

Lista 1 de Machine Learning

Gabriel Vaz Cançado Ferreira

IMPA, Verão 2026

Sumário

1 Exercício 1a	3
2 Exercício 1b	4
3 Exercício 1c	5
4 Exercício 1d	6
5 Exercício 1e	7
6 Exercício 2a	8
7 Exercício 2b	9
8 Exercício 2c	11
9 Exercício 2d	13
10 Exercício 3a	18
11 Exercício 3b	19
12 Exercício 3c	20
13 Exercício 3d	21
14 Exercício 3e	22
15 Exercício 3f	24
16 Exercício 4a	29
17 Exercício 4b	30
18 Exercício 4c	32
19 Exercício 4d	34

20 Exercício 5a	36
21 Exercício 5b	37
22 Exercício 5d	43
23 Exercício 5d	44
24 Exercício 5e	46
25 Exercício 5f	48
26 Referências	49

1 Exercício 1a

Falso.

Apesar de um erro de teste baixo ser um resultado positivo para avaliar a precisão do modelo, o erro de teste ainda sim pode trazer informações valiosas relativas ao modelo. Por exemplo, se temos um erro de teste baixo e um erro de treino baixo podemos concluir que o modelo é bom, mas indica que pode-se reduzir a complexidade do modelo sem perder performance, algo sempre desejado quando se lida com largos volumes de dados. Caso se tenha um erro de teste baixo mas um erro de treino muito inferior, há uma indicação de que houve overfitting do modelo aos dados de treino. Por fim, caso o erro de teste seja baixo e o erro de treino seja alto, temos um indicativo de underfitting, ou seja, o modelo não foi capaz de se adequar as features dos próprios dados de treinamento, sugerindo que há espaço para um aumento de complexidade/robustez do modelo.

Logo, o análise do erro de treino produz informações valiosas acerca do trade-off viés-variância, nos dando indícios de como aumentar ou diminuir flexibilidade e robustez do modelo para reduzir a parte redutível do erro associada ao modelo.

2 Exercício 1b

Verdadeiro.

Em uma regressão linear, a realização de um teste t para testar a hipótese $H_0 : \beta_j = 0$ depende de hipóteses sobre a distribuição dos erros. Sob normalidade dos erros (condicionalmente a X), tem-se que o estimador de variância

$$\hat{\sigma}^2 = \frac{1}{n-p-1} \sum_{i=1}^n \hat{\varepsilon}_i^2 \quad \Rightarrow \quad \frac{(n-p-1)\hat{\sigma}^2}{\sigma^2} \sim \chi^2_{n-p-1},$$

e, quando σ é conhecida, o estatístico padronizado

$$Z = \frac{\hat{\beta}_j - \beta_j}{\text{SE}(\hat{\beta}_j)} \sim \mathcal{N}(0, 1).$$

Como σ é desconhecida na prática, substitui-se σ por $\hat{\sigma}$, obtendo

$$t = \frac{\hat{\beta}_j - 0}{\text{SE}(\hat{\beta}_j)} \sim t_{n-p-1},$$

o que fundamenta o teste t para $H_0 : \beta_j = 0$.

3 Exercício 1c

Considere a seguinte matriz de confusão para modelos de detecção de fraudes bancárias.

Previsto \ Real	Fraude	Não fraude
Fraude	TP (Verdadeiro Positivo)	FP (Falso Positivo)
Não fraude	FN (Falso Negativo)	TN (Verdadeiro Negativo)

A acurácia e a taxa de detecção de fraudes são dadas por:

$$\text{Acurácia} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}, \quad \text{Detecção de Fraudes} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

Suponha que os modelos 1 (98,5% de acurácia) e 2 (99% de acurácia) foram usados para detectar fraudes em um dataset com 1000 transações bancárias, das quais 50 eram fraudes e 950 eram transações legítimas.

Considere o seguinte exemplo para o modelo 1:

Previsto \ Real	Fraude	Não fraude
Fraude	TP = 49	FP = 14
Não fraude	FN = 1	TN = 936

Nesse caso, a acurácia é

$$\text{Acurácia} = \frac{\text{TP} + \text{TN}}{1000} = \frac{49 + 936}{1000} = \frac{985}{1000} = 98,5\%,$$

e a taxa de detecção de fraudes é

$$\text{Detecção de Fraudes} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{49}{49 + 1} = \frac{49}{50} = 98\%.$$

Considere o seguinte exemplo para o modelo 2:

Previsto \ Real	Fraude	Não fraude
Fraude	TP = 45	FP = 5
Não fraude	FN = 5	TN = 945

Nesse caso, a acurácia é

$$\text{Acurácia} = \frac{\text{TP} + \text{TN}}{1000} = \frac{45 + 945}{1000} = \frac{990}{1000} = 99\%,$$

e a taxa de detecção de fraudes é

$$\text{Detecção de Fraudes} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{45}{45 + 5} = \frac{45}{50} = 90\%.$$

Logo, como o banco está mais interessado em detectar fraudes, seu interesse está focado na taxa de detecção de fraudes (sensibilidade) e não somente na acurácia do modelo. Assim, um modelo com menor acurácia, porém com maior sensibilidade para detectar fraudes, pode ser preferível quando comparado a um modelo de maior acurácia, porém com mais falsos negativos (fraudes não detectadas).

4 Exercício 1d

Verdadeiro.

A regressão Ridge é definida por

$$\hat{\beta}_{\text{ridge}}(\lambda) = \arg \min_{\beta \in \mathbb{R}^p} \left\{ \text{RSS}(\beta) + \lambda \sum_{j=1}^p \beta_j^2 \right\}, \quad \lambda \geq 0,$$

onde

$$\text{RSS}(\beta) = \|y - X\beta\|_2^2.$$

Já a regressão linear por mínimos quadrados (OLS) é

$$\hat{\beta}_{\text{ls}} = \arg \min_{\beta \in \mathbb{R}^p} \text{RSS}(\beta).$$

Observa-se que, ao escolher $\lambda = 0$, obtemos

$$\hat{\beta}_{\text{ridge}}(0) = \hat{\beta}_{\text{ls}}.$$

Logo, existe sempre um valor de λ (em particular, $\lambda = 0$) que garante que a performance da regressão Ridge não seja pior do que a regressão linear, pois nesse caso os dois modelos coincidem.

5 Exercício 1e

Verdadeiro. O intervalo

$$\left[\hat{\beta}_j \pm 2\sqrt{\hat{\sigma}^2[(X^\top X)^{-1}]_{jj}} \right]$$

baseia-se em aproximações de normalidade para a distribuição de $\hat{\beta}_j$. Se os erros não têm distribuição Normal, essa aproximação pode ser inadequada e os intervalos de confiança obtidos via bootstrap se tornam uma alternativa mais apropriada por não imporem normalidade.

6 Exercício 2a

A hipótese de resíduos não-correlacionados,

$$\mathbb{E}[\varepsilon_i \varepsilon_j] = 0 \quad \text{para } i \neq j,$$

e de homoscedasticidade,

$$\text{Var}(\varepsilon_i) = \text{Var}(\varepsilon_j) = \sigma^2 \quad \text{para todo } i, j,$$

implica que a matriz de covariâncias dos erros

$$\Sigma = \text{Var}(\varepsilon)$$

é diagonal e tem todos os elementos da diagonal principal iguais. Logo,

$$\Sigma = \sigma^2 I.$$

7 Exercício 2b

(b.i) Provemos que

$$\hat{\beta}_\Sigma = (X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} Y.$$

Como o problema é convexo, basta derivar e igualar a 0:

$$\begin{aligned} \frac{\partial}{\partial \beta} (Y - X\beta)^T \Sigma^{-1} (Y - X\beta) &= -2X^T \Sigma^{-1} (Y - X\beta) = 0 \\ \Rightarrow X^T \Sigma^{-1} Y &= X^T \Sigma^{-1} X \beta \\ \Rightarrow \hat{\beta}_\Sigma &= (X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} Y. \end{aligned}$$

(b.ii) Provemos que

$$\mathbb{E}(\hat{\beta}_\Sigma) = \beta.$$

$$\begin{aligned} \mathbb{E}[\hat{\beta}_\Sigma] &= \mathbb{E}[(X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} y] \\ &= \mathbb{E}[(X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} (X\beta + \varepsilon)] \\ &= (X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} \mathbb{E}[X\beta + \varepsilon] \\ &= (X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} (\mathbb{E}[X\beta] + \mathbb{E}[\varepsilon]), \quad \mathbb{E}[\varepsilon] = 0 \\ &= (X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} X \beta \\ \Rightarrow \mathbb{E}[\hat{\beta}_\Sigma] &= \beta. \end{aligned}$$

