

Lista 1 de Machine Learning

Gabriel Dias Vilela

IMPA, Verão 2025

Sumário

1	Exercicio 1a	3
2	Exercicio 1b	4
3	Exercicio 1c	5
4	Exercicio 1d	6
5	Exercicio 1e	7
6	Exercicio 2a	8
7	Exercicio 2b	9
8	Exercicio 2c	11
9	Exercicio 2d	13
10	Exercicio 3a	14
11	Exercicio 3b	15
12	Exercicio 3c	16
13	Exercicio 3d	17
14	Exercicio 3e	18
15	Exercicio 3f	19
16	Exercicio 4a	21
17	Exercicio 4b	22
18	Exercicio 4c	23
19	Exercicio 4d	24

20 Exercício 5a	25
21 Exercício 5b	26
22 Exercício 5c	27
23 Exercício 5d	29
24 Exercício 6a	31
25 Exercício 6b	32
26 Exercício 6c	36
27 Exercício 6d	38
28 Exercício 6e	40
29 Exercício 6f	41

1 Exercício 1a

Falso. A diferença entre o erro de teste e o de treino, mesmo no caso em que o erro de teste é baixo, pode trazer informações úteis sobre o grau de complexidade ideal para o modelo no contexto do problema, bem como auxiliar na identificação de overfitting etc...

2 Exercício 1b

Verdadeiro. A hipótese sobre a distribuição dos erros gera como consequência uma hipótese sobre a distribuição da variável t que por sua vez é comparada a distribuição observada de t (t_{obs}) para analisar H_0 .

3 Exercício 1c

Falso. Como o banco pretende atribuir peso maior aos erros relacionados a não-identificação de transações fraudulentas, é necessário avaliar a acurácia do modelo apenas na análise de amostras fraudulentas também.

4 Exercício 1d

Verdadeiro. No caso limite, basta tomar $\lambda = 0$. Neste caso, a regressão de ridge se comportará como uma regressão linear e, portanto, terá a mesma performance.

5 Exercício 1e

Verdadeiro. Os intervalos de confiança apresentados são deduzidos tendo como hipótese que os erros são independentes, identicamente distribuídos e seguem distribuição normal $N(0, \sigma^2)$. No caso em que essa hipótese não é válida, utilizar uma distribuição gerada via bootstrap para avaliar intervalos de confiança se mostra mais vantajoso devido ao seu caráter "empírico" e à sua proximidade com a realidade dos dados.

6 Exercício 2a

Reescrevendo a variância:

$$\Sigma = V(\epsilon) = \mathbb{E} \left[(\epsilon - \mathbb{E}[\epsilon])(\epsilon - \mathbb{E}[\epsilon])^\top \right] = \mathbb{E}[B_{n \times n}]$$

Note que:

$$\epsilon - \mathbb{E}[\epsilon] = \begin{bmatrix} \epsilon_1 - \mathbb{E}[\epsilon_1] \\ \epsilon_2 - \mathbb{E}[\epsilon_2] \\ \vdots \\ \epsilon_n - \mathbb{E}[\epsilon_n] \end{bmatrix}$$

Portanto:

$$(\epsilon - \mathbb{E}[\epsilon])(\epsilon - \mathbb{E}[\epsilon])^\top = \begin{bmatrix} (\epsilon_1 - \mathbb{E}[\epsilon_1])^2 & (\epsilon_1 - \mathbb{E}[\epsilon_1])(\epsilon_2 - \mathbb{E}[\epsilon_2]) & \cdots & (\epsilon_n - \mathbb{E}[\epsilon_n])(\epsilon_1 - \mathbb{E}[\epsilon_1]) \\ (\epsilon_1 - \mathbb{E}[\epsilon_1])(\epsilon_2 - \mathbb{E}[\epsilon_2]) & (\epsilon_2 - \mathbb{E}[\epsilon_2])^2 & \cdots & (\epsilon_n - \mathbb{E}[\epsilon_n])(\epsilon_2 - \mathbb{E}[\epsilon_2]) \\ \vdots & \vdots & \ddots & \vdots \\ (\epsilon_n - \mathbb{E}[\epsilon_n])(\epsilon_1 - \mathbb{E}[\epsilon_1]) & (\epsilon_n - \mathbb{E}[\epsilon_n])(\epsilon_2 - \mathbb{E}[\epsilon_2]) & \cdots & (\epsilon_n - \mathbb{E}[\epsilon_n])^2 \end{bmatrix}$$

Analisando as hipóteses:

1. Resíduos não correlacionados:

Seja $i, j \in \{1, \dots, n\}$, $i \neq j$, e tome a expressão:

$$\mathbb{E}[(\epsilon_i - \mathbb{E}[\epsilon_i])(\epsilon_j - \mathbb{E}[\epsilon_j])] = \mathbb{E}[\epsilon_i \epsilon_j - \mathbb{E}[\epsilon_i] \epsilon_j - \mathbb{E}[\epsilon_j] \epsilon_i + \mathbb{E}[\epsilon_i] \mathbb{E}[\epsilon_j]]$$

Por hipótese, temos que $\mathbb{E}[\epsilon] = 0 \implies \mathbb{E}[\epsilon_i] = 0, \forall i \in \{1, \dots, n\}$.

Assim:

$$\mathbb{E}[(\epsilon_i - \mathbb{E}[\epsilon_i])(\epsilon_j - \mathbb{E}[\epsilon_j])] = \mathbb{E}[\epsilon_i \epsilon_j]$$

Portanto $b_{ij} = b_{ji} = (\epsilon_i - \mathbb{E}[\epsilon_i])(\epsilon_j - \mathbb{E}[\epsilon_j]) = \epsilon_i \epsilon_j$.

Logo, os elementos Σ_{ij} da matriz $\Sigma_{n \times n} = \mathbb{E}[B_{n \times n}]$, $i \neq j$, serão da forma $\mathbb{E}[\epsilon_i \epsilon_j]$.

Assim, a hipótese $\mathbb{E}[\epsilon_i \epsilon_j] = 0, \forall i \neq j$, anula todos os entradas da matriz Σ , exceto sua diagonal principal.

2. Homoscedasticidade

Seja $V(\epsilon_i) = V(\epsilon_j) = \sigma^2, \forall i, j \in \{1, \dots, n\}$.

Tendo em vista os cálculos já feitos, a diagonal principal da matriz Σ terá todos os seus elementos dados por σ^2 .

Por fim, concluímos que, dado que as hipóteses 1 e 2, Σ vale:

$$\Sigma = \sigma^2 I_{n \times n}, \quad \text{onde } V(\epsilon_i) = \sigma^2, \forall i \in \{1, \dots, n\}$$

7 Exercício 2b

(i) Seja $\hat{\beta}_\Sigma = \arg \min_\beta (y - X\beta)^\top \Sigma^{-1} (y - X\beta) = \arg \min_\beta G(\beta)$. Tomando:

$$\frac{\partial G(\beta)}{\partial \beta} = \frac{\partial}{\partial \beta} [(y - X\beta)^\top \Sigma^{-1} (y - X\beta)] = \frac{\partial}{\partial \beta} [y^\top \Sigma^{-1} y - 2y^\top \Sigma^{-1} X\beta + \beta^\top X^\top \Sigma^{-1} X\beta],$$

Concluimos que:

$$\frac{\partial G(\beta)}{\partial \beta} = -2y^\top \Sigma^{-1} X + 2\beta^\top X^\top \Sigma^{-1} X$$

Como a função é convexa (mostrado em *), para β ser tal que minimize $G(\beta)$, ele deve ser solução da seguinte equação:

$$-2y^\top \Sigma^{-1} X + 2\beta^\top X^\top \Sigma^{-1} X = 0,$$

o que implica em:

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

(*): Mostraremos que a função G é convexa analisando a sua segunda derivada:

$$\frac{\partial^2 G(\beta)}{\partial \beta^2} = \frac{\partial}{\partial \beta} [-2y^\top \Sigma^{-1} X + 2\beta^\top X^\top \Sigma^{-1} X] = 2(X^\top \Sigma^{-1} X).$$

Como $X^\top \Sigma^{-1} X$ é positiva semi-definida (mostrado em **), G é convexo e o β calculado é um ponto de mínimo global de G .

