Lista 1 de Machine Learning

Gabriel Dias Vilela

IMPA, Verão 2025

Sumário

1 Exercicio 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 Exercicio 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 Exercicio 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 analise de amostras fraudulentas também.

4 Exercicio 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 Exercicio 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 Exercicio 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 $i, j \in \{1, \ldots, n\}$, $i \neq j$, e tome a expressão:

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

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

   Assim:

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

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

   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:

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

7 Exercicio 2b

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

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

Concluímos 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:

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

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

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

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, concluímos:

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

Tome agora

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

Logo,

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

(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 $\hat{\beta}_\Sigma$:

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

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

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

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

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 beta não é variável aleatória, mas sim uma matriz fixa. Além disso, como estamos analisando a variância dado X, podemos assumi-lo como fixo também.

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

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

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

Substituindo M na expressão acima, obtemos:

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

Simplificando:

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

(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 alfa 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 $V[\hat{\beta}_Z \mid X] = (X^\top\Sigma^{-1}X)^{-1}$. Logo, $\hat{\beta}_\Sigma$ segue a distribuição $N(\beta, (X^\top\Sigma^{-1}X)^{-1})$

8 Exercicio 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,

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

Logo,

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

(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

$$\Sigma^{-\frac{1}{2}} \;=\; \operatorname{diag}\big(\tfrac{1}{\sigma_1}, \tfrac{1}{\sigma_2}, \ldots, \tfrac{1}{\sigma_n}\big) \quad (***):$$

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

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

Assim:

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

9 Exercicio 2d

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

```
beta_sigma = np.linalg.inv(X.T @ np.linalg.inv(Sigma) @ X) @ X.T @ np.linalg.inv(
    Sigma) @ y
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:

```
error_sigma = np.sum((beta - beta_sigma)**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:

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

Obtemos:

$$Z = 105.75329561203299$$

10 Exercicio 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:

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

Logo,

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

Calculando a função de verossimilhança:

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

Sabendo que $p_{Laplace}\big(y_i,\, \beta^\top x_i,\, b\big) = \dfrac{1}{2\,b}\exp\Big(-\dfrac{|\,y_i - \beta^\top x_i\,|}{b}\Big)$, temos:

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

A máxima verossimilhança será, portanto, dada por:

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

E a log-máxima verossimilhança será:

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

11 Exercicio 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, 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 Exercicio 3c

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

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

Isto equivale a:

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

Assim, podemos definir como função perda:

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

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

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

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 Exercicio 3d

Pela descida de gradiente, temos que:

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

Aplicando a regra da cadeia:

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

Logo

$$\hat{\beta}^{(t)} \;=\; =\; \hat{\beta}^{(t-1)} \;+\; \frac{\eta}{n}\sum_{i=1}^{n}\big[\; x_i \,\mathrm{sgn}(y_i - \beta^T x_i)\big]$$