(b-iii) Provemos que

$$\mathbb{V}[\hat{\beta}_\Sigma] = (X^T \Sigma^{-1} X)^{-1}.$$

$$\begin{aligned} \mathbb{V}[\hat{\beta}_\Sigma] &= \mathbb{V}[(X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} y] \\ &= \mathbb{V}[(X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} (X\beta + \varepsilon)] \\ &= (X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} \mathbb{V}[X\beta + \varepsilon] [(X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1}]^T \\ &= (X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} X \mathbb{V}[\beta] X^T [(X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1}]^T + (X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} \Sigma [(X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1}]^T \\ &= (X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} X (X^T \Sigma^{-1} X)^{-1} \\ \Rightarrow \mathbb{V}[\hat{\beta}_\Sigma] &= (X^T \Sigma^{-1} X)^{-1}. \end{aligned}$$

(b-iv) Como

$$\hat{\beta}_\Sigma = (X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} Y, \quad Y = X\beta + \varepsilon,$$

temos

$$\hat{\beta}_\Sigma = \beta + (X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} \varepsilon.$$

Defina

$$A := (X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1}.$$

Se $\varepsilon \sim \mathcal{N}(0, \Sigma)$, então, por transformação linear de normal multivariada,

$$A\varepsilon \sim \mathcal{N}(0, A\Sigma A^T),$$

logo

$$\hat{\beta}_\Sigma \sim \mathcal{N}(\beta, A\Sigma A^T).$$

Agora,

$$\begin{aligned} A\Sigma A^T &= (X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} \Sigma \Sigma^{-1} X [(X^T \Sigma^{-1} X)^{-1}]^T \\ &= (X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} X (X^T \Sigma^{-1} X)^{-1} = (X^T \Sigma^{-1} X)^{-1}. \end{aligned}$$

Portanto,

$$\hat{\beta}_\Sigma \sim \mathcal{N}(\beta, (X^T \Sigma^{-1} X)^{-1}).$$

8 Exercício 2c

(c.i) Provemos que

$$\hat{\beta}_\Sigma = \arg \min_{\beta} (Y - X\beta)^T \Sigma^{-1} (Y - X\beta).$$

é equivalente a

$$\hat{\beta}_\Sigma = \arg \min_{\beta} \sum_{i=1}^n w_i^2 \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2.$$

Sejam

$$Y = \begin{bmatrix} y_1 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix} \in \mathbb{R}^{n \times 1}, \quad X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n-1,1} & x_{n-1,2} & \cdots & x_{n-1,p} \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix} \in \mathbb{R}^{n \times p}.$$

$$\Sigma = \begin{bmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_n^2 \end{bmatrix} \in \mathbb{R}^{n \times n} \Rightarrow \Sigma^{-1} = \begin{bmatrix} w_1^2 & 0 & \cdots & 0 \\ 0 & w_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & w_n^2 \end{bmatrix} \in \mathbb{R}^{n \times n}.$$

$$\beta^T = [\beta_0 \ \ \beta_1 \ \ \cdots \ \ \beta_{p-1}].$$

Assim,

$$X\beta = \begin{bmatrix} \sum_{j=1}^p x_{1j}\beta_j + \beta_0 \\ \vdots \\ \sum_{j=1}^p x_{nj}\beta_j + \beta_0 \end{bmatrix} \in \mathbb{R}^{n \times 1}, \quad Y - X\beta = \begin{bmatrix} y_1 - \beta_0 - \sum_{j=1}^p x_{1j}\beta_j \\ \vdots \\ y_n - \beta_0 - \sum_{j=1}^p x_{nj}\beta_j \end{bmatrix} \in \mathbb{R}^{n \times 1}.$$

$$(Y - X\beta)^T = [y_1 - \beta_0 - \sum_{j=1}^p x_{1j}\beta_j \quad \cdots \quad y_n - \beta_0 - \sum_{j=1}^p x_{nj}\beta_j] \in \mathbb{R}^{1 \times n}.$$

$$(Y - X\beta)^T \Sigma^{-1} = [w_1^2(y_1 - \beta_0 - \sum_{j=1}^p x_{1j}\beta_j) \quad \cdots \quad w_n^2(y_n - \beta_0 - \sum_{j=1}^p x_{nj}\beta_j)] \in \mathbb{R}^{1 \times n}.$$

$$(Y - X\beta)^T \Sigma^{-1} (Y - X\beta) = \sum_{i=1}^n w_i^2 \left(y_i - \beta_0 - \sum_{j=1}^p x_{ij}\beta_j \right)^2.$$

(c-ii)

A escolha dos pesos $w_i^2 = \frac{1}{\sigma_i^2}$, uma vez que σ_i^2 mede a variância dos dados (isto é, seu ruído/incerteza), sugere que observações com maior ruído recebam menor peso na soma de resíduos quadrados, valorizando, assim, as observações mais precisas (com menor incerteza).

(c-iii) Prove que

$$\hat{\beta} = \arg \min_{\beta} (\tilde{Y} - \tilde{X}\beta)^T (\tilde{Y} - \tilde{X}\beta), \quad \text{onde } \tilde{Y} = \Sigma^{-1/2}Y \text{ e } \tilde{X} = \Sigma^{-1/2}X.$$

$$\begin{aligned} (\tilde{Y} - \tilde{X}\beta)^T (\tilde{Y} - \tilde{X}\beta) &= (\Sigma^{-1/2}Y - \Sigma^{-1/2}X\beta)^T (\Sigma^{-1/2}Y - \Sigma^{-1/2}X\beta) \\ &= (\Sigma^{-1/2}(Y - X\beta))^T (\Sigma^{-1/2}(Y - X\beta)) = (Y - X\beta)^T (\Sigma^{-1/2})^T \Sigma^{-1/2} (Y - X\beta) \\ &= (Y - X\beta)^T \Sigma^{-1} (Y - X\beta). \end{aligned}$$

9 Exercício 2d

(d-i) Dados gerados por:



```
import numpy as np
from scipy import stats

n = 50
Sigma = np.diag([10**((i-20)/5) for i in range(1,n+1)])
np.random.seed(0)
X = np.array([np.ones(n),np.random.normal(0,1,n)]).T
beta = np.array([1,0.25])
epsilon = np.random.multivariate_normal(np.zeros(n),Sigma)
y = X @ beta + epsilon
```

(d-ii) Calculamos β_{ls} e β_{Σ} :



```
# beta least squares

beta_ls = np.linalg.inv(X.T @ X) @ X.T @ y

print('Beta_ls = ',beta_ls)

# beta sigma
A = X.T @ np.linalg.inv(Sigma) @ X
beta_sig = np.linalg.inv(A) @ X.T @ np.linalg.inv(Sigma) @ y
print('Beta_sigma = ',beta_sig)
```

Obtendo:

$$\begin{aligned}\beta_{ls} &= [-34.46344733 \ 6.94756948] \\ \beta_{\Sigma} &= [1.01885254 \ 0.24436202]\end{aligned}$$

Em seguida podemos calcular o erro quadrático dos estimadores:



```
sum_ls = np.sum((beta - beta_ls)**2)
sum_sig = np.sum((beta - beta_sig)**2)

print(sum_ls, sum_sig)
```

Obtendo:

$$\|\beta - \hat{\beta}_{ls}\|_2^2 = 1302.5135337720312$$

e

$$\|\beta - \hat{\beta}_{\Sigma}\|_2^2 = 0.000387205026253018$$

Logo, vemos que o beta estimado pelo modelo ponderado demonstrou um erro quadrático muito inferior ao método de mínimos quadrados.

(d-iii) Supondo que os dados foram gerados por meio de um vetor de resíduos ε com $\Sigma = \sigma^2 I$. Para calcular o p -valor, seguimos o seguinte procedimento. A variância σ^2 é estimada por

$$\hat{\sigma}^2 = \frac{1}{n - (p + 1)} \sum_{i=1}^n (y_i - \hat{y}_i)^2.$$

Em seguida, o estatístico t associado ao coeficiente $\hat{\beta}_j$ é calculado por

$$t^{(j)} = \frac{\hat{\beta}_j}{\sqrt{\hat{\sigma}^2 [(X^\top X)^{-1}]_{jj}}}.$$

Por fim, assumindo que $t^{(j)} \sim t_{n-p-1}$ sob $H_0 : \beta_j = 0$, calcula-se o p -valor por

$$p\text{-valor} = 2 \left(1 - F_{t_{n-p-1}} \left(|t^{(j)}| \right) \right),$$

onde $F_{t_{n-p-1}}(\cdot)$ é a função de distribuição acumulada da t de Student com $n - p - 1$ graus de liberdade.



```
p = X.shape[1]
y_hat = X@beta_ls

sigma2_hat = (1/(n - p - 1))*np.sum((y - y_hat)**2)

Sigma_linha = sigma2_hat * np.identity(n = n)

# Para testar H0: bo = 0:

t0 = beta_ls[0]/np.sqrt(sigma2_hat * np.linalg.inv(X.T@X)[0,0])

df = n - p - 1

p_bilateral = 2*(1 - stats.t.cdf(abs(t0), df))
```

Por fim, obtivemos p -valor = 0.15876753326459347. Esse valor é significativo e não nos permite rejeitar a hipótese nula.