(**): Sabemos que Σ é positiva definida. Assim, $v^\top \Sigma v > 0$ para todo vetor $v \neq 0$.

Seja w um vetor qualquer e tome Σ^{-1} . Analisaremos $w^\top \Sigma^{-1} w$. Reescrevendo w como Σv , concluimos:

$$w^\top \Sigma^{-1} w = v^\top \Sigma^\top \Sigma^{-1} \Sigma v = v^\top \Sigma^\top v > 0 \implies \Sigma^{-1} \text{ é positiva definida.}$$

Tome agora

$$v^\top (X^\top \Sigma^{-1} X) v = (Xv)^\top \Sigma^{-1} (Xv) > 0 \text{ pois } \Sigma^{-1} \text{ é positiva definida.}$$

Logo,

$$X^\top \Sigma^{-1} X \text{ é positiva definida} \implies X^\top \Sigma^{-1} X \text{ é positiva semi-definida}$$

(ii) Primeiramente, suponha que X não seja uma variável aleatória neste contexto analisado. Além disso, note que β não é variável aleatória, mas sim uma matriz fixa que gera os valores amostrais isentos de erros. Assim, desenvolvendo o valor esperado de $\hat{\beta}_\Sigma$:

$$\mathbb{E}[\hat{\beta}_\Sigma] = \mathbb{E}[(X^\top \Sigma^{-1} X)^{-1} X^\top \Sigma^{-1} y] = \mathbb{E}[(X^\top \Sigma^{-1} X)^{-1} X^\top \Sigma^{-1} (X\beta + \epsilon)].$$

$$\begin{aligned}
&= \mathbb{E} \left[(X^\top \Sigma^{-1} X)^{-1} X^\top \Sigma^{-1} X \beta + (X^\top \Sigma^{-1} X)^{-1} X^\top \Sigma^{-1} \epsilon \right]. \\
&= \mathbb{E} \left[I \beta + (X^\top \Sigma^{-1} X)^{-1} X^\top \Sigma^{-1} \epsilon \right]. \\
&= \beta + \mathbb{E} \left[(X^\top \Sigma^{-1} X)^{-1} X^\top \Sigma^{-1} \epsilon \right]
\end{aligned}$$

Como X não se trata de uma variável aleatória neste contexto, temos que:

$$\mathbb{E} \left[(X^\top \Sigma^{-1} X)^{-1} X^\top \Sigma^{-1} \epsilon \right] = (X^\top \Sigma^{-1} X)^{-1} X^\top \Sigma^{-1} \mathbb{E}[\epsilon] = 0.$$

Logo:

$$\mathbb{E}[\hat{\beta}_\Sigma] = \beta.$$

(iii) Por motivos análogos ao item anterior, note que β não é variável aleatória, mas sim uma matriz fixa. Além disso, como estamos analisando a variância dado X , podemos assumi-lo como fixo também (apesar de que, no contexto geral do problema, X já é tido como fixo).

$$V[\hat{\beta}_\Sigma | X] = V \left[(X^\top \Sigma^{-1} X)^{-1} X^\top \Sigma^{-1} (X \beta + \epsilon) | X \right].$$

Seja $M = (X^\top \Sigma^{-1} X)^{-1} X^\top \Sigma^{-1}$. Assim:

$$V[\hat{\beta}_\Sigma | X] = V[\beta + M\epsilon | X] = MV[\epsilon]M^\top = M\Sigma M^\top$$

Substituindo M na expressão acima, obtemos:

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

Simplificando:

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

(iv) Tome $(X^\top \Sigma^{-1} X)^{-1} X^\top \Sigma^{-1} \epsilon = \alpha$. Note que, condicionado a X , ϵ e α seguem distribuições com a mesma natureza (ou seja, α também segue uma distribuição normal), afinal a matriz $(X^\top \Sigma^{-1} X)^{-1} X^\top \Sigma^{-1}$ não se trata de uma variável aleatória (seu valor é fixo e invariável em função de reamostragens) e portanto a forma da curva de distribuição de α é determinado pela única variável aleatória relacionada a ela, ϵ .

Seja também $\hat{\beta}_\Sigma = \alpha + \beta$. De maneira análoga a forma da curva da distribuição de $\hat{\beta}_\Sigma$ é determinada apenas por α visto que β não é uma variável aleatória. Assim, $\hat{\beta}_\Sigma$ segue uma distribuição $N(\mu, \sigma^2)$. Como já calculado, $E[\hat{\beta}_\Sigma] = \beta$ e $V[\hat{\beta}_\Sigma | X] = (X^\top \Sigma^{-1} X)^{-1}$. Logo, $\hat{\beta}_\Sigma$ segue a distribuição $N(\beta, (X^\top \Sigma^{-1} X)^{-1})$.

8 Exercício 2c

(i) Note que

$$y - X\beta = \begin{bmatrix} y_1 - f(X_1) \\ \vdots \\ y_n - f(X_n) \end{bmatrix} \quad \text{e} \quad \Sigma^{-1} = \begin{bmatrix} \frac{1}{\sigma_1^2} & 0 & \cdots & 0 \\ 0 & \frac{1}{\sigma_2^2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\sigma_n^2} \end{bmatrix}.$$

Assim,

$$(y - X\beta)^\top \Sigma^{-1} (y - X\beta) = \sum_{i=1}^n \frac{(y_i - f(X_i))^2}{\sigma_i^2} = \sum_{i=1}^n w_i^2 \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2.$$

Logo,

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

(ii) A escolha dos pesos w_i^2 diminuí o impacto de amostras com alta variância na soma dos resíduos quadrados. Assim, ele atua de modo a "normalizar" suas amostras de acordo com a variância do erro atrelado a elas.

(iii) Note que

$$\Sigma^{-\frac{1}{2}} = \text{diag}\left(\frac{1}{\sigma_1}, \frac{1}{\sigma_2}, \dots, \frac{1}{\sigma_n}\right) \quad (**):$$

Tome $(y - X\beta)^\top \Sigma^{-1} (y - X\beta)$.

Desenvolvendo:

$$(y - X\beta)^\top \Sigma^{-1} (y - X\beta) = (y^T - \beta^T X^T) \Sigma^{-1/2} \Sigma^{-1/2} (y - X\beta) = (\Sigma^{-1/2} y - \Sigma^{-1/2} X\beta)^T (\Sigma^{-1/2} y - \Sigma^{-1/2} X\beta)$$

$$= (\tilde{y} - \tilde{X}\beta)^\top (\tilde{y} - \tilde{X}\beta)$$

Calculando \tilde{y} e \tilde{X} utilizando (**):

$$\tilde{y} = \begin{bmatrix} \frac{y_1}{\sigma_1} \\ \frac{y_2}{\sigma_2} \\ \vdots \\ \frac{y_n}{\sigma_n} \end{bmatrix} \quad \text{e} \quad \tilde{X} = \begin{bmatrix} \frac{1}{\sigma_1} & \cdots & \frac{x_{1p}}{\sigma_1} \\ \frac{1}{\sigma_2} & \cdots & \frac{x_{2p}}{\sigma_2} \\ \vdots & \ddots & \vdots \\ \frac{1}{\sigma_n} & \cdots & \frac{x_{np}}{\sigma_n} \end{bmatrix}.$$

Portanto: $(\tilde{y} - \tilde{X}\beta)^\top (\tilde{y} - \tilde{X}\beta) = \sum_{i=1}^n \left(\frac{y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij}}{\sigma_i} \right)^2 = \sum_{i=1}^n \left(\tilde{y}_i - \frac{\beta_0}{\sigma_i} - \sum_{j=1}^p \beta_j \tilde{x}_{ij} \right)^2$

Onde $\tilde{y}_i = \frac{y_i}{\sigma_i}$ e $\tilde{x}_{ij} = \frac{x_{ij}}{\sigma_i}$.