14 Exercicio 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.

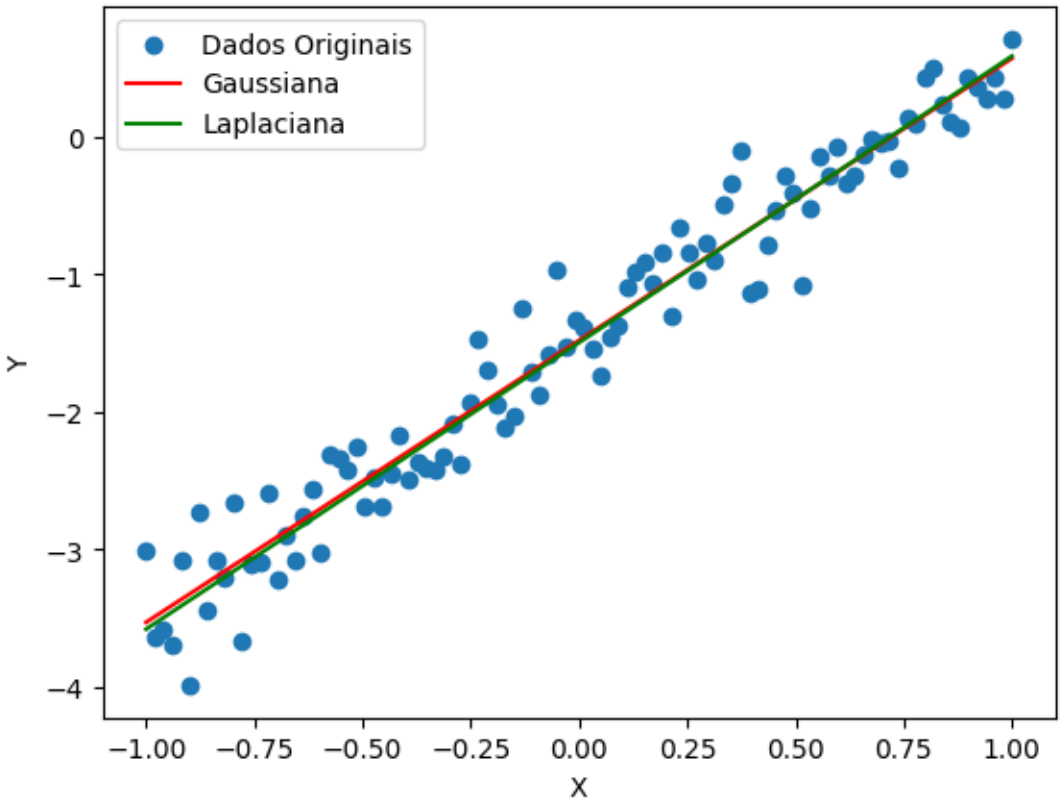
15 Exercicio 3f

(i): O código completo se encontra abaixo:

```
# Exerc cio 3(f)

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

np.random.seed(1)

beta = np.array([-1.5, 2.0])
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)
# y[80] = 10

def calculate_laplacian_loss(parameters, X, y):
    return np.sum(np.abs(y - X @ parameters))

# Complete as linhas abaixo
beta_hat_gaussian = linalg.inv(X.T @ X) @ X.T @ y
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:

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



Calculando o erro de cada um dos betas:

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

Obteve-se:

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

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

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

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

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

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

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

16 Exercicio 4a

(i) Seja:

$$E\big[\nabla_\beta \ln(p)\big] = E\left[\sum_{i=1}^{n}\big(y_i - p_\beta(x_i)\big)\, x_i\right].$$

Como x_i é fixo para todo $i \in \{1, \ldots, n\}$:

$$E\big[\nabla_\beta \ln(p)\big] = E\left[\sum_{i=1}^{n}\big(y_i - p_\beta(x_i)\big)\right] x_i$$

Como $Y_i \sim \text{Bern}(p_\beta(x_i))$, então $E[y_i] = p_\beta(x_i)$. Note também que $p_\beta(x_i)$ é fixo, logo $E[p_\beta(x_i)] = p_\beta(x_i)$. Além disso, como $E[\cdot]$ é um operador linear, podemos "permutá-lo" com o somatório. Portanto:

$$E\big[\nabla_\beta \ln(p)\big] = \sum_{i=1}^{n} E\big[y_i - p_\beta(x_i)\big]\, x_i = \sum_{i=1}^{n}\Big(E[y_i] - p_\beta(x_i)\Big)\, x_i = \sum_{i=1}^{n}\big(p_\beta(x_i) - p_\beta(x_i)\big)\, x_i = 0.$$

(ii) Seja:

$$E\big[\nabla^2_\beta \ln(p)\big] = E\left[-\sum_{i=1}^{n} x_i\, x_i^T\, p_\beta(x_i)\,\big(1 - p_\beta(x_i)\big)\right].$$

Note que x_i e $p_\beta(x_i)$ são fixos para todo $i \in \{1, \ldots, n\}$. Assim:

$$E\big[\nabla^2_\beta \ln(p)\big] = -\sum_{i=1}^{n} x_i\, x_i^T\; p_\beta(x_i)\,\big(1 - p_\beta(x_i)\big) = \nabla^2_\beta \ln(p).$$

(iii) Seja:

$$V\big(\nabla_\beta \ln(p)\big) = E\big[\big(\nabla_\beta \ln(p)\big)\,\big(\nabla_\beta \ln(p)\big)^T\big] - E\big[\nabla_\beta \ln(p)\big]\, E\big[\nabla_\beta \ln(p)\big]^T$$

Como já mostrado,

$$E\big[\nabla_\beta \ln(p)\big] = 0.$$

Logo:

$$V\big(\nabla_\beta \ln(p)\big) = E\Big[\big(\nabla_\beta \ln(p)\big)\,\big(\nabla_\beta \ln(p)\big)^T\Big].$$

Escrevendo explicitamente:

$$V\big(\nabla_\beta \ln(p)\big) = E\left[\Big(\sum_{i=1}^{n}\big(y_i - p_\beta(x_i)\big)\, x_i\Big)\Big(\sum_{j=1}^{n}\big(y_j - p_\beta(x_j)\big)\, x_j\Big)^T\right].$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n} E\Big[\big(y_i - p_\beta(x_i)\big)\,\big(y_j - p_\beta(x_j)\big)\Big]\, x_i\, x_j^T.$$

Como $\{(y_i, x_i)\}_{i=1,\ldots,n}$ são i.i.d.,

$$E\Big[\big(y_i - p_\beta(x_i)\big)\big(y_j - p_\beta(x_j)\big)\Big] \neq 0 \quad \text{apenas quando } i = j.$$

Logo:

$$V(\nabla_\beta \ln(p)) = \sum_{i=1}^{n} E\big[\big(y_i - p_\beta(x_i)\big)^2\big]\; x_i\, x_i^T$$

Como $Y_i \sim \text{Bern}\big(p_\beta(x_i)\big)$:

$$E\big[\big(y_i - p_\beta(x_i)\big)^2\big] = V\big[\, y_i \,\big] = p_\beta(x_i)\,\big(1 - p_\beta(x_i)\big).$$

$$\implies V(\nabla_\beta \ln(p)) = \sum_{i=1}^{n} p_\beta(x_i)\,\big(1 - p_\beta(x_i)\big)\; x_i\, x_i^T = -\,\nabla_\beta^2 \ln(p).$$

17 Exercicio 4e

Código utilizado para calcular a estatística pedida:

```
# Fitting the logistic regression model
lr_model = LogisticRegression(random_state=0).fit(X, y)

logits_beta_hat = 1/(1+np.exp(-X@lr_model.coef_[0]))