(d-iv)
Calculando a estatística

$$Z = (\hat{\beta}_\Sigma)_q^\top (X^\top \Sigma^{-1} X)^{-1}.$$

por meio de:



```
Z = beta_sig[0]/np.sqrt(np.linalg.inv(X.T @ np.linalg.inv(Sigma) @ X)[1,1])
```

Obtemos $Z = 105.75329561203317$

(d-v)

Condicionado em X , o estimador pode ser escrito como

$$\hat{\beta}_\Sigma = (X^\top \Sigma^{-1} X)^{-1} X^\top \Sigma^{-1} y = \beta + (X^\top \Sigma^{-1} X)^{-1} X^\top \Sigma^{-1} \varepsilon.$$

Se $\varepsilon \mid X \sim \mathcal{N}(0, \Sigma)$, então, por ser uma transformação linear de uma Normal,

$$\hat{\beta}_\Sigma \mid X \sim \mathcal{N}\left(\beta, (X^\top \Sigma^{-1} X)^{-1}\right).$$

Para testar $H_0 : \beta_0 = 0$ vs. $H_1 : \beta_0 \neq 0$, usamos

$$Z = \frac{(\hat{\beta}_\Sigma)_0 - 0}{\sqrt{[(X^\top \Sigma^{-1} X)^{-1}]_{00}}}.$$

Sob H_0 , vale que

$$Z \mid X \sim \mathcal{N}(0, 1).$$

Assim, o p -valor é dado por

$$p\text{-valor} = 2(1 - \Phi(|Z_{\text{obs}}|)) = 2\Phi(-|Z_{\text{obs}}|),$$

onde $\Phi(\cdot)$ é a função de distribuição acumulada da Normal padrão.

Em python, o cálculo pode ser realizado da seguinte forma:



```
from scipy.stats import norm

p_valor = 2*(1 - norm.cdf(abs(Z)))
```

Figura 1: Enter Caption

Onde obtemos $p\text{-valor} = \text{np.float64}(0.0)$, uma vez que nosso valor de $(\hat{\beta}_\Sigma)_0$ é muito próximo de 1. Logo, a hipótese deve ser rejeitada.

10 Exercício 3a

Uma vez que $\varepsilon_i \stackrel{iid}{\sim} \text{Laplace}(0, b)$ e que a distribuição Laplace possui a propriedade de translação: se $U \sim \text{Laplace}(0, b)$, então $U + \mu \sim \text{Laplace}(\mu, b)$, para qualquer $\mu \in \mathbb{R}$. Logo, condicionado em $X_i = x_i$, temos

$$Y_i | X_i = x_i = \beta^\top x_i + \varepsilon_i.$$

Pela propriedade acima, segue que

$$Y_i | X_i = x_i \sim \text{Laplace}(\beta^\top x_i, b),$$

isto é,

$$Y_i | X_i \sim \text{Laplace}(\beta^\top X_i, b).$$

A função de verossimilhança é a densidade de probabilidade conjunta dos dados observados, vista como função dos parâmetros desconhecidos.

A densidade de uma variável aleatória com distribuição Laplace(μ, b) é

$$f(y | \mu, b) = \frac{1}{2b} \exp\left(-\frac{|y - \mu|}{b}\right).$$

Portanto, a densidade condicional de y_i dado x_i é

$$f(y_i | x_i, \beta, b) = \frac{1}{2b} \exp\left(-\frac{|y_i - \beta^\top x_i|}{b}\right).$$

Como os erros ε_i são iid, segue que, condicionado em X , as variáveis Y_1, \dots, Y_n são independentes. Assim, a densidade conjunta (condicional em X) do vetor $y = (y_1, \dots, y_n)$ é o produto das densidades individuais:

$$f(y | X, \beta, b) = \prod_{i=1}^n f(y_i | x_i, \beta, b).$$

Logo, a verossimilhança de β será

$$\begin{aligned} L(\beta) &= f(y | X, \beta, b) = \prod_{i=1}^n \frac{1}{2b} \exp\left(-\frac{|y_i - \beta^\top x_i|}{b}\right). \\ L(\beta) &= \left(\frac{1}{2b}\right)^n \exp\left(-\frac{1}{b} \sum_{i=1}^n |y_i - \beta^\top x_i|\right). \end{aligned}$$

Tirando o logarítmico, obtemos

$$\ell(\beta) = \log L(\beta) = -n \log(2b) - \frac{1}{b} \sum_{i=1}^n |y_i - \beta^\top x_i|.$$

11 Exercício 3b

Em problemas de regressão linear, buscamos maximizar a função de verossimilhança a fim de obter o estimador de máxima verossimilhança. No caso de erros Laplacionos, a log-verossimilhança pode ser escrita como

$$\ell(\beta) = -n \log(2b) - \frac{1}{b} \sum_{i=1}^n |y_i - \beta^\top x_i|.$$

Como $-n \log(2b)$ e o fator multiplicativo $1/b$ não dependem de β , maximizar $\ell(\beta)$ é equivalente a minimizar

$$\sum_{i=1}^n |y_i - \beta^\top x_i|.$$

Assim, uma função-perda que independe de b é

$$\mathcal{L}(\beta) = \sum_{i=1}^n |y_i - \beta^\top x_i|.$$

12 Exercício 3c

No caso de erros gaussianos, a probabilidade condicional de $Y \mid X_i$ é dada por

$$L(\beta) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - \beta^\top x_i)^2}{2\sigma^2}\right).$$

Podemos reescrever como

$$L(\beta) = \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^n \exp\left(-\sum_{i=1}^n \frac{(y_i - \beta^\top x_i)^2}{2\sigma^2}\right).$$

Tomando o logaritmo da função de verossimilhança, obtemos

$$\ell(\beta) = -n \ln(\sqrt{2\pi}\sigma) - \sum_{i=1}^n \frac{(y_i - \beta^\top x_i)^2}{2\sigma^2}.$$

Logo, a função perda será

$$L(\beta) = \sum_{i=1}^n (y_i - \beta^\top x_i)^2.$$

A função perda alcançada utilizando erros laplacianos nos leva a uma perda L1, ao passo que utilizando erros gaussianos temos uma perda L2. Assim, a perda L2 penaliza quadraticamente os erros, dando importância bem maior a erros extremos. Em conjuntos de dados que temos alguns outliers que desejamos ignorar, a perda L1 atribui menos importância a estes eventos, mantendo, assim, maior coesão nas previsões.

13 Exercício 3d

O método do gradiente descendente é dado por

$$\theta^{(t)} = \theta^{(t-1)} - \rho_t \nabla f(\theta^{(t-1)}).$$

No nosso caso, temos

$$\beta^{(t)} = \beta^{(t-1)} - \eta \nabla L(\beta^{(t-1)}),$$

onde tomamos como função de perda

$$L(\beta) = \sum_{i=1}^n |y_i - \beta^\top x_i|.$$

Sabemos que $\frac{d}{dz}|z| = \text{sgn}(z)$, logo

$$\frac{dL(\beta)}{d\beta} = \sum_{i=1}^n \frac{d}{d\beta} |y_i - \beta^\top x_i| = \sum_{i=1}^n -x_i \text{sgn}(y_i - \beta^\top x_i).$$

Portanto, a regra de atualização do gradiente descendente será

$$\beta^{(t)} = \beta^{(t-1)} + \eta \sum_{i=1}^n x_i \text{sgn}(y_i - (\beta^{(t-1)})^\top x_i).$$

14 Exercício 3e

(i) $n = 1 \Rightarrow (x, y)$

\Rightarrow O passo laplaciano depende de

$$-\eta x \operatorname{sgn}\left(y - (\beta^{t-1})^\top x\right)$$

\Rightarrow O passo gaussiano depende de

$$-2\eta x \left(y - (\beta^{t-1})^\top x\right)$$

Logo, quando $\beta^T x$ está próximo de y , o passo laplaciano se mantém constante e com magnitude $|x|$, enquanto o passo laplaciano tem magnitude $|rx| - > 0$ uma vez que $r - > 0$. Logo, para $n = 1$ o passo para o modelo com erros laplacianos é sempre maior em magnitude.

(ii)

Vamos analisar os possíveis casos:

- Caso $(y - (\beta^{t-1})^\top x) > 0$:

$$\begin{aligned} \text{Passo Laplaciano} &= -\eta x, \\ \text{Passo Gaussiano} &= -2\eta x r, \quad r > 0. \end{aligned}$$

Mesma direção.

- Caso $(y - (\beta^{t-1})^\top x) = 0$:

$$\begin{aligned} \text{Passo Laplaciano} &= 0, \\ \text{Passo Gaussiano} &= 0. \end{aligned}$$

Mesma direção.

- Caso $(y - (\beta^{t-1})^\top x) < 0$:

$$\begin{aligned} \text{Passo Laplaciano} &= \eta x, \\ \text{Passo Gaussiano} &= 2\eta x r, \quad r > 0. \end{aligned}$$

Mesma direção.

Logo, a afirmação é verdadeira.

(iii) Não. Para $n > 1$, os modelos com erro Laplaciano e Gaussiano não necessariamente terão passos de gradiente na mesma direção, pois o gradiente Gaussiano é ponderado pela magnitude dos erros, enquanto o gradiente Laplaciano depende apenas do sinal dos erros. Eles coincidem apenas em casos especiais, como quando todos os resíduos têm o mesmo sinal e magnitudes semelhantes.