Assim:

$$\hat{\beta} = \arg \min_{\beta} (\tilde{y} - \tilde{X}\beta)^\top (\tilde{y} - \tilde{X}\beta) = \arg \min_{\beta} (y - X\beta)^\top \Sigma^{-1} (y - X\beta) = \arg \min_{\beta} \sum_{i=1}^n \left(\tilde{y}_i - \frac{\beta_0}{\sigma_i} - \sum_{j=1}^p \beta_j \tilde{x}_{ij} \right)^2$$

9 Exercício 2d

(ii): Calculando beta para o conjunto de dados proposto:

```
1 beta_sigma = np.linalg.inv(X.T @ np.linalg.inv(Sigma) @ X) @ X.T @ np.linalg.inv(Sigma) @ y
2 beta_min_quad = np.linalg.inv(X.T @ X) @ X.T @ y
```

Obtivemos os seguintes coeficientes:

$$\hat{\beta}_{1s} = [-34.46344733, 6.94756948]$$

$$\hat{\beta}_{\Sigma} = [1.01885254, 0.24436202]$$

Calculando os erros:

```
1 error_sigma = np.sum((beta - beta_sigma)**2)
2 error_min_quad = np.sum((beta - beta_min_quad)**2)
```

Obtivemos:

$$\|\beta - \hat{\beta}_{1s}\|_2^2 = 1302.5135337720308$$

$$\|\beta - \hat{\beta}_{\Sigma}\|_2^2 = 0.00038720502625294963$$

(iv): Calculando a estatística Z:

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

Obtemos:

$$Z = 105.75329561203299$$

10 Exercício 3a

Sabemos que $Y_i = \beta^\top X_i + \varepsilon_i$. Como $\varepsilon_i \sim \text{Laplace}(0, b)$ e como $\beta^\top X_i$ é fixo condicionado a X_i , então $Y_i \sim \text{Laplace}(\mu, b)$.

Calculando μ :

$$\mu = \mathbb{E}[Y_i | X_i] = \mathbb{E}[\beta^\top X_i + \varepsilon_i | X_i] = \beta^\top X_i + \mathbb{E}[\varepsilon_i] = \beta^\top X_i.$$

Logo,

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

Calculando a função de verossimilhança:

$$\ell(\beta) = \prod_{i=1}^n p_{\text{Laplace}}(y_i, \beta^\top x_i, b).$$

Sabendo que $p_{\text{Laplace}}(y_i, \beta^\top x_i, b) = \frac{1}{2b} \exp\left(-\frac{|y_i - \beta^\top x_i|}{b}\right)$, temos:

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

A máxima verossimilhança será, portanto, dada por:

$$\hat{\beta} = \arg \max_{\beta} \left(\frac{1}{2b}\right)^n \exp\left(-\sum_{i=1}^n \frac{|y_i - \beta^\top x_i|}{b}\right).$$

E a log-máxima verossimilhança será:

$$\hat{\beta} = \arg \max_{\beta} \left[n \log\left(\frac{1}{2b}\right) - \sum_{i=1}^n \frac{|y_i - \beta^\top x_i|}{b} \right].$$

11 Exercício 3b

A partir da log-máxima verossimilhança, podemos sugerir como função perda:

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

Como b é fixo e constante, podemos simplificar essa função perda para:

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

Ou ainda, calculando a média:

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

12 Exercício 3c

Assumindo que os erros seguem uma distribuição $N(0, \sigma^2)$, a log-máxima verossimiliança será da forma:

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

Isto equivale a:

$$\hat{\beta} = \arg \max_{\beta} \left[n \log \left(\frac{1}{\sqrt{2\pi\sigma^2}} \right) - \frac{1}{2\sigma^2} \sum_{i=1}^n (y_i - \beta^\top x_i)^2 \right].$$

Assim, podemos definir como função perda:

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

Ou ainda, removendo os termos como fixos/constantes e calculando a média,

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

Que é o erro médio quadrático.

Comparando ambas as funções perda, notamos que a primeira é menos sensível à grandes erros e outliers pois enquanto o erro quadrático médio é função do quadrado dos resíduos, a função perda referente a distribuição de Laplace é função apenas do módulo dos resíduos. Esse comportamento é desejável tendo em vista que a distribuição de Laplace apresenta caudas pesadas.

13 Exercício 3d

Pela descida de gradiente, temos que:

$$\hat{\beta}^{(t)} = \hat{\beta}^{(t-1)} - \eta \frac{\partial L}{\partial \beta} = \hat{\beta}^{(t-1)} - \eta \frac{\partial}{\partial \beta} \left[\frac{1}{n} \sum_{i=1}^n |y_i - \beta^\top x_i| \right]$$

Aplicando a regra da cadeia:

$$\hat{\beta}^{(t)} = \hat{\beta}^{(t-1)} - \frac{\eta}{n} \sum_{i=1}^n \frac{\partial(y_i - \beta^\top x_i)}{\partial \beta} \frac{\partial |y_i - \beta^\top x_i|}{\partial (y_i - \beta^\top x_i)}$$

Logo

$$\hat{\beta}^{(t)} = \hat{\beta}^{(t-1)} + \frac{\eta}{n} \sum_{i=1}^n [x_i \operatorname{sgn}(y_i - \beta^\top x_i)]$$

14 Exercício 3e

O método da descida de gradiente para o caso gaussiano é dado por:

$$\hat{\beta}^{(t)} = \hat{\beta}^{(t-1)} + \frac{2\eta}{n} \sum_{i=1}^n x_i (y_i - \beta^\top x_i).$$

- (i) Como a predição $\beta^\top x_i$ está próxima de y_i , espera-se que $y_i - \beta^\top x_i$ seja próximo de 0, tornando o passo da descida de gradiente para o caso gaussiano baixo. Por outro lado, como o passo da descida para o caso laplaciano não depende de $y_i - \beta^\top x_i$, mas sim de $\text{sgn}(y_i - \beta^\top x_i)$, e y_i é diferente de $\beta^\top x_i$ (apesar de serem próximos), espera-se que o passo para o caso laplaciano seja maior.
- (ii) Sim. Caso $y_i > \beta^\top x_i$, o termo que determina o passo será positivo em ambas as distribuições. O análogo ocorre quando $y_i < \beta^\top x_i$, ambos os termos serão negativos. Quando $y_i = \beta^\top x_i$, ambos terão passo 0.
- (iii) Não. Como há mais de uma amostra, o passo depende do somatório de $x_i \gamma$ para cada amostra, onde $\gamma = (y_i - \beta^\top x_i)$ no caso gaussiano e $\gamma = \text{sgn}(y_i - \beta^\top x_i)$ no caso laplaciano. Assim, quanto maior a diferença entre y_i e $\beta^\top x_i$, maior será o “impacto” dessa amostra no passo do caso gaussiano. Isso não ocorre no caso laplaciano, onde o “impacto” é distribuído igualmente. Por conta disso, não somos capazes de garantir que ambos os passos serão dados na mesma direção.