# Fischer information matrix
fischer_info = np.zeros((p,p))
for i, sample in enumerate(X):
    fischer_info += np.outer(sample, sample) * logits_beta_hat[i] * (1 -
    logits_beta_hat[i])

fischer_info_inverse = np.linalg.inv(fischer_info)

# Calculating a statistic to make hypothesis testing
stats = (lr_model.coef_[0][1] - 1)/np.sqrt(np.abs(fischer_info_inverse[1,1]))
```

Valor da estatística encontrado: 0.37038849246557315
Coeficientes estimados pelo modelo: [1.02467568, 1.03721328, 1.18159829, 1.17893451, 1.00181261]

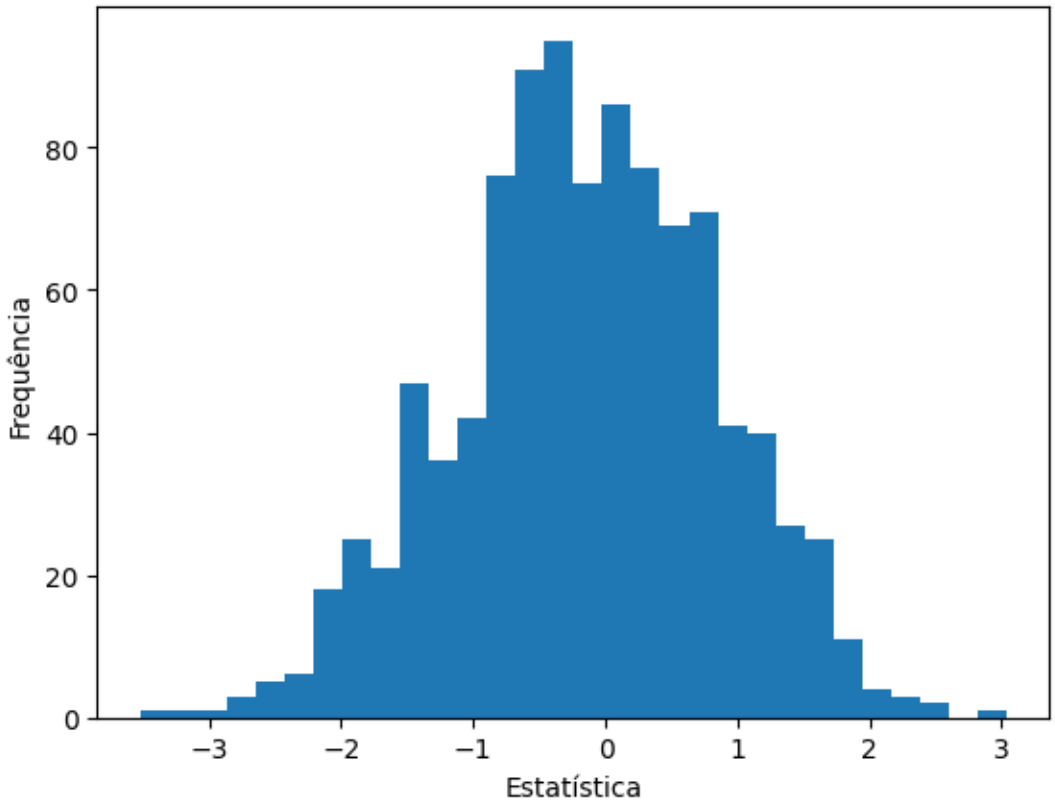
18 Exercicio 4f

Código utilizado para realizar as 1000 simulações:

```
np.random.seed(10)
n_simulations = 1000
stats_list = []
beta_hat_list = []
beta_list = []
X_list = []
y_list = []
for simulation in range(n_simulations):
    beta = np.ones(p)
    X = np.random.normal(loc=0,scale=1,size=(n,p))

    beta = 1/(1+np.exp(-X@beta))
    y = np.random.binomial(n=np.ones(n).astype(int),p=beta, size = n)

    lr_model = LogisticRegression(random_state=0).fit(X, y)

    logits_beta_hat = 1/(1+np.exp(-X@lr_model.coef_[0]))

    fischer_info = np.zeros((p,p))
    for i, sample in enumerate(X):
        fischer_info += np.outer(sample, sample) * logits_beta_hat[i] * (1 -
   logits_beta_hat[i])

    fischer_info_inverse = np.linalg.inv(fischer_info)
    stats = (lr_model.coef_[0][1] - 1)/np.sqrt(fischer_info_inverse[1,1])

    # Saving the results and the used data
    stats_list.append(stats)
    beta_hat_list.append(lr_model.coef_[0])
    beta_list.append(beta)
    X_list.append(X)
    y_list.append(y)
```

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

```
plt.hist(stats_list, bins=30)
plt.ylabel('Frequ ncia')
plt.xlabel('Estat stica')
plt.show()
```



Assim, podemos notar que a distribuição apresenta alto grau de assímetria em relação a média, se distânciando bastante do comportamento normal inicialmente suposto.

19 Exercicio 4e

Código utilizado para comparar a matriz

```
# Inverse of fisher information matrix using beta
fischer_info_beta = np.zeros((p,p))
print(beta_list)
for X, beta in zip(X_list, beta_list):
    print(X)
    logits_beta = 1/(1+np.exp(-X@beta))
    for i, sample in enumerate(X):
        fischer_info_beta += np.outer(sample, sample) * logits_beta[i] * (1 - logits_beta[i])


fischer_info_beta_inverse = np.linalg.inv(fischer_info_beta)
avg_fischer_info_beta_inverse = fischer_info_beta_inverse/n_simulations

