

Vamos agora aprender conceitos mais específicos acerca de Python, NumPy e Vetorização

Nome do arquivo que trabalharemos agora:

codigo - Python, NumPy e Vetorização.ipynb

Observação:

Trata-se de uma revisão bastante importante sobre criação de vetores e matrizes usando a biblioteca NumPy. O código também demonstra, por meio de um exemplo, a importância da vetorização.

Atividade de aula

Parte 1

Rode todo o "codigo - Python, NumPy e Vetorização.ipynb" sem fazer qualquer tipo de alteração. Certifique-se de que você o compreendeu.

Parte 2

1. Qual foi a diferença de tempo observada entre rodar comandos usando "loop for" versus vetorização?