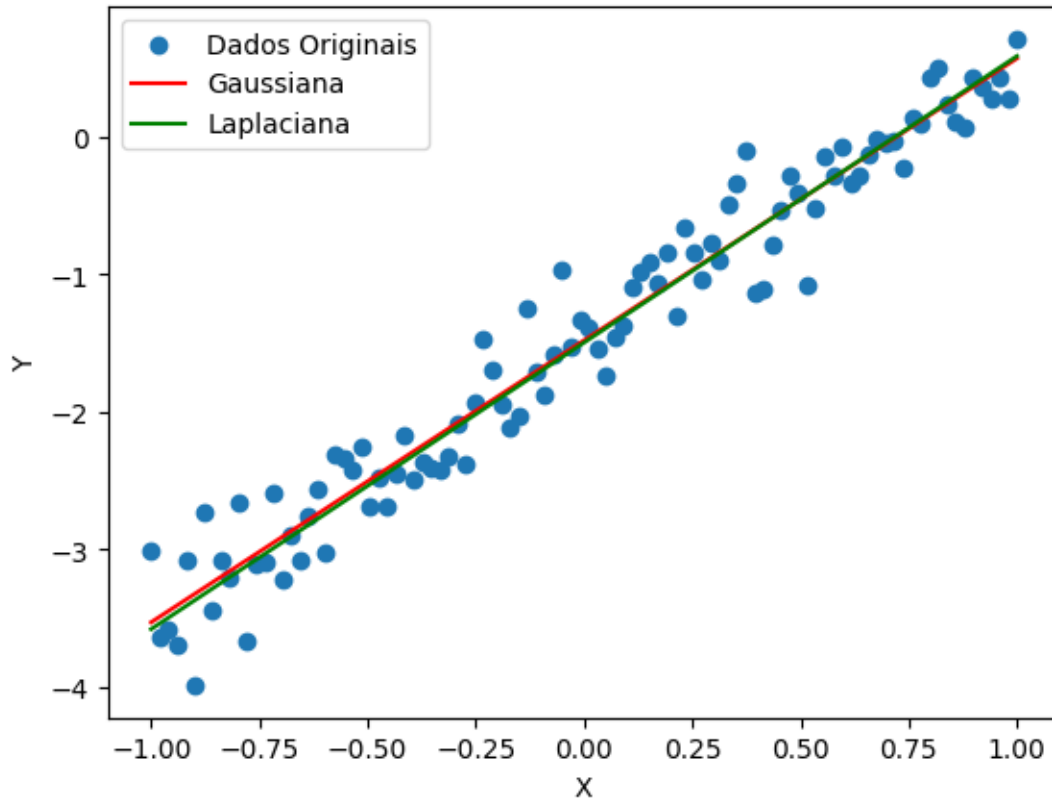
15 Exercício 3f

(i): O código completo se encontra abaixo:

```
1 # Exercício 3(f)
2
3 import numpy as np
4 from scipy.optimize import minimize
5 import matplotlib.pyplot as plt
6 from numpy import linalg
7
8 np.random.seed(1)
9
10 beta = np.array([-1.5, 2.0])
11 input_range = np.linspace(-1, 1, 100)
12 X = np.vstack([np.ones(100), input_range]).T
13 y = X @ beta + np.random.normal(0, 0.3, 100)
14 # y[80] = 10
15
16 def calculate_laplacian_loss(parameters, X, y):
17     return np.sum(np.abs(y - X @ parameters))
18
19 # Complete as linhas abaixo
20 beta_hat_gaussian = linalg.inv(X.T @ X) @ X.T @ y
21 beta_hat_laplacian = minimize(calculate_laplacian_loss, beta, args=(X, y))
```

(ii): Abaixo se encontram o código usado para gerar o gráfico e seu output:

```
1 plt.scatter(X[:, 1], y, label='Dados Originais')
2 plt.plot(X[:, 1], X @ beta_hat_gaussian, label='Gaussiana', color='red')
3 plt.plot(X[:, 1], X @ beta_hat_laplacian.x, label='Laplaciana', color='green')
4 plt.xlabel('X')
5 plt.ylabel('Y')
6 plt.legend()
7 plt.show()
```



Calculando o erro de cada um dos betas:

```
1 # Calculando o erro:
2 error_gaussian = np.sqrt(np.sum((beta - beta_hat_gaussian)**2))
3 error_laplacian = np.sqrt(np.sum((beta - beta_hat_laplacian.x)**2))
```

Obteve-se:

$$\|\beta - \hat{\beta}_{\text{gaussian}}\|_2 = 0.052850115160611374$$

$$\|\beta - \hat{\beta}_{\text{laplacian}}\|_2 = 0.08280752213622893$$

E, portanto, o modelo escolhido seria o gaussiano, por apresentar erro menor.

(iii): Recalculando os erros após a adição do outlier, obtemos o seguinte resultado:

$$\|\beta - \hat{\beta}_{\text{gaussian}}\|_2 = 0.2662249653360438$$

$$\|\beta - \hat{\beta}_{\text{laplacian}}\|_2 = 0.08553563848755102$$

Desse modo, passaríamos a escolher o modelo laplaciano que é menos sensível a outliers.

16 Exercício 4a

Insira sua solução aqui, incluindo qualquer conta, código ou figura relevante para a sua solução.

17 Exercício 4b

Insira sua solução aqui, incluindo qualquer conta, código ou figura relevante para a sua solução.

18 Exercício 4c

Insira sua solução aqui, incluindo qualquer conta, código ou figura relevante para a sua solução.

19 Exercício 4d

Insira sua solução aqui, incluindo qualquer conta, código ou figura relevante para a sua solução.

20 Exercício 5a

É necessário normalizar as features pois assim evitaremos que a escala e redimensionamentos interfiram na classificação.