# Inverse of fisher information matrix using beta_hat
fischer_info_beta_hat = np.zeros((p,p))
for X, logits in zip(X_list, beta_hat_list):
    logits_beta_hat = 1/(1+np.exp(-X@logits))
    for i, sample in enumerate(X):
        fischer_info_beta_hat += np.outer(sample, sample) * logits_beta_hat[i] * (1 - logits_beta_hat[i])

fischer_info_beta_hat_inverse = np.linalg.inv(fischer_info_beta_hat)
avg_fischer_info_beta_hat_inverse = fischer_info_beta_hat_inverse/n_simulations
```

20 Exercicio 5a

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

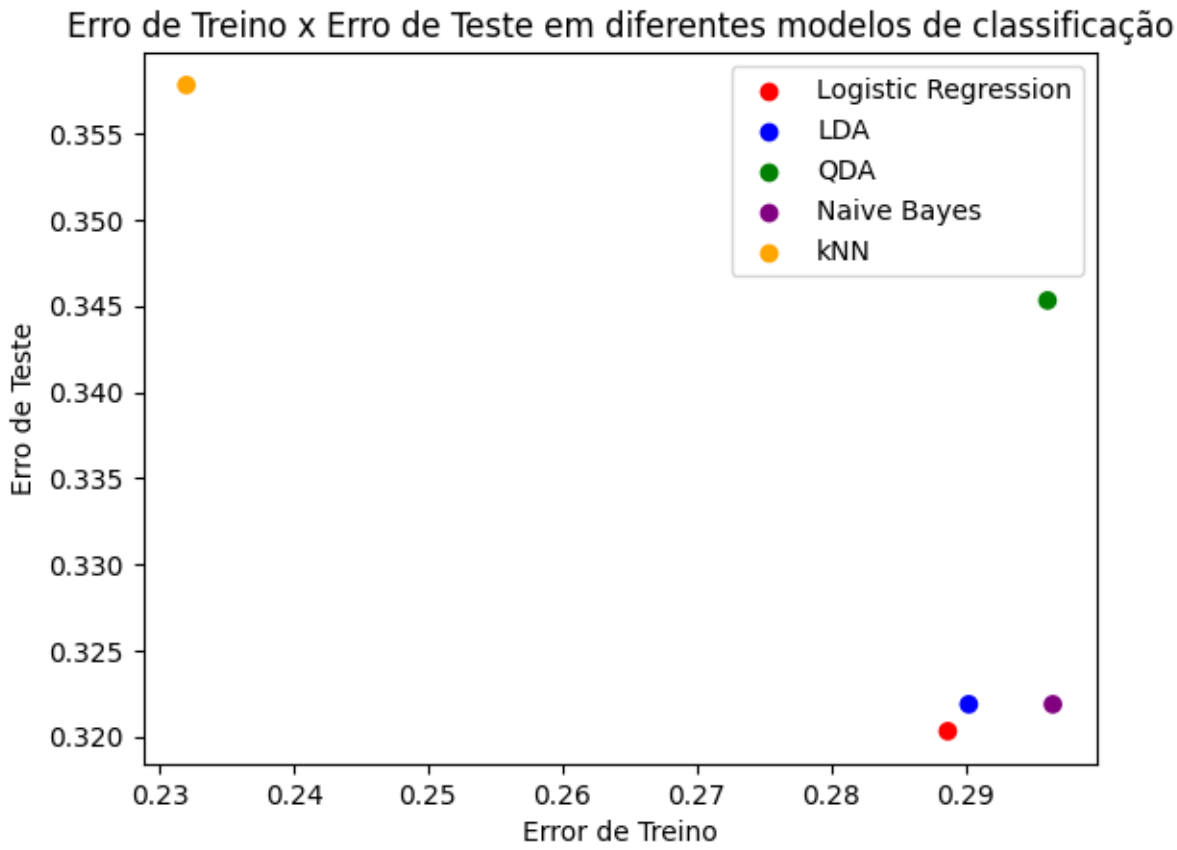
21 Exercicio 5b

```
# Regress o Log stica
lr = LR()
fitted_lr = lr.fit(X_train, y_train.values.ravel())

y_pred_train_lr = fitted_lr.predict(X_train)
y_pred_test_lr = fitted_lr.predict(X_test)

# LDA
lda = LDA()
fitted_lda = lda.fit(X_train, y_train.values.ravel())

y_pred_train_lda = fitted_lda.predict(X_train)
y_pred_test_lda = fitted_lda.predict(X_test)

# QDA
qda = QDA()
fitted_qda = qda.fit(X_train, y_train.values.ravel())

y_pred_train_qda = fitted_qda.predict(X_train)
y_pred_test_qda = fitted_qda.predict(X_test)

# Naive Bayes
nb = NB()
fitted_nb = nb.fit(X_train, y_train.values.ravel())

y_pred_train_nb = fitted_nb.predict(X_train)
y_pred_test_nb = fitted_nb.predict(X_test)

# KNN com k=5
knn = kNN(n_neighbors=5)
fitted_knn = knn.fit(X_train, y_train.values.ravel())

y_pred_train_knn = fitted_knn.predict(X_train)
y_pred_test_knn = fitted_knn.predict(X_test)
```

22 Exercicio 5c

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