15 Exercício 3f

- (i) Completando o código para calcular $\hat{\beta}$ para os erros Gaussianos e Laplacianos.



```
import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt

np.random.seed(1)
beta = np.array([-1.5, 2])
input_range = np.linspace(-1, 1, 100)
X = np.vstack([np.ones(100), input_range]).T
y = X @ beta + np.random.normal(0, 0.3, 100)
def calculate_laplacian_loss(parameters, X, y):
    return np.sum(np.abs(y - X @ parameters))

params = np.array([1, 1])
beta_hat_gaussian = np.linalg.inv(X.T @ X) @ X.T @ y
beta_hat_laplacian = minimize(
    calculate_laplacian_loss,
    x0=beta_hat_gaussian,
    args=(X, y),
    method="Nelder-Mead"
)

print('Beta hat para erros Gaussianos: ', beta_hat_gaussian)
print('Beta hat para erros Laplacianos: \n', beta_hat_laplacian)
```

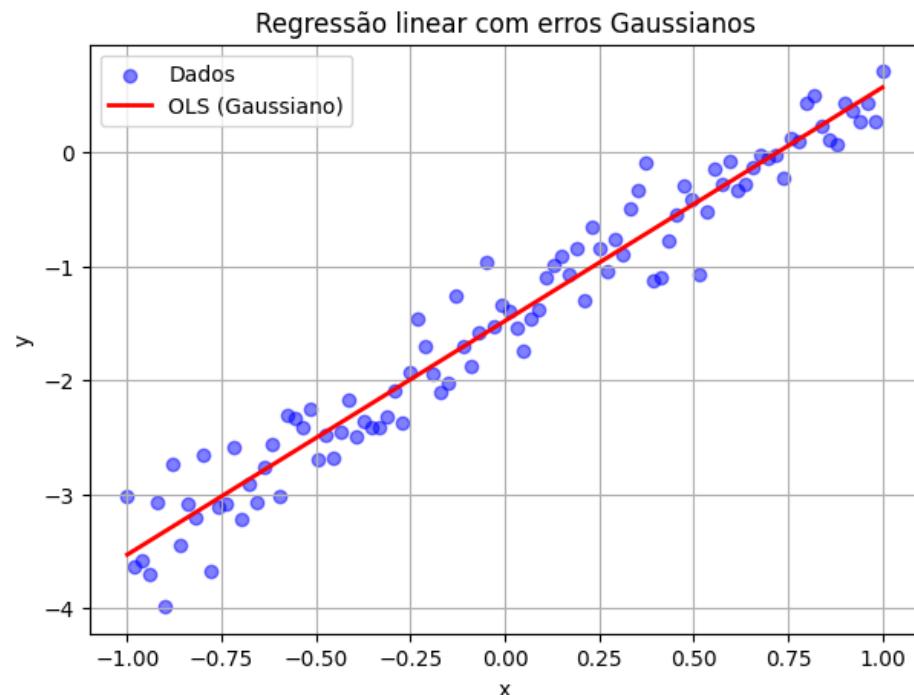
Resultados:

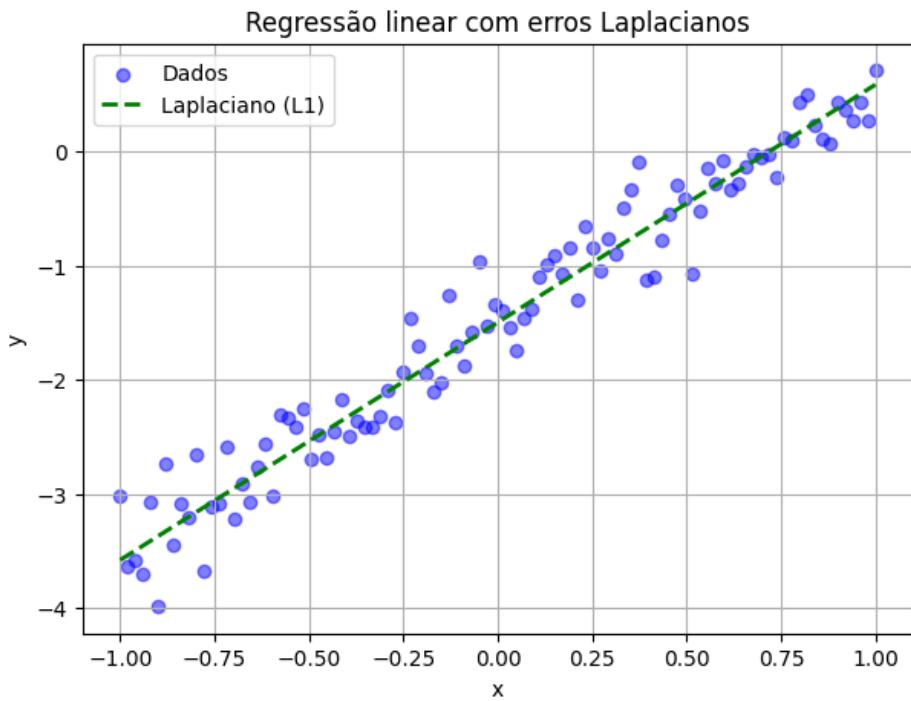


```
Beta hat para erros Gaussianos: [-1.48182514  2.0496267]
Beta hat para erros Laplacianos:
  message: Optimization terminated successfully.
  success: True
  status: 0
  fun: 20.634522953702422
  x: [-1.498e+00  2.082e+00]
  nit: 30
  nfev: 59
final_simplex: (array([[ -1.498e+00,   2.082e+00],
       [ -1.498e+00,   2.082e+00],
       [ -1.498e+00,   2.082e+00]]), array([ 2.063e+01,  2.063e+01,  2.063e+01]))
```

(ii) Código para plot das regressões lineares:

```
● ● ●  
plt.figure(figsize=(7,5))  
  
plt.title('Regressão linear com erros Gaussianos')  
plt.scatter(input_range, y, alpha=0.5, label="Dados", color = 'blue')  
  
y_hat_gaussian = X @ beta_hat_gaussian  
  
plt.plot(input_range, y_hat_gaussian, label="OLS (Gaussiano)", linewidth=2, color = 'red')  
  
plt.xlabel("x")  
plt.ylabel("y")  
plt.legend()  
plt.grid(True)  
  
plt.show()  
  
plt.figure(figsize=(7,5))  
  
plt.title('Regressão linear com erros Laplacianos')  
plt.scatter(input_range, y, alpha=0.5, label="Dados", color = 'blue')  
  
y_hat_laplacian = X @ beta_hat_laplacian.x  
  
plt.plot(input_range, y_hat_laplacian, label="Laplaciano (L1)", linewidth=2, linestyle="--", color = 'green')  
  
plt.xlabel("x")  
plt.ylabel("y")  
plt.legend()  
plt.grid(True)  
  
plt.show()
```