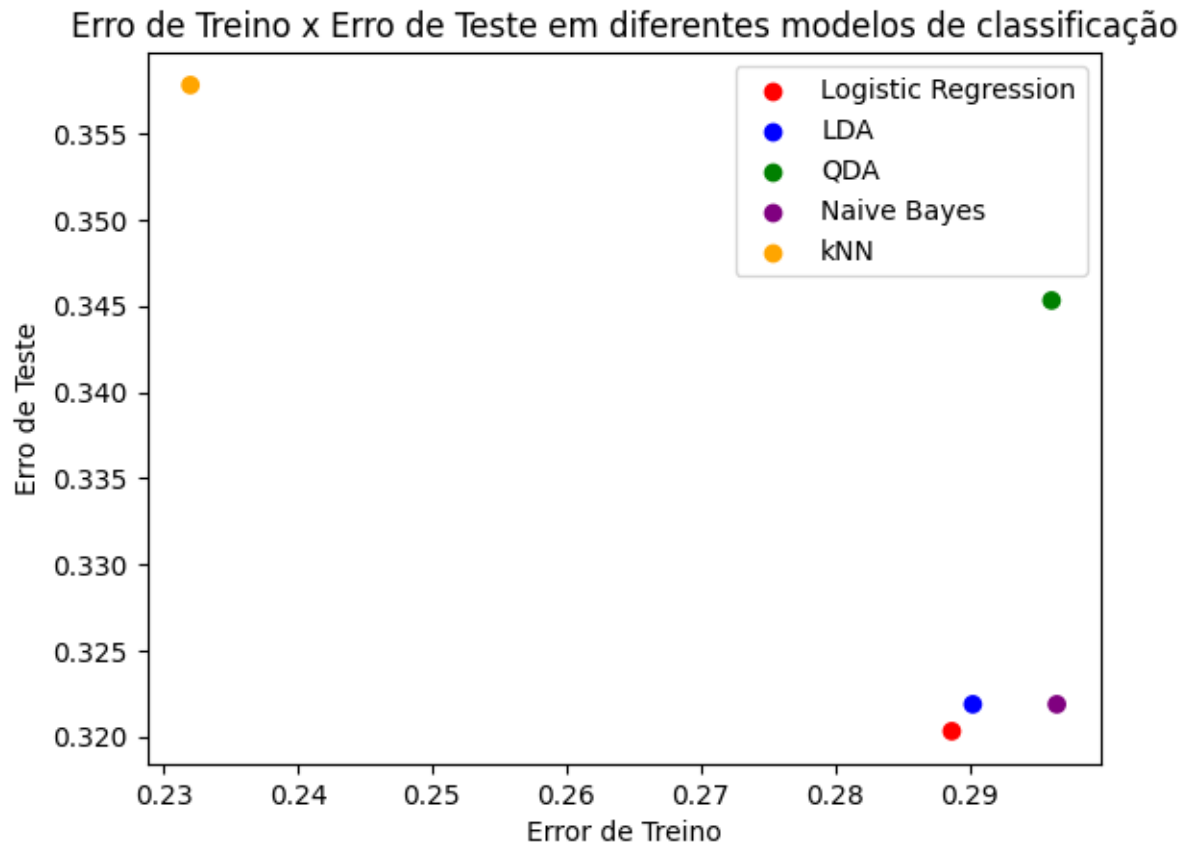
21 Exercício 5b

```
1 # Regress o Log stica
2 lr = LR()
3 fitted_lr = lr.fit(X_train, y_train.values.ravel())
4
5 y_pred_train_lr = fitted_lr.predict(X_train)
6 y_pred_test_lr = fitted_lr.predict(X_test)
7
8 # LDA
9 lda = LDA()
10 fitted_lda = lda.fit(X_train, y_train.values.ravel())
11
12 y_pred_train_lda = fitted_lda.predict(X_train)
13 y_pred_test_lda = fitted_lda.predict(X_test)
14
15 # QDA
16 qda = QDA()
17 fitted_qda = qda.fit(X_train, y_train.values.ravel())
18
19 y_pred_train_qda = fitted_qda.predict(X_train)
20 y_pred_test_qda = fitted_qda.predict(X_test)
21
22 # Naive Bayes
23 nb = NB()
24 fitted_nb = nb.fit(X_train, y_train.values.ravel())
25
26 y_pred_train_nb = fitted_nb.predict(X_train)
27 y_pred_test_nb = fitted_nb.predict(X_test)
28
29 # KNN com k=5
30 knn = kNN(n_neighbors=5)
31 fitted_knn = knn.fit(X_train, y_train.values.ravel())
32
33 y_pred_train_knn = fitted_knn.predict(X_train)
34 y_pred_test_knn = fitted_knn.predict(X_test)
```

22 Exercício 5c

Em ordem, o código utilizado e a imagem gerada a partir dele.

```
1 color_list = ['red', 'blue', 'green', 'purple', 'orange']
2
3 taxa_erro_treino_lr = (y_pred_train_lr != y_train.values.ravel()).sum() / len(
    y_train)
4 taxa_erro_teste_lr = (y_pred_test_lr != y_test.values.ravel()).sum() / len(y_test)
5
6 taxa_erro_treino_lda = (y_pred_train_lda != y_train.values.ravel()).sum() / len(
    y_train)
7 taxa_erro_teste_lda = (y_pred_test_lda != y_test.values.ravel()).sum() / len(
    y_test)
8
9 taxa_erro_treino_qda = (y_pred_train_qda != y_train.values.ravel()).sum() / len(
    y_train)
10 taxa_erro_teste_qda = (y_pred_test_qda != y_test.values.ravel()).sum() / len(
    y_test)
11
12 taxa_erro_treino_nb = (y_pred_train_nb != y_train.values.ravel()).sum() / len(
    y_train)
13 taxa_erro_teste_nb = (y_pred_test_nb != y_test.values.ravel()).sum() / len(y_test)
14
15 taxa_erro_treino_knn = (y_pred_train_knn != y_train.values.ravel()).sum() / len(
    y_train)
16 taxa_erro_teste_knn = (y_pred_test_knn != y_test.values.ravel()).sum() / len(
    y_test)
17
18 plt.scatter(
19     [taxa_erro_treino_lr, taxa_erro_treino_lda, taxa_erro_treino_qda,
20      taxa_erro_treino_nb, taxa_erro_treino_knn],
21     [taxa_erro_teste_lr, taxa_erro_teste_lda, taxa_erro_teste_qda,
22      taxa_erro_teste_nb, taxa_erro_teste_knn],
23     c=color_list
24 )
25
26 for i, model in enumerate(['Logistic Regression', 'LDA', 'QDA', 'Naive Bayes', '
    kNN']):
27     plt.scatter([], [], c=color_list[i], label=model)
28
29 plt.xlabel('Error de Treino')
30 plt.ylabel('Erro de Teste')
31 plt.title('Erro de Treino x Erro de Teste em diferentes modelos de classifica o
    ')
32 plt.legend()
33 plt.show()
```



Analisando o gráfico, percebemos que o modelo kNN está possivelmente realizando um overfitting dos dados, tendo em vista que seu erro de treino é o mais baixo dentre os modelos e seu erro de teste é o mais alto. Além disso, o modelo de Regressão Logística apresenta a melhor performance dentre os analisados, pois ele apresenta o menor erro de teste e o segundo melhor erro de treino (perdendo apenas para o kNN overfittado). Dado a diferença de performance entre o QDA e os modelos Naive Bayes, Regressão Logística e LDA, podemos supor que a fronteira que separa as classes é (ou está muito próxima de ser) linear.