```
color_list = ['red', 'blue', 'green', 'purple', 'orange']

taxa_erro_treino_lr = (y_pred_train_lr != y_train.values.ravel()).sum() / len(
    y_train)
taxa_erro_teste_lr = (y_pred_test_lr != y_test.values.ravel()).sum() / len(y_test)

taxa_erro_treino_lda = (y_pred_train_lda != y_train.values.ravel()).sum() / len(
    y_train)
taxa_erro_teste_lda = (y_pred_test_lda != y_test.values.ravel()).sum() / len(
    y_test)

taxa_erro_treino_qda = (y_pred_train_qda != y_train.values.ravel()).sum() / len(
    y_train)
taxa_erro_teste_qda = (y_pred_test_qda != y_test.values.ravel()).sum() / len(
    y_test)

taxa_erro_treino_nb = (y_pred_train_nb != y_train.values.ravel()).sum() / len(
    y_train)
taxa_erro_teste_nb = (y_pred_test_nb != y_test.values.ravel()).sum() / len(y_test)

taxa_erro_treino_knn = (y_pred_train_knn != y_train.values.ravel()).sum() / len(
    y_train)
taxa_erro_teste_knn = (y_pred_test_knn != y_test.values.ravel()).sum() / len(
    y_test)

plt.scatter(
    [taxa_erro_treino_lr, taxa_erro_treino_lda, taxa_erro_treino_qda,
    taxa_erro_treino_nb, taxa_erro_treino_knn],
    [taxa_erro_teste_lr, taxa_erro_teste_lda, taxa_erro_teste_qda,
    taxa_erro_teste_nb, taxa_erro_teste_knn],
    c=color_list
)
for i, model in enumerate(['Logistic Regression', 'LDA', 'QDA', 'Naive Bayes', '
    kNN']):
    plt.scatter([], [], c=color_list[i], label=model)

plt.xlabel('Error de Treino')
plt.ylabel('Erro de Teste')
plt.title('Erro de Treino x Erro de Teste em diferentes modelos de classifica  o
    ')
plt.legend()
plt.show()
```



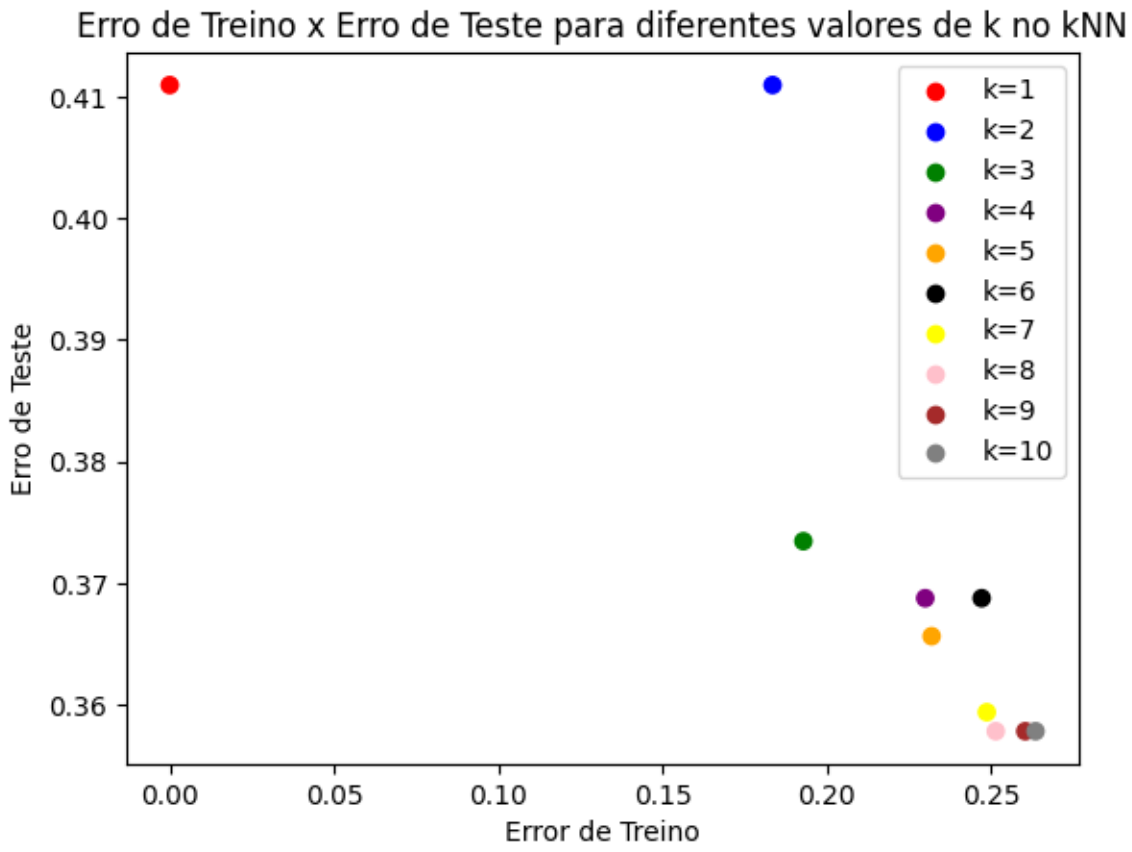
Analisando o gráfico, percebemos que o modelo kNN está possívelmente realizando um overfitting dos dados, tendo em vista que seu erro de treino é o mais baixo dentre os modelos e seu erro de teste é o mais alto. Além disso, o modelo de Regressão Logística apresenta a melhor performance dentre os analisados, 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 Exercicio 5d

De acordo com o grafico, 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.