Cálculo de $\|\beta - \hat{\beta}\|_2^2$:

```
● ● ●
erro_gaussian = np.sum((beta - beta_hat_gaussian)**2)
erro_laplacian = np.sum((beta - beta_hat_laplacian['x'])**2)

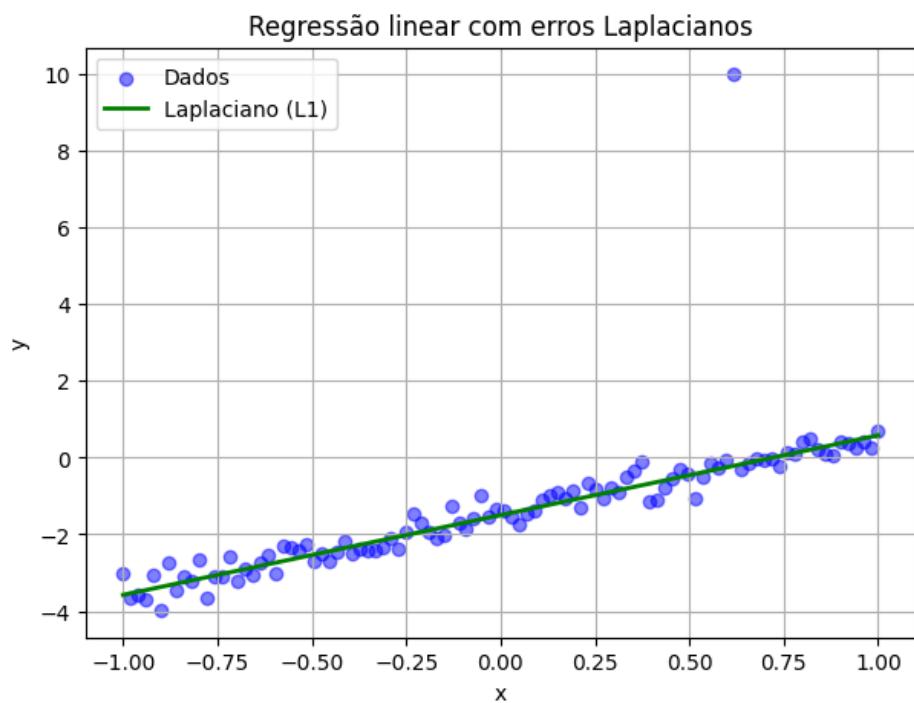
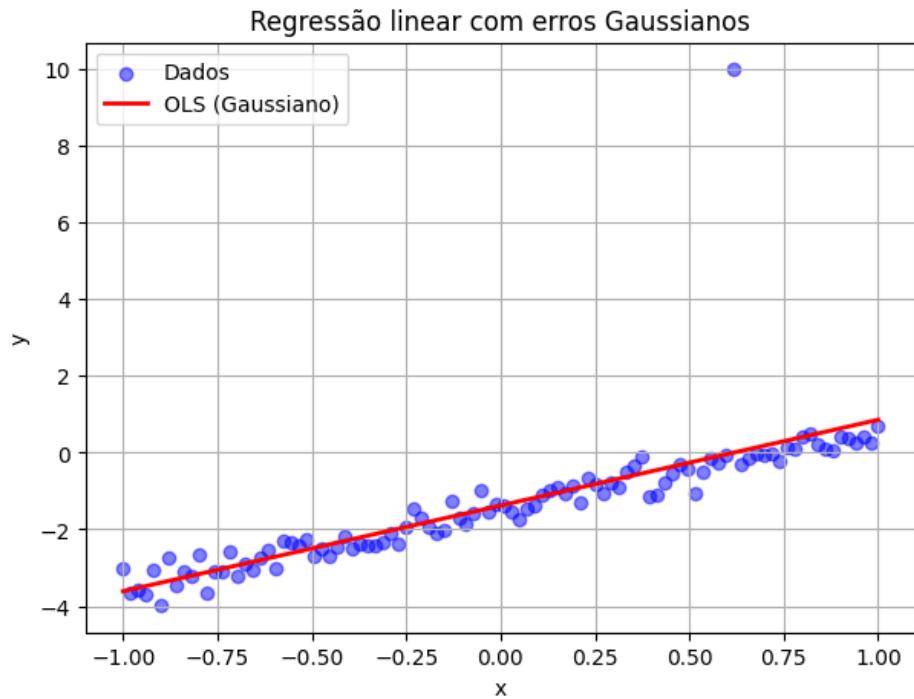
print('Erro quadrático para beta calculado com erros gaussianos = ', erro_gaussian)
print('Erro quadrático para beta calculado com erros laplacianos = ', erro_laplacian)
```

Resultado:

```
● ● ●
Erro quadrático para beta calculado com erros gaussianos =  0.0027931346724899653
Erro quadrático para beta calculado com erros laplacianos =  0.006743736731706675
```

Ou seja, o modelo com erros gaussianos obteve menor erro quadrático no cálculo do estimador $\hat{\beta}$

(iii) Adicionando o outlier $y[80] = 10$, obtemos os seguintes resultados:





```
Erro quadrático para beta calculado com erros gaussianos = 0.07087573216817813
Erro quadrático para beta calculado com erros laplacianos = 0.006784102400615049
```

Por fim, observamos aumento significativo no erro apresentado pelo modelo que utiliza erros gaussianos, demonstrando a sensibilidade deste a outliers e também a baixa sensibilidade do modelo que utiliza erros laplacianos a outliers. Nesse caso, passamos a escolher o modelo que considera que os erros seguem uma distribuição laplaciana.

16 Exercício 4a

A normalização dos dados é necessária por alguns motivos. O KNN utiliza métricas de distância como a distância eucliadiana, logo, quanto maior a escala de uma feature mais provável que essa feature domine as outras distâncias tornando outras features irrelevantes para a classificação. Além disso, modelos que assumem distribuições gaussianas para os dados utilizam médias e matrizes de covariância, fazendo com que features de maior variância dominem a matriz de covariância, além da possibilidade de levar a uma matriz não definida. Por fim, se desejamos comparar modelos é necessário que a escala dos dados não impacte a performance de nenhum dos modelos, pois isso levaria a conclusões imprecisas acerca da qualidade dos modelos implementados.

17 Exercício 4b

Treino dos modelos e previsões nos dados de treino e de teste:



```
from sklearn.metrics import accuracy_score

lda = LDA()
lda_fit = lda.fit(X_train, y_train)

qda = QDA()
qda_fit = qda.fit(X_train, y_train)

lr = LR()
lr_fit = lr.fit(X_train, y_train)

nb = NB()
nb_fit = nb.fit(X_train, y_train)

knn = kNN(n_neighbors=5)
knn_fit = knn.fit(X_train, y_train)

models = {
    "LDA": lda_fit,
    "QDA": qda_fit,
    "LogisticRegression": lr_fit,
    "NaiveBayes": nb_fit,
    "KNN": knn_fit
}

for name, m in models.items():
    y_pred_test = m.predict(X_test)
    y_pred_train = m.predict(X_train)

    acc_test = accuracy_score(y_test, y_pred_test)
    acc_train = accuracy_score(y_train, y_pred_train)

    print(name)
    print('Acurácia de teste = ', acc_test)
    print('Acurácia de treino = ', acc_train)
```

Resultados:



LDA

Acurácia de teste = 0.678125

Acurácia de treino = 0.709765625

QDA

Acurácia de teste = 0.6546875

Acurácia de treino = 0.70390625

LogisticRegression

Acurácia de teste = 0.6796875

Acurácia de treino = 0.711328125

NaiveBayes

Acurácia de teste = 0.678125

Acurácia de treino = 0.703515625

kNN

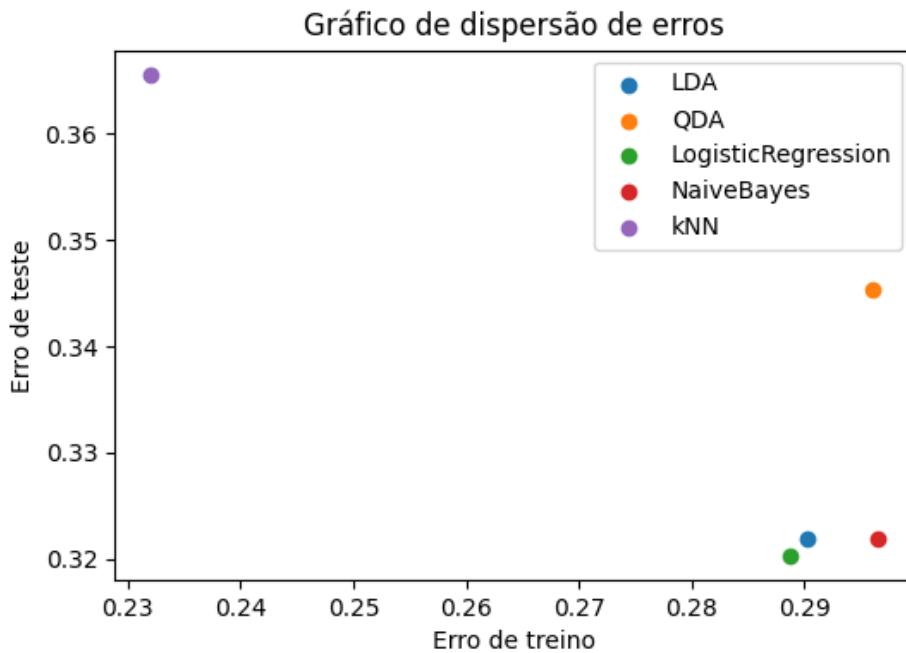
Acurácia de teste = 0.634375

Acurácia de treino = 0.76796875

18 Exercício 4c

Cálculo dos erros e plotagem do gráfico de dispersão:

```
● ● ●  
errors = []  
  
for name, m in models.items():  
    y_pred_test = m.predict(X_test)  
    y_pred_train = m.predict(X_train)  
  
    acc_test = accuracy_score(y_test, y_pred_test)  
    acc_train = accuracy_score(y_train, y_pred_train)  
  
    errors[name] = (1-acc_train, 1-acc_test)  
  
plt.figure(figsize=(6,4))  
plt.title('Gráfico de dispersão de erros')  
  
for model in models:  
    plt.scatter(x = errors[model][0], y = errors[model][1], label = model)  
  
plt.xlabel('Erro de treino')  
plt.ylabel('Erro de teste')  
plt.legend()  
plt.show()
```