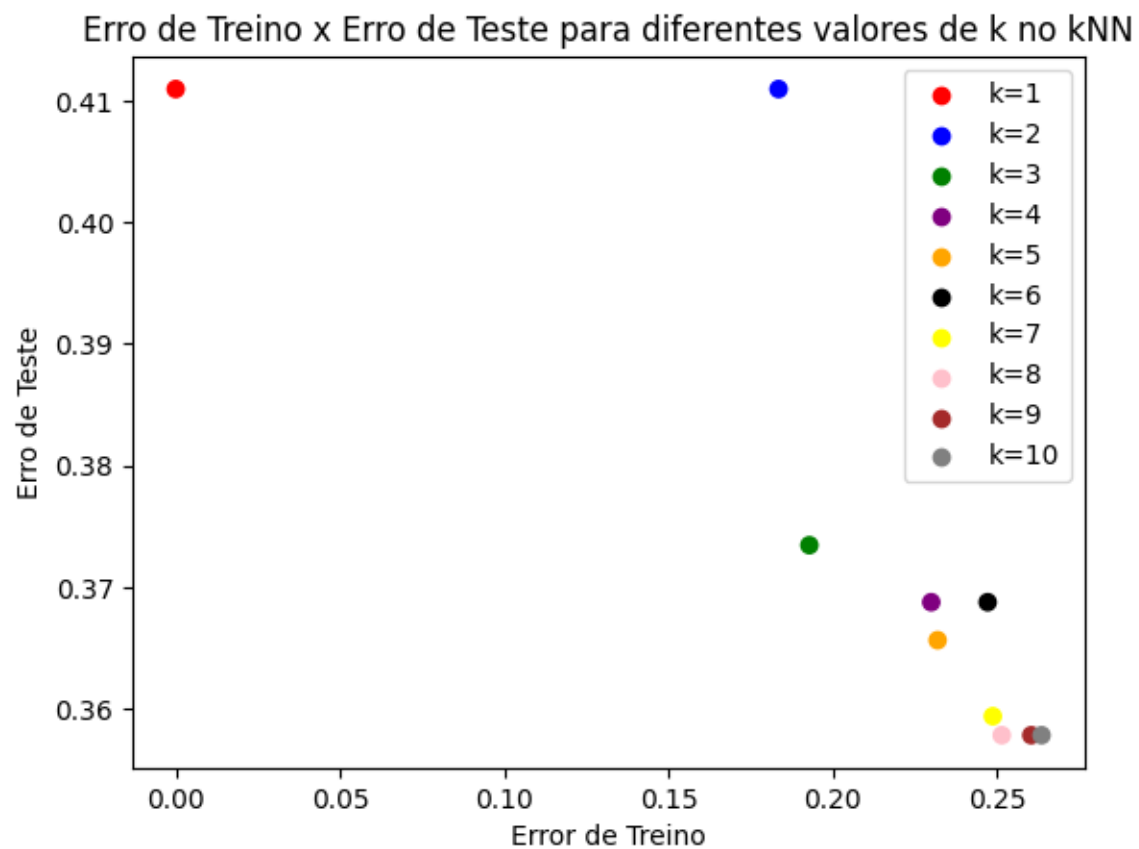
23 Exercício 5d

De acordo com o gráfico, podemos notar que o erro de treino decresce junto ao valor de k. Ou seja, modelos com k menor tendem a fittar melhor os dados de treino.

Contudo, é possível observar que a diminuição do valor de k tende a aumentar o erro de teste, o que indica que essa diminuição está gerando overfitting no modelo.

Em ordem, o código utilizado e a imagem gerada a partir dele.

```
1 ## Multiplos KNN
2
3 color_list = ['red', 'blue', 'green', 'purple', 'orange', 'black', 'yellow', 'pink',
4               ', 'brown', 'gray']
5
6 error_list = []
7 for i in range(1, 11):
8     knn = kNN(n_neighbors=i)
9
10    fitted_knn = knn.fit(X_train, y_train.values.ravel())
11
12    y_pred_treino_knn = fitted_knn.predict(X_train)
13    y_pred_test_knn = fitted_knn.predict(X_test)
14
15    error_treino = (y_pred_treino_knn != y_train.values.ravel()).sum() / len(y_train)
16    erro_teste = (y_pred_test_knn != y_test.values.ravel()).sum() / len(y_test)
17
18    error_list.append((error_treino, erro_teste))
19
20 plt.scatter(
21     x=[error[0] for error in error_list],
22     y=[error[1] for error in error_list],
23     c=color_list
24 )
25
26 for i, k in enumerate(range(1, 11)):
27     plt.scatter([], [], c=color_list[i], label=f'k={k}')
28
29 plt.xlabel('Error de Treino')
30 plt.ylabel('Erro de Teste')
31 plt.title('Erro de Treino x Erro de Teste para diferentes valores de k no kNN')
32 plt.legend()
33 plt.show()
```



24 Exercício 6a

O modelo de lasso requer normalização previa dos dados pois tal modelo penaliza os coeficientes de acordo com o quão alto são seus módulos e, portanto, pode erroneamente penalizar alguns coeficientes devido a natureza da feature associada a eles.

Por exemplo, features com o módulo da media alto (em relação as outras features do modelo) tendem a ter uma variância maior e um módulo maior. Portanto, o coeficiente associado a elas tende a ser mais baixo. Logo, tal coeficiente será pouco afetado por penalizações do modelo de lasso devido a natureza de sua feature. Para evitar isso, normalizamos essa feature subtraindo-a de sua media e dividindo-a por sua variância.

25 Exercício 6b

Abaixo se encontram os códigos utilizados para realizar a seleção a partir de cada método pedido.

Best subset selection

```
1 from itertools import combinations
2
3 def get_all_possible_subsets(n, features):
4     return list(combinations(features, n))
5
6 ## Best subset selection
7 overall_best_params_bs = None
8 overall_max_coef_bs = 0
9 best_model_by_numb_pred_bs = {}
10 for numb_pred in range(1, 14):
11
12     # Get all possible subsets and remove duplicates
13     possibilities = get_all_possible_subsets(numb_pred, X_train.columns)
14     possibilities = [tuple(sorted(possibility)) for possibility in possibilities]
15     possibilities = list(set(possibilities))
16
17     possibilit_max_coef = 0
18     for possibilit in possibilities:
19         r2_coef = 0
20         # We perform a 5-fold cross validation
21         for fold_index in range(0, 5):
22             x_fold_train = X_train.loc[cv_fold != fold_index, possibilit]
23             y_fold_train = y_train.loc[cv_fold != fold_index]
24             fitted_lm = sm.OLS(y_fold_train, x_fold_train).fit()
25
26             y_fold_validation = y_train.loc[cv_fold == fold_index]
27             x_fold_validation = X_train.loc[cv_fold == fold_index, possibilit]
28
29             y_fold_validation_pred = fitted_lm.predict(x_fold_validation)
30
31             # R2 coefficient on the validation set
32             r2_coef += 1 - np.sum((y_fold_validation - y_fold_validation_pred) **
33                                     2) / np.sum((y_fold_validation - np.mean(y_fold_validation)) ** 2)
34
35             # avg of the r2 coefficient on the 5 folds for this subset of features
36             avg_r2_coef = r2_coef / 5
37
38             # If the model with this subset of features is better than the previous
39             # best model, we save its features and update the best r2 coefficient
40             if possibilit_max_coef < avg_r2_coef:
41                 possibilit_max_coef = avg_r2_coef
42                 best_subset = [param for param in possibilit]
43
44         best_model_by_numb_pred_bs[numb_pred] = {
45             "features": best_subset,
46             "r2coef": possibilit_max_coef
47         }
48
49     # If the model with this number of features is better than the previous best
50     # model, we save its features and update the best r2 coefficient
```



```

48     if overall_max_coef_bs < possibilit_max_coef:
49         overall_max_coef_bs = possibilit_max_coef
50         overall_best_params_bs = best_subset
51
52 # Retrain the best model using all training data
53 best_bs_model = sm.OLS(y_train, X_train[overall_best_params_bs]).fit()

```

Melhor conjunto de features encontrado:

Conjunto: ['Abdomen', 'Biceps', 'Forearm', 'Height', 'Hip', 'Neck', 'Thigh', 'Weight', 'Wrist']

R^2 no conjunto de validação: 0.7039921033347978

Forward stepwise selection:

```

1 # Forward stepwise selection
2 overall_best_params_forward = None
3 overall_max_coef_forward = 0
4 best_model_by_numb_pred_forward = {}
5 previous_stage_best_features = []
6
7 for numb_pred_forward in range(1, 14):
8     available_features = set(X_train.columns) - set(previous_stage_best_features)
9     possibilit_max_coef = 0
10    for feature in available_features:
11        # We add a new feature to the previous best subset of features to test the
12        # model with this new subset
13        new_features = previous_stage_best_features + [feature]
14        r2_coef = 0
15        # We perform a 5-fold cross validation
16        for fold_index in range(0, 5):
17            x_fold_train = X_train.loc[cv_fold != fold_index, new_features]
18            y_fold_train = y_train.loc[cv_fold != fold_index]
19            fitted_lm = sm.OLS(y_fold_train, x_fold_train).fit()
20
21            y_fold_validation = y_train.loc[cv_fold == fold_index]
22            x_fold_validation = X_train.loc[cv_fold == fold_index, new_features]
23
24            y_fold_validation_pred = fitted_lm.predict(x_fold_validation)
25
26            r2_coef += 1 - np.sum((y_fold_validation - y_fold_validation_pred) **
27            2) / np.sum((y_fold_validation - np.mean(y_fold_validation)) ** 2)
28            avg_r2_coef = r2_coef / 5
29
30        # If the model with this subset of features is better than the previous
31        # best model, we save its features and update the best r2 coefficient
32        if possibilit_max_coef < avg_r2_coef:
33            possibilit_max_coef = avg_r2_coef
34            best_subset = new_features
35
36    # Save the best model for this number of features
37    best_model_by_numb_pred_forward[numb_pred_forward] = {
38        "features": best_subset,
39        "r2coef": possibilit_max_coef
40    }
41    previous_stage_best_features = best_subset