```
## Multiplos KNN

color_list = ['red', 'blue', 'green', 'purple', 'orange', 'black', 'yellow', 'pink', 'brown', 'gray']

error_list = []
for i in range(1, 11):
    knn = kNN(n_neighbors=i)

    fitted_knn = knn.fit(X_train, y_train.values.ravel())

    y_pred_treino_knn = fitted_knn.predict(X_train)
    y_pred_test_knn = fitted_knn.predict(X_test)

    error_treino = (y_pred_treino_knn != y_train.values.ravel()).sum() / len(y_train)
    erro_teste = (y_pred_test_knn != y_test.values.ravel()).sum() / len(y_test)

    error_list.append((error_treino, erro_teste))

plt.scatter(
    x=[error[0] for error in error_list],
    y=[error[1] for error in error_list],
    c=color_list
)
for i, k in enumerate(range(1, 11)):
    plt.scatter([], [], c=color_list[i], label=f'k={k}')

plt.xlabel('Error de Treino')
plt.ylabel('Erro de Teste')
plt.title('Erro de Treino x Erro de Teste para diferentes valores de k no kNN')
plt.legend()
plt.show()
```


Erro de Treino x Erro de Teste para diferentes valores de k no kNN
Erro de Teste
Error de Treino
0.41
0.40
0.39
0.38
0.37
0.36
0.00
0.05
0.10
0.15
0.20
0.25
k=1
k=2
k=3
k=4
k=5
k=6
k=7
k=8
k=9
k=10

24 Exercicio 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 Exercicio 6b

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

Best subset selection

```
from itertools import combinations

def get_all_possible_subsets(n, features):
    return list(combinations(features, n))

## Best subset selection
overall_best_params_bs = None
overall_max_coef_bs = 0
best_model_by_numb_pred_bs = {}
for numb_pred in range(1, 14):

    # Get all possible subsets and remove duplicates
    possibilities = get_all_possible_subsets(numb_pred, X_train.columns)
    possibilities = [tuple(sorted(possibility)) for possibility in possibilities]
    possibilities = list(set(possibilities))

    possibilit_max_coef = 0
    for possibilit in possibilities:
        r2_coef = 0
        # We perform a 5-fold cross validation
        for fold_index in range(0, 5):
            x_fold_train = X_train.loc[cv_fold != fold_index, possibilit]
            y_fold_train = y_train.loc[cv_fold != fold_index]
            fitted_lm = sm.OLS(y_fold_train, x_fold_train).fit()

            y_fold_validation = y_train.loc[cv_fold == fold_index]
            x_fold_validation = X_train.loc[cv_fold == fold_index, possibilit]

            y_fold_validation_pred = fitted_lm.predict(x_fold_validation)

            # R2 coefficient on the validation set
            r2_coef += 1 - np.sum((y_fold_validation - y_fold_validation_pred) **
    2) / np.sum((y_fold_validation - np.mean(y_fold_validation)) ** 2)

        # avg of the r2 coefficient on the 5 folds for this subset of features
        avg_r2_coef = r2_coef / 5

        # If the model with this subset of features is better than the previous
    best model, we save its features and update the best r2 coefficient
        if possibilit_max_coef < avg_r2_coef:
            possibilit_max_coef = avg_r2_coef
            best_subset = [param for param in possibilit]

    best_model_by_numb_pred_bs[numb_pred] = {
        "features": best_subset,
        "r2coef": possibilit_max_coef
    }

    # If the model with this number of features is better than the previous best
    model, we save its features and update the best r2 coefficient
```

```
    if overall_max_coef_bs < possibilit_max_coef:
        overall_max_coef_bs = possibilit_max_coef
        overall_best_params_bs = best_subset

# Retrain the best model using all training data
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:

```
# Forward stepwise selection
overall_best_params_forward = None
overall_max_coef_forward = 0
best_model_by_numb_pred_forward = {}
previous_stage_best_features = []

for numb_pred_forward in range(1, 14):
    available_features = set(X_train.columns) - set(previous_stage_best_features)
    possibilit_max_coef = 0
    for feature in available_features:
        # We add a new feature to the previous best subset of features to test the model with this new subset
        new_features = previous_stage_best_features + [feature]
        r2_coef = 0
        # We perform a 5-fold cross validation
        for fold_index in range(0, 5):
            x_fold_train = X_train.loc[cv_fold != fold_index, new_features]
            y_fold_train = y_train.loc[cv_fold != fold_index]
            fitted_lm = sm.OLS(y_fold_train, x_fold_train).fit()

            y_fold_validation = y_train.loc[cv_fold == fold_index]
            x_fold_validation = X_train.loc[cv_fold == fold_index, new_features]

            y_fold_validation_pred = fitted_lm.predict(x_fold_validation)

            r2_coef += 1 - np.sum((y_fold_validation - y_fold_validation_pred) ** 2) / np.sum((y_fold_validation - np.mean(y_fold_validation)) ** 2)
        avg_r2_coef = r2_coef / 5

        # If the model with this subset of features is better than the previous best model, we save its features and update the best r2 coefficient
        if possibilit_max_coef < avg_r2_coef:
            possibilit_max_coef = avg_r2_coef
            best_subset = new_features

    # Save the best model for this number of features
    best_model_by_numb_pred_forward[numb_pred_forward] = {
        "features": best_subset,
        "r2coef": possibilit_max_coef
    }
    previous_stage_best_features = best_subset