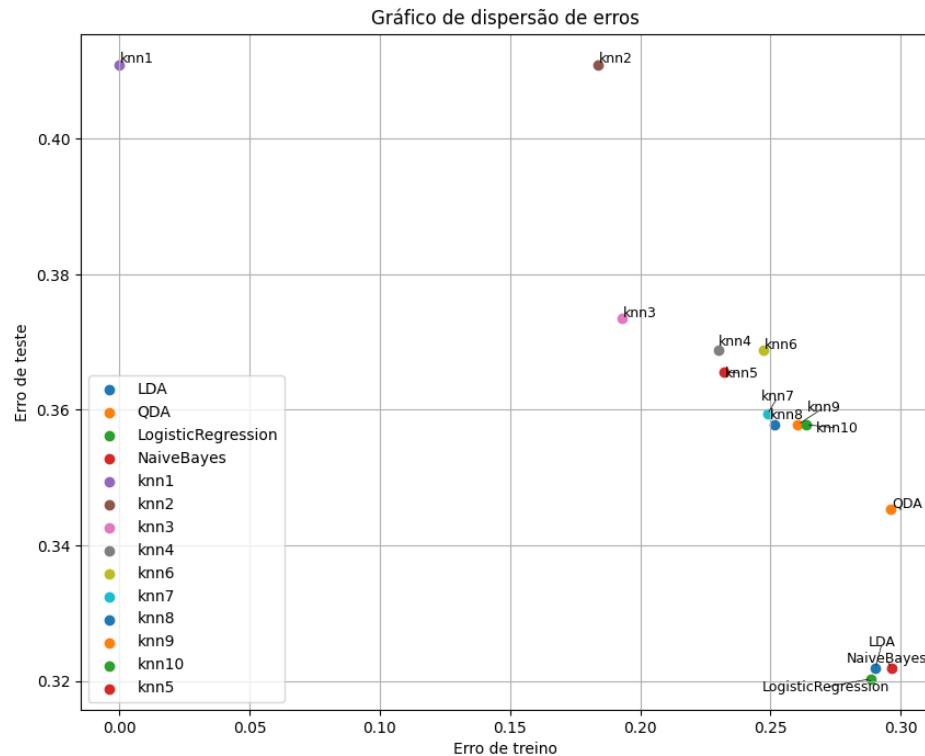


Dentre os modelos analisadosm o KNN foi o que apresentou maior erro de teste, ao mesmo tempo em que foi o com menor erro de treino, nos levando a concluir um possível overfitting para $k = 5$. Dentre os modelos restantesm todos tiveram erros de treino muito próximos, ao passo que dentre eles somente o QDA teve erro de teste consideravelmente superior aos demais, nos levando a concluir que as hipóteses utilizadas por tal modelo não sejam adequadas aos dados utilizados.

19 Exercício 4d

Repetindo o procedimento para $k = 1, 2, \dots, 10$; obtemos

```
● ● ●  
from adjustText import adjust_text  
  
plt.figure(figsize=(10,8))  
plt.title('Gráfico de dispersão de erros')  
  
texts = []  
for model, (x_val, y_val) in errors.items():  
    plt.scatter(x_val, y_val, label=model)  
    texts.append(plt.text(x_val, y_val, model, fontsize=9))  
  
plt.xlabel('Erro de treino')  
plt.ylabel('Erro de teste')  
plt.legend()  
plt.grid(True)  
  
adjust_text(  
    texts,  
    arrowprops=dict(arrowstyle="-", lw=0.5),  
    expand_points=(1.2, 1.2),  
    expand_text=(1.2, 1.2)  
)  
  
plt.show()
```



O comportamento observado sugere que com a diminuição do valor de k o modelo se torna cada vez mais propenso ao overfitting, tendo seu erro de treino diminuído e o de teste aumentado. Contudo, independente do valor de k testado, os outros modelos ainda assim apresentaram melhor desempenho no conjunto de teste.

20 Exercício 5a

O modelo que exige que os dados sejam normalizados para que cada feature tenha variância 1 é o lasso. Modelos de regressão com encolhimento demandam a normalização dos dados de modo a evitar que alguns coeficientes sejam mais penalizados pelo encolhimento do que outros simplesmente por terem maior variância ou ordem de grandeza.

21 Exercício 5b

Implementação dos algoritmos de seleção por meio de maximização de R^2 .

Best Subset Selection:



```
from itertools import combinations
from tqdm import tqdm

def best_subset_selection(X, y):

    cols = list(X.columns)
    p = len(cols)
    results = {}
    for k in tqdm(range(1, p+1)):

        best_r2 = -np.inf

        best_subset = None

        for subset in (combinations(cols,k)):

            test_subset = list(subset)

            X_sub = sm.add_constant(X[test_subset])
            model = sm.OLS(y, X_sub).fit()

            r2 = model.rsquared

            if r2 > best_r2:

                best_r2 = r2

                best_subset = test_subset

            results[k] = {
                'Subset': best_subset,
                'r2': best_r2
            }

    return results
```

Forward stepwise selection:



```
def forward_selection(X, y):

    cols = list(X.columns)
    p = len(cols)
    results = {}
    selected = []
    for k in tqdm(range(1,p+1)):

        best_r2 = -np.inf
        best_subset = None

        for col in cols:
            if col not in selected:

                subset = selected + [col]
                X_sub = sm.add_constant(X[subset])
                model = sm.OLS(y, X_sub).fit()
                r2 = model.rsquared

                if r2 > best_r2:

                    best_r2 = r2
                    best_subset = subset

                selected = best_subset
                results[k] = {
                    'subset': selected,
                    'r2': best_r2
                }
    return results
```

Backward stepwise selection:



```
def backward_selection(X, y):
    cols = list(X.columns)
    p = len(cols)
    results = {}
    selected = cols
    removed = []

    for k in tqdm(range(p, 0, -1)):
        if k == 13:
            X_sub = sm.add_constant(X)
            model = sm.OLS(y, X_sub).fit()
            r2 = model.rsquared
            print(r2)
            results[k] = {
                'subset': cols.copy(),
                'r2': r2}
        else:
            best_r2 = -np.inf
            feature_to_drop = None

            for col in cols:
                trial = selected.copy()
                trial.remove(col)
                X_sub = sm.add_constant(X[trial])
                model = sm.OLS(y, X_sub).fit()
                r2 = model.rsquared

                if r2 > best_r2:
                    best_r2 = r2
                    feature_to_drop = col

            selected.remove(feature_to_drop)
            results[k] = {
                'subset': selected.copy(),
                'r2': best_r2
            }

    return results
```