```

```

40 # Save the best overall model
41 if overall_max_coef_forward < possibilit_max_coef:
42     overall_max_coef_forward = possibilit_max_coef
43     overall_best_params_forward = best_subset
44
45 # Retrain the best model using all training data
46 best_forward_model = sm.OLS(y_train, X_train[overall_best_params_forward]).fit()

```

Melhor conjunto de features encontrado:

Conjunto: ['Abdomen', 'Wrist', 'Hip', 'Forearm', 'Neck', 'Chest', 'Height', 'Knee', 'Weight', 'Biceps', 'Thigh']

R^2 no conjunto de validação: 0.7004284501872192

Backward stepwise selection

```

1 # Backward stepwise selection
2 overall_best_params_backward = None
3 overall_max_coef_backward = 0
4 best_model_by_numb_pred_backward = {}
5 previous_stage_best_features = list(X_train.columns)
6
7
8 # Include the case where no variable is removed
9 r2_coef = 0
10 for fold_index in range(0, 5):
11     x_fold_train = X_train.loc[cv_fold != fold_index]
12     y_fold_train = y_train.loc[cv_fold != fold_index]
13     fitted_lm = sm.OLS(y_fold_train, x_fold_train).fit()
14
15     y_fold_validation = y_train.loc[cv_fold == fold_index]
16     x_fold_validation = X_train.loc[cv_fold == fold_index]
17
18     y_fold_validation_pred = fitted_lm.predict(x_fold_validation)
19
20     r2_coef += 1 - np.sum((y_fold_validation - y_fold_validation_pred) ** 2) / np.
21     sum((y_fold_validation - np.mean(y_fold_validation)) ** 2)
22
23 best_model_by_numb_pred_backward[13] = {
24     "features": list(X_train.columns),
25     "r2coef": r2_coef / 5
26 }
27
28 for numb_removed_pred_backward in range(1, 13):
29
30     available_features = set(previous_stage_best_features)
31
32     possibilit_max_coef = 0
33     for feature in available_features:
34         # We remove a feature from the previous best subset of features to test
35         # the model with this new subset
36         new_features = list(set(previous_stage_best_features) - set([feature]))
37         r2_coef = 0
38         # We perform a 5-fold cross validation
39         for fold_index in range(0, 5):
40             x_fold_train = X_train.loc[cv_fold != fold_index, new_features]

```

```

39     y_fold_train = y_train.loc[cv_fold != fold_index]
40     fitted_lm = sm.OLS(y_fold_train, x_fold_train).fit()
41
42     y_fold_validation = y_train.loc[cv_fold == fold_index]
43     x_fold_validation = X_train.loc[cv_fold == fold_index, new_features]
44
45     y_fold_validation_pred = fitted_lm.predict(x_fold_validation)
46
47     r2_coef += 1 - np.sum((y_fold_validation - y_fold_validation_pred) **
48 2) / np.sum((y_fold_validation - np.mean(y_fold_validation)) ** 2)
49     avg_r2_coef = r2_coef / 5
50
51     # If the model with this subset of features is better than the previous
52     # best model, we save its features and update the best r2 coefficient
53     if possibilit_max_coef < avg_r2_coef:
54         possibilit_max_coef = avg_r2_coef
55         best_subset = new_features
56
57     # Save the best model for this number of features
58     best_model_by_numb_pred_backward[13 - numb_removed_pred_backward] = {
59         "features": best_subset,
60         "r2coef": possibilit_max_coef
61     }
62     previous_stage_best_features = best_subset
63
64     # Save the best overall model
65     if overall_max_coef_backward < possibilit_max_coef:
66         overall_max_coef_backward = possibilit_max_coef
67         overall_best_params_backward = best_subset
68
69     # Retrain the best model using all training data
70     best_backward_model = sm.OLS(y_train, X_train[overall_best_params_backward]).fit()

```

Melhor conjunto de features encontrado:

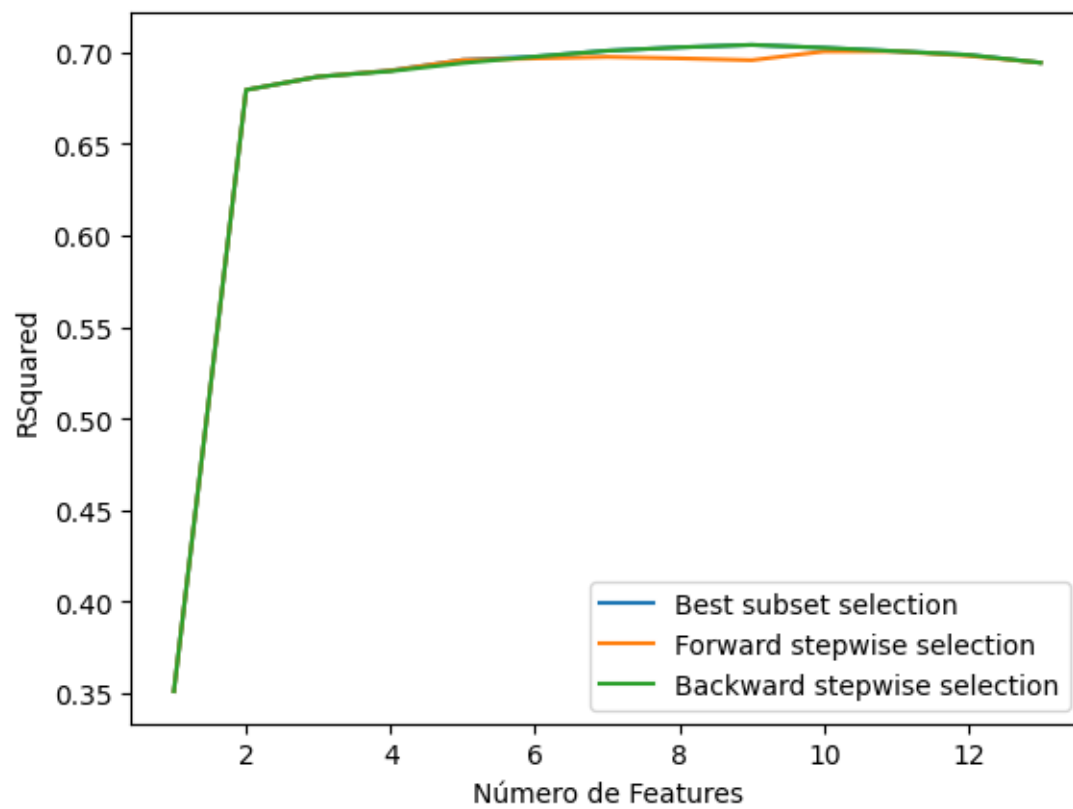
Conjunto: ['Weight', 'Wrist', 'Forearm', 'Hip', 'Neck', 'Thigh', 'Height', 'Biceps', 'Abdomen']