```

```
    # Save the best overall model
    if overall_max_coef_forward < possibilit_max_coef:
        overall_max_coef_forward = possibilit_max_coef
        overall_best_params_forward = best_subset

# Retrain the best model using all training data
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

```
# Backward stepwise selection
overall_best_params_backward = None
overall_max_coef_backward = 0
best_model_by_numb_pred_backward = {}
previous_stage_best_features = list(X_train.columns)


# Include the case where no variable is removed
r2_coef = 0
for fold_index in range(0, 5):
    x_fold_train = X_train.loc[cv_fold != fold_index]
    y_fold_train = y_train.loc[cv_fold != fold_index]
    fitted_lm = sm.OLS(y_fold_train, x_fold_train).fit()

    y_fold_validation = y_train.loc[cv_fold == fold_index]
    x_fold_validation = X_train.loc[cv_fold == fold_index]

    y_fold_validation_pred = fitted_lm.predict(x_fold_validation)

    r2_coef += 1 - np.sum((y_fold_validation - y_fold_validation_pred) ** 2) / np.sum((y_fold_validation - np.mean(y_fold_validation)) ** 2)

best_model_by_numb_pred_backward[13] = {
    "features": list(X_train.columns),
    "r2coef": r2_coef / 5
}

for numb_removed_pred_backward in range(1, 13):

    available_features = set(previous_stage_best_features)

    possibilit_max_coef = 0
    for feature in available_features:
        # We remove a feature from the previous best subset of features to test the model with this new subset
        new_features = list(set(previous_stage_best_features) - set([feature]))
        r2_coef = 0
        # We perform a 5-fold cross validation
        for fold_index in range(0, 5):
            x_fold_train = X_train.loc[cv_fold != fold_index, new_features]
```

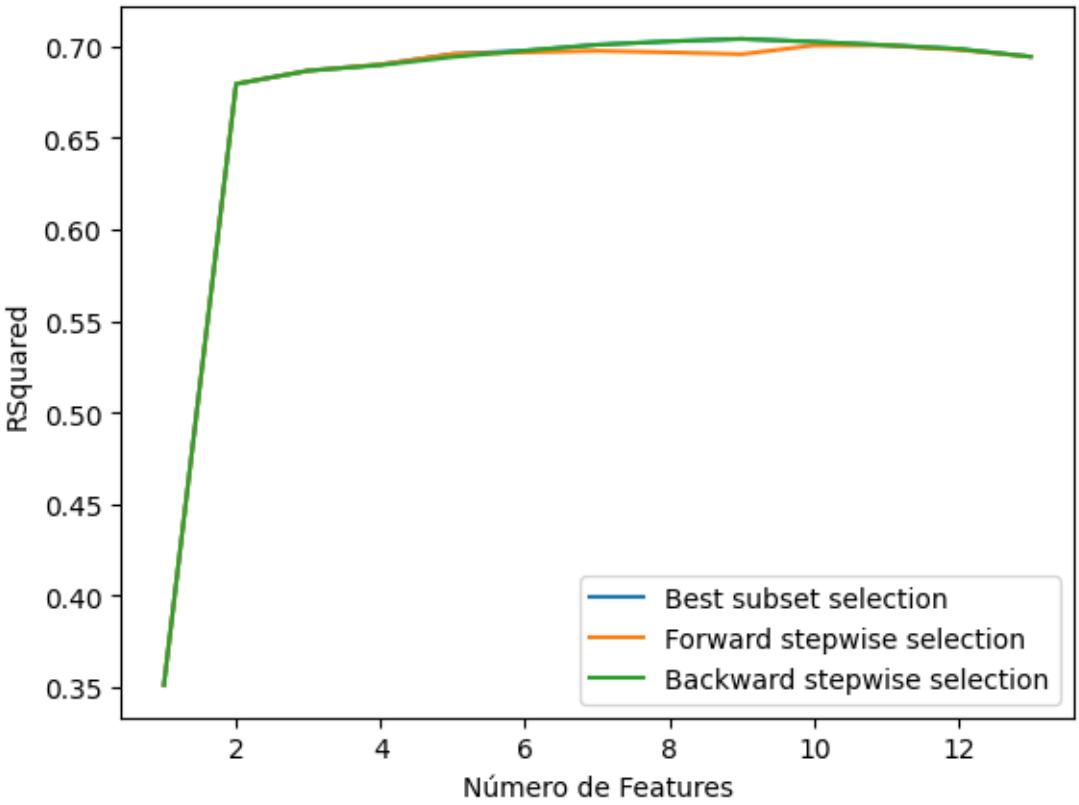
```
            y_fold_train = y_train.loc[cv_fold != fold_index]
            fitted_lm = sm.OLS(y_fold_train, x_fold_train).fit()

            y_fold_validation = y_train.loc[cv_fold == fold_index]
            x_fold_validation = X_train.loc[cv_fold == fold_index, new_features]

            y_fold_validation_pred = fitted_lm.predict(x_fold_validation)

            r2_coef += 1 - np.sum((y_fold_validation - y_fold_validation_pred) ** 2) / np.sum((y_fold_validation - np.mean(y_fold_validation)) ** 2)
        avg_r2_coef = r2_coef / 5