Resultados para o conjunto de treino:

```
● ● ●  
----- k = 1 -----  
Algoritmo : Best Subset Selection  
Subset escolhido: ['Abdomen']  
Valor de R^2: 0.6404157973526721  
  
Algoritmo : Foward Stepwise Selection  
Subset escolhido: ['Abdomen']  
Valor de R^2: 0.6404157973526721  
  
Algoritmo : Backward Stepwise Selection  
Subset escolhido: ['Abdomen']  
Valor de R^2: 0.6404157973526721  
  
----- k = 2 -----  
Algoritmo : Best Subset Selection  
Subset escolhido: ['Weight', 'Abdomen']  
Valor de R^2: 0.693716876511187  
  
Algoritmo : Foward Stepwise Selection  
Subset escolhido: ['Weight', 'Abdomen']  
Valor de R^2: 0.6937168765111872  
  
Algoritmo : Backward Stepwise Selection  
Subset escolhido: ['Weight', 'Abdomen']  
Valor de R^2: 0.693716876511187  
  
----- k = 3 -----  
Algoritmo : Best Subset Selection  
Subset escolhido: ['Weight', 'Abdomen', 'Wrist']  
Valor de R^2: 0.7100124087139327  
  
Algoritmo : Foward Stepwise Selection  
Subset escolhido: ['Weight', 'Abdomen', 'Wrist']  
Valor de R^2: 0.7100124087139326  
  
Algoritmo : Backward Stepwise Selection  
Subset escolhido: ['Weight', 'Abdomen', 'Wrist']  
Valor de R^2: 0.7100124087139327  
  
----- k = 4 -----  
Algoritmo : Best Subset Selection  
Subset escolhido: ['Weight', 'Abdomen', 'Biceps', 'Wrist']  
Valor de R^2: 0.7192801308495235  
  
Algoritmo : Foward Stepwise Selection  
Subset escolhido: ['Weight', 'Abdomen', 'Biceps', 'Wrist']  
Valor de R^2: 0.7192801308495234  
  
Algoritmo : Backward Stepwise Selection  
Subset escolhido: ['Weight', 'Abdomen', 'Forearm', 'Wrist']  
Valor de R^2: 0.7189909295160879
```



```
----- k = 5 -----
Algoritmo : Best Subset Selection
Subset escolhido: ['Weight', 'Abdomen', 'Biceps', 'Forearm', 'Wrist']
Valor de R^2: 0.7231554582194162

Algoritmo : Foward Stepwise Selection
Subset escolhido: ['Weight', 'Abdomen', 'Biceps', 'Forearm', 'Wrist']
Valor de R^2: 0.7231554582194162

Algoritmo : Backward Stepwise Selection
Subset escolhido: ['Weight', 'Abdomen', 'Thigh', 'Forearm', 'Wrist']
Valor de R^2: 0.7226406762641202

----- k = 6 -----
Algoritmo : Best Subset Selection
Subset escolhido: ['Weight', 'Neck', 'Abdomen', 'Biceps', 'Forearm', 'Wrist']
Valor de R^2: 0.7263015578363177

Algoritmo : Foward Stepwise Selection
Subset escolhido: ['Weight', 'Neck', 'Abdomen', 'Biceps', 'Forearm', 'Wrist']
Valor de R^2: 0.7263015578363178

Algoritmo : Backward Stepwise Selection
Subset escolhido: ['Age', 'Weight', 'Abdomen', 'Thigh', 'Forearm', 'Wrist']
Valor de R^2: 0.7257902921160901

----- k = 7 -----
Algoritmo : Best Subset Selection
Subset escolhido: ['Age', 'Weight', 'Neck', 'Abdomen', 'Thigh', 'Forearm', 'Wrist']
Valor de R^2: 0.7287047610965436

Algoritmo : Foward Stepwise Selection
Subset escolhido: ['Weight', 'Neck', 'Abdomen', 'Thigh', 'Biceps', 'Forearm', 'Wrist']
Valor de R^2: 0.728615436727488

Algoritmo : Backward Stepwise Selection
Subset escolhido: ['Age', 'Weight', 'Neck', 'Abdomen', 'Thigh', 'Forearm', 'Wrist']
Valor de R^2: 0.7287047610965436

----- k = 8 -----
Algoritmo : Best Subset Selection
Subset escolhido: ['Age', 'Weight', 'Neck', 'Abdomen', 'Thigh', 'Biceps', 'Forearm', 'Wrist']
Valor de R^2: 0.7316170437252492

Algoritmo : Foward Stepwise Selection
Subset escolhido: ['Age', 'Weight', 'Neck', 'Abdomen', 'Thigh', 'Biceps', 'Forearm', 'Wrist']
Valor de R^2: 0.7316170437252489

Algoritmo : Backward Stepwise Selection
Subset escolhido: ['Age', 'Weight', 'Neck', 'Abdomen', 'Thigh', 'Biceps', 'Forearm', 'Wrist']
Valor de R^2: 0.7316170437252492
```



```
----- k = 9 -----
Algoritmo : Best Subset Selection
Subset escolhido: ['Age', 'Weight', 'Neck', 'Abdomen', 'Hip', 'Thigh', 'Biceps', 'Forearm', 'Wrist']
Valor de R^2: 0.7336884885165282

Algoritmo : Foward Stepwise Selection
Subset escolhido: ['Age', 'Weight', 'Neck', 'Abdomen', 'Hip', 'Thigh', 'Biceps', 'Forearm', 'Wrist']
Valor de R^2: 0.7336884885165281

Algoritmo : Backward Stepwise Selection
Subset escolhido: ['Age', 'Weight', 'Neck', 'Abdomen', 'Hip', 'Thigh', 'Biceps', 'Forearm', 'Wrist']
Valor de R^2: 0.7336884885165282

----- k = 10 -----
Algoritmo : Best Subset Selection
Subset escolhido: ['Age', 'Weight', 'Neck', 'Abdomen', 'Hip', 'Thigh', 'Ankle', 'Biceps', 'Forearm', 'Wrist']
Valor de R^2: 0.7348072970162933

Algoritmo : Foward Stepwise Selection
Subset escolhido: ['Age', 'Weight', 'Neck', 'Abdomen', 'Hip', 'Thigh', 'Ankle', 'Biceps', 'Forearm', 'Wrist']
Valor de R^2: 0.7348072970162935

Algoritmo : Backward Stepwise Selection
Subset escolhido: ['Age', 'Weight', 'Neck', 'Abdomen', 'Hip', 'Thigh', 'Ankle', 'Biceps', 'Forearm', 'Wrist']
Valor de R^2: 0.7348072970162933

----- k = 11 -----
Algoritmo : Best Subset Selection
Subset escolhido: ['Age', 'Weight', 'Neck', 'Chest', 'Abdomen', 'Hip', 'Thigh', 'Ankle', 'Biceps', 'Forearm', 'Wrist']
Valor de R^2: 0.735076289045421

Algoritmo : Foward Stepwise Selection
Subset escolhido: ['Age', 'Weight', 'Neck', 'Chest', 'Abdomen', 'Hip', 'Thigh', 'Ankle', 'Biceps', 'Forearm', 'Wrist']
Valor de R^2: 0.735076289045421

Algoritmo : Backward Stepwise Selection
Subset escolhido: ['Age', 'Weight', 'Neck', 'Chest', 'Abdomen', 'Hip', 'Thigh', 'Ankle', 'Biceps', 'Forearm', 'Wrist']
Valor de R^2: 0.735076289045421

----- k = 12 -----
Algoritmo : Best Subset Selection
Subset escolhido: ['Age', 'Weight', 'Height', 'Neck', 'Chest', 'Abdomen', 'Hip', 'Thigh', 'Ankle', 'Biceps', 'Forearm', 'Wrist']
Valor de R^2: 0.7353078015250749

Algoritmo : Foward Stepwise Selection
Subset escolhido: ['Age', 'Weight', 'Height', 'Neck', 'Chest', 'Abdomen', 'Hip', 'Thigh', 'Ankle', 'Biceps', 'Forearm', 'Wrist']
Valor de R^2: 0.7353078015250749

Algoritmo : Backward Stepwise Selection
Subset escolhido: ['Age', 'Weight', 'Height', 'Neck', 'Chest', 'Abdomen', 'Hip', 'Thigh', 'Ankle', 'Biceps', 'Forearm', 'Wrist']
Valor de R^2: 0.7353078015250749

----- k = 13 -----
Algoritmo : Best Subset Selection
Subset escolhido: ['Age', 'Weight', 'Height', 'Neck', 'Chest', 'Abdomen', 'Hip', 'Thigh', 'Knee', 'Ankle', 'Biceps', 'Forearm', 'Wrist']
Valor de R^2: 0.735423471848239

Algoritmo : Foward Stepwise Selection
Subset escolhido: ['Age', 'Weight', 'Height', 'Neck', 'Chest', 'Abdomen', 'Hip', 'Thigh', 'Knee', 'Ankle', 'Biceps', 'Forearm', 'Wrist']
Valor de R^2: 0.735423471848239

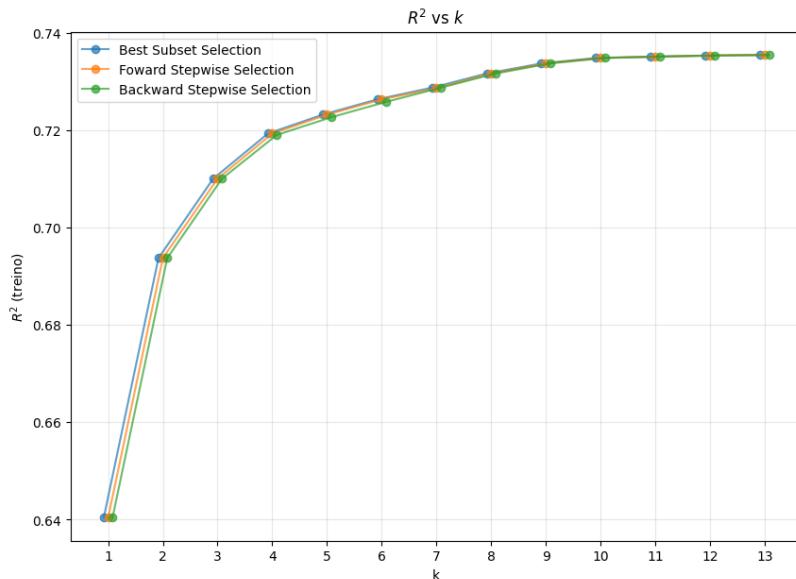
Algoritmo : Backward Stepwise Selection
Subset escolhido: ['Age', 'Weight', 'Height', 'Neck', 'Chest', 'Abdomen', 'Hip', 'Thigh', 'Knee', 'Ankle', 'Biceps', 'Forearm', 'Wrist']
Valor de R^2: 0.735423471848239
```

22 Exercício 5d

Código:

```
● ● ●  
ks = np.arange(1, len(X_train.columns) + 1)  
  
plt.figure(figsize=(10, 7))  
  
models_list = list(models.keys())  
offsets = np.linspace(-0.08, 0.08, len(models_list))  
  
for model, off in zip(models_list, offsets):  
    r2s = [models[model][k]["r2"] for k in ks]  
    plt.plot(ks + off, r2s, marker="o", label=model, alpha=0.7)  
  
plt.xlabel("K")  
plt.ylabel(r"$R^2$ (treino)")  
plt.title(r"$R^2$ vs $k$")  
plt.xticks(ks)  
plt.grid(True, alpha=0.3)  
plt.legend()  
plt.show()
```

Plot:



Tendo em vista mudanças muito pequenas nos subsets escolhidos pelos métodos, o valor de R^2 foi praticamente igual no conjunto de treino, sugerindo que alguma feature deve dominar o modelo.