R^2 no conjunto de validação: 0.7039921033348016

26 Exercício 6c

A seguir, código usado e o gráfico gerado por ele:

```
1 plt.plot(  
2     list(best_model_by_numb_pred_bs.keys()),  
3     [best_model_by_numb_pred_bs[key]["r2coef"] for key in  
4     best_model_by_numb_pred_bs.keys()],  
5     label="Best subset selection"  
6 )  
7 plt.plot(  
8     list(best_model_by_numb_pred_forward.keys()),  
9     [best_model_by_numb_pred_forward[key]["r2coef"] for key in  
10    best_model_by_numb_pred_forward.keys()],  
11    label="Forward stepwise selection"  
12 )  
13 plt.plot(  
14     list(best_model_by_numb_pred_backward.keys()),  
15     [best_model_by_numb_pred_backward[key]["r2coef"] for key in  
16     best_model_by_numb_pred_backward.keys()],  
17     label="Backward stepwise selection"  
18 )  
19 plt.xlabel('N mero de Features')  
20 plt.ylabel('RSquared')  
21 plt.legend()  
22 plt.show()
```



27 Exercício 6d

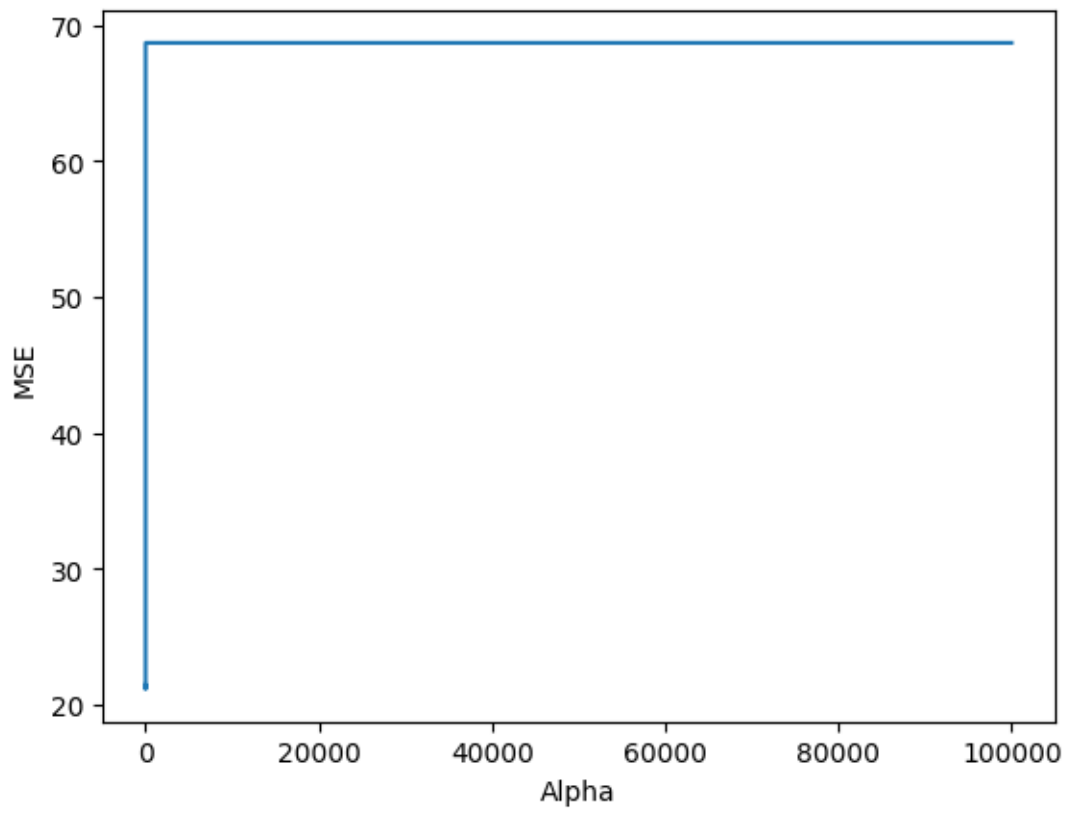
Código usado para treinar modelo de lasso:

```
1 X_train_processed = preprocessing.scale(X_train)
2
3 min_mse = np.inf
4 best_alpha = None
5 alpha_and_mse = {}
6 # Lasso for all alphas
7 for alpha in alphas:
8     lasso_model = Lasso(alpha=alpha)
9     sum_mse = 0
10    for fold_index in range(0, 5):
11        x_fold_train = X_train_processed[cv_fold != fold_index]
12        y_fold_train = y_train[cv_fold != fold_index]
13        lasso_model.fit(x_fold_train, y_fold_train)
14
15        y_fold_validation = y_train[cv_fold == fold_index]
16        x_fold_validation = X_train_processed[cv_fold == fold_index]
17        y_fold_validation_pred = lasso_model.predict(x_fold_validation)
18
19        sum_mse += np.sum((y_fold_validation - y_fold_validation_pred) ** 2) / len
20        (y_fold_validation)
21
22    avg_mse = sum_mse / 5
23    alpha_and_mse[alpha] = avg_mse
24
25    if avg_mse < min_mse:
26        min_mse = avg_mse
27        best_alpha = alpha
28
29 # Retrain the best model using all training data
30 lasso_model = Lasso(alpha=best_alpha)
31 best_lasso_model = lasso_model.fit(X_train_processed, y_train)
```

Alfa com menor erro dentre os valores avaliados: $\alpha^* = 0.031257158496882355$

Código usado para gerar gráfico, juntamente com o seu output:

```
1 plt.plot(
2     list(alpha_and_mse.keys()),
3     [alpha_and_mse[key] for key in alpha_and_mse.keys()],
4     label="Lasso"
5 )
```



28 Exercício 6e

Código usado para calcular o MSE de cada um dos melhores modelos de cada método de seleção aplicado e do modelo do lasso com menor MSE fittado anteriormente:

```
1 # MSE for lasso
2 X_test_processed = preprocessing.scale(X_test)
3 y_test_pred_lasso = best_lasso_model.predict(X_test_processed)
4 mse_test_lasso = np.sum((y_test - y_test_pred_lasso) ** 2) / len(y_test)
5
6 # MSE for best subset selection
7 X_test_best_bs = X_test[overall_best_params_bs]
8 y_test_pred_bs = best_bs_model.predict(X_test_best_bs)
9 mse_test_bs = np.sum((y_test - y_test_pred_bs) ** 2) / len(y_test)
10
11 # MSE for forward stepwise selection
12 X_test_best_forward = X_test[overall_best_params_forward]
13 y_test_pred_forward = best_forward_model.predict(X_test_best_forward)
14 mse_test_forward = np.sum((y_test - y_test_pred_forward) ** 2) / len(y_test)
15
16 # MSE for backward stepwise selection
17 X_test_best_backward = X_test[overall_best_params_backward]
18 y_test_pred_backward = best_backward_model.predict(X_test_best_backward)
19 mse_test_backward = np.sum((y_test - y_test_pred_backward) ** 2) / len(y_test)
20
21 print(
22     f"MSE - Lasso: {mse_test_lasso}",
23     f"MSE - Best subset selection: {mse_test_bs}",
24     f"MSE - Forward stepwise selection: {mse_test_forward}",
25     f"MSE - Backward stepwise selection: {mse_test_backward}"
26 )
```

Output: MSE - Lasso: 16.062248033733823
MSE - Best subset selection: 16.366309346853466
MSE - Forward stepwise selection: 16.704696595152456
MSE - Backward stepwise selection: 16.366309346853328

29 Exercício 6f

Como averiguado, o modelo com melhor resultado se trata do Lasso com $\alpha = 0.031257158496882355$ e parâmetros (pós fit nos dados de treino) iguais a:

[0.62821524, -2.59442678, -0.20642187, -0.9040139, -0.23080468, 10.04281614, -1.30107934, 1.24593471, 0.02149373, 0.23701975, 0.68679073, 0.78870312, -1.78343953]

Para as features:

[Age, Weight, Height, Neck, Chest, Abdomen, Hip, Thigh, Knee, Ankle, Biceps, Forearm, Wrist]