23 Exercício 5d

Para executar a validação cruzada em 5 folds. O erro quadrático médio (MSE) foi calculado com a função `mean_squared_error`, disponível em `sklearn.metrics`. Ao usarmos o modelo de Lasso regression é necessário normalizar os dados de teste em cada fold, o que foi feito por meio da biblioteca `sklearn.preprocessing` por meio da classe `StandardScaler`. O MSE final para cada valor de α é a média dos MSEs obtidos em cada fold.

```
● ● ●

from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Lasso

alphas = 10**np.linspace(5, -2, 100)

mse_per_alpha = []

for a in alphas:
    mse_per_fold = []

    for i, (train_index, test_index) in enumerate(kf.split(X)):

        X_train_fold = X.iloc[train_index]
        y_train_fold = y.iloc[train_index]

        X_test_fold = X.iloc[test_index]
        y_test_fold = y.iloc[test_index]

        scaler = StandardScaler()
        X_train_fold_sc = scaler.fit_transform(X_train_fold)
        X_test_fold_sc = scaler.fit_transform(X_test_fold)

        model = Lasso(alpha=a, max_iter=100000)
        model.fit(X_train_fold_sc, y_train_fold)

        y_pred = model.predict(X_test_fold_sc)
        mse = mean_squared_error(y_test_fold, y_pred)

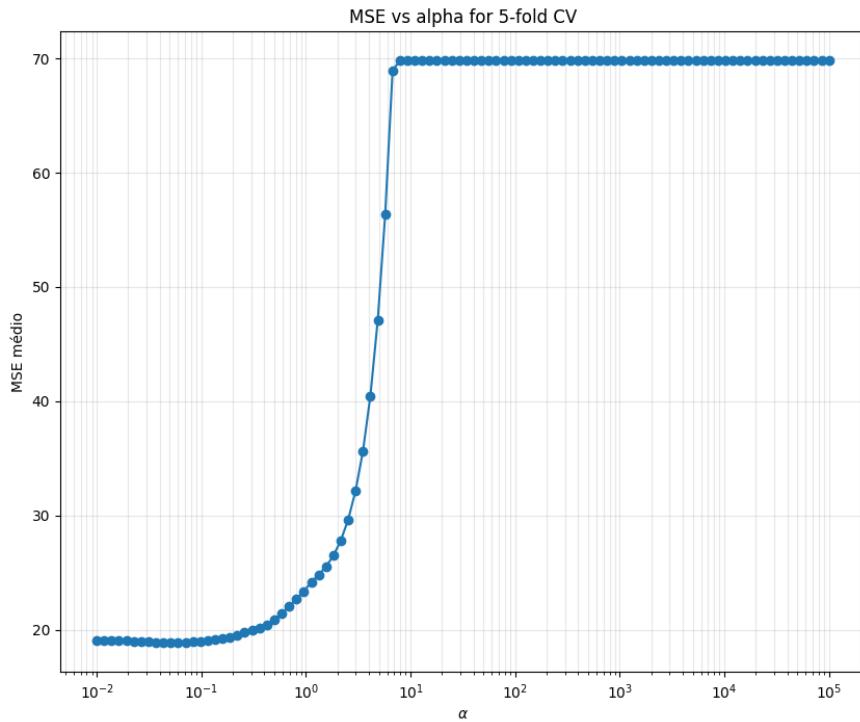
        mse_per_fold.append(mse)

    mse_per_alpha.append(np.mean(mse_per_fold))
```

A seguir podemos ver a performance do modelo de acordo com a variação de alpha.

```
● ● ●

plt.figure(figsize=(10, 8))
plt.title("MSE vs alpha for 5-fold CV")
plt.plot(alphas, mse_per_alpha, marker="o")
plt.xscale("log")
plt.xlabel(r"\alpha")
plt.ylabel("MSE médio")
plt.grid(True, which="both", alpha=0.3)
plt.show()
```



Podemos obter o valor de *alpha* que entrega o menor MSE da seguinte forma:



```
min_mse = np.min(mse_per_alpha)
min_mse_index = mse_per_alpha.index(min_mse)
min_alpha = alphas[min_mse_index]
```

Obtendo $\alpha^* = 0.05094138014816386$ com um MSE médio de 18.858386328448656.

24 Exercício 5e

O cálculo do MSE no conjunto de treino também foi feito utilizando a função `mean_squared_error`. Bastou uma pequena alteração nas funções implementadas para cada método nas questões anteriores para salvarmos também o modelo treinado para cada método e cada k.



```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
import pandas as pd
import statsmodels.api as sm
mse_test_set = {}
min_mse = np.inf
model_min_mse = None
for method in models.keys():
    if method == 'Lasso':
        scaler = StandardScaler()
        scaler.fit(X_train)
        X_test_scaled = scaler.fit_transform(X_test)
        X_test_scaled = pd.DataFrame(
            X_test_scaled,
            columns=X_test.columns,
            index=X_test.index
        )
        y_pred = models[method].predict(X_test_scaled)
        mse_test_set[method] = mean_squared_error(y_test, y_pred)

        if mean_squared_error(y_test, y_pred) < min_mse:
            min_mse = mean_squared_error(y_test, y_pred)
            model_min_mse = (method, min_mse)
    else:
        for k in range(1, len(X.columns)+1):
            subset = models[method][0][k]['subset']
            X_test_sub = sm.add_constant(X_test[subset], has_constant="add")
            y_pred = models[method][1][k].predict(X_test_sub)
            mse_test_set[(method, k)] = mean_squared_error(y_test, y_pred)

            if mean_squared_error(y_test, y_pred) < min_mse:
                min_mse = mean_squared_error(y_test, y_pred)
                model_min_mse = (method, k, subset, min_mse)
```

Resultando em:



```
{('Best Subset Selection', 1): 20.012028642731764,
 ('Best Subset Selection', 2): 14.576799544906331,
 ('Best Subset Selection', 3): 16.16242713054986,
 ('Best Subset Selection', 4): 17.08675704232918,
 ('Best Subset Selection', 5): 16.688329648854243,
 ('Best Subset Selection', 6): 16.3233358201795,
 ('Best Subset Selection', 7): 15.718843071708706,
 ('Best Subset Selection', 8): 16.10606546301081,
 ('Best Subset Selection', 9): 15.941848911364172,
 ('Best Subset Selection', 10): 15.994838849230844,
 ('Best Subset Selection', 11): 16.11823920604883,
 ('Best Subset Selection', 12): 16.008373435237633,
 ('Best Subset Selection', 13): 16.042849196277633,
 ('Foward Stepwise Selection', 1): 20.012028642731764,
 ('Foward Stepwise Selection', 2): 14.57679954490627,
 ('Foward Stepwise Selection', 3): 16.16242713054984,
 ('Foward Stepwise Selection', 4): 17.086757042329218,
 ('Foward Stepwise Selection', 5): 16.688329648854193,
 ('Foward Stepwise Selection', 6): 16.32333582017958,
 ('Foward Stepwise Selection', 7): 16.919621166847385,
 ('Foward Stepwise Selection', 8): 16.10606546301087,
 ('Foward Stepwise Selection', 9): 15.941848911364312,
 ('Foward Stepwise Selection', 10): 15.994838849230813,
 ('Foward Stepwise Selection', 11): 16.118239206048653,
 ('Foward Stepwise Selection', 12): 16.008373435237587,
 ('Foward Stepwise Selection', 13): 16.04284919627747,
 ('Backward Stepwise Selection', 1): 20.012028642731764,
 ('Backward Stepwise Selection', 2): 14.576799544906331,
 ('Backward Stepwise Selection', 3): 16.16242713054986,
 ('Backward Stepwise Selection', 4): 16.107455139013474,
 ('Backward Stepwise Selection', 5): 16.749182556536912,
 ('Backward Stepwise Selection', 6): 16.058136735168546,
 ('Backward Stepwise Selection', 7): 15.718843071708706,
 ('Backward Stepwise Selection', 8): 16.10606546301081,
 ('Backward Stepwise Selection', 9): 15.941848911364172,
 ('Backward Stepwise Selection', 10): 15.994838849230844,
 ('Backward Stepwise Selection', 11): 16.11823920604883,
 ('Backward Stepwise Selection', 12): 16.008373435237633,
 ('Backward Stepwise Selection', 13): 16.042849196277633,
 'Lasso': 15.840552653627844}
```

25 Exercício 5f

O método com o menor MSE no conjunto de teste foi:



```
('Forward Stepwise Selection','k = 2', ['Abdomen', 'Weight'], 'MSE = 14.57679954490627')
```

Esse resultado confirma nossa hipótese desenvolvida na letra b deste em exercício de que apenas algumas features dominam o modelo.

26 Referências

SCIKIT-LEARN DEVELOPERS. *scikit-learn: Machine Learning in Python — scikit-learn 1.8.0 documentation*. [S. l.], 2025. Disponível em: <https://scikit-learn.org/stable/index.html>. Acesso em: 23 jan. 2026.

JAMES, Gareth; WITTEN, Daniela; HASTIE, Trevor; TIBSHIRANI, Robert. *An Introduction to Statistical Learning: with Applications in R*. 2. ed. New York: Springer, 2021. Disponível em: <https://www.statlearning.com/>. Acesso em: 23 jan. 2026.

HASTIE, Trevor; TIBSHIRANI, Robert; FRIEDMAN, Jerome. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2. ed. New York: Springer, 2009. Disponível em: <https://hastie.su.domains/ElemStatLearn/>. Acesso em: 23 jan. 2026.