        # If the model with this subset of features is better than the previous best model, we save its features and update the best r2 coefficient
        if possibilit_max_coef < avg_r2_coef:
            possibilit_max_coef = avg_r2_coef
            best_subset = new_features

    # Save the best model for this number of features
    best_model_by_numb_pred_backward[13 - numb_removed_pred_backward] = {
        "features": best_subset,
        "r2coef": possibilit_max_coef
    }
    previous_stage_best_features = best_subset

    # Save the best overall model
    if overall_max_coef_backward < possibilit_max_coef:
        overall_max_coef_backward = possibilit_max_coef
        overall_best_params_backward = best_subset

# Retrain the best model using all training data
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 Exercicio 6c

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

```
plt.plot(
    list(best_model_by_numb_pred_bs.keys()),
    [best_model_by_numb_pred_bs[key]["r2coef"] for key in
    best_model_by_numb_pred_bs.keys()],
    label="Best subset selection"
)

plt.plot(
    list(best_model_by_numb_pred_forward.keys()),
    [best_model_by_numb_pred_forward[key]["r2coef"] for key in
    best_model_by_numb_pred_forward.keys()],
    label="Forward stepwise selection"
)

plt.plot(
    list(best_model_by_numb_pred_backward.keys()),
    [best_model_by_numb_pred_backward[key]["r2coef"] for key in
    best_model_by_numb_pred_backward.keys()],
    label="Backward stepwise selection"
)

plt.xlabel('N mero de Features')
plt.ylabel('RSquared')
plt.legend()
plt.show()
```


0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
RSquared
2
4
6
8
10
12
Número de Features
Best subset selection
Forward stepwise selection
Backward stepwise selection

27 Exercicio 6d

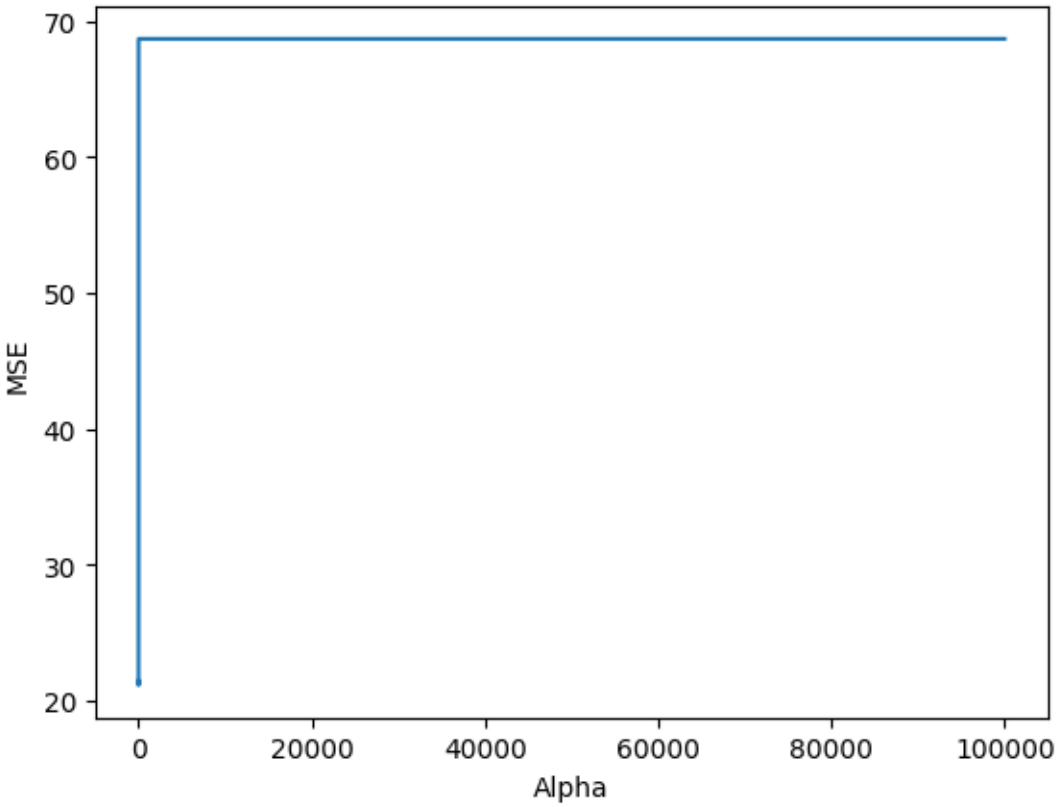
Código usado para treinar modelo de lasso:

```
X_train_processed = preprocessing.scale(X_train)

min_mse = np.inf
best_alpha = None
alpha_and_mse = {}
# Lasso for all alphas
for alpha in alphas:
    lasso_model = Lasso(alpha=alpha)
    sum_mse = 0
    for fold_index in range(0, 5):
        x_fold_train = X_train_processed[cv_fold != fold_index]
        y_fold_train = y_train[cv_fold != fold_index]
        lasso_model.fit(x_fold_train, y_fold_train)

        y_fold_validation = y_train[cv_fold == fold_index]
        x_fold_validation = X_train_processed[cv_fold == fold_index]
        y_fold_validation_pred = lasso_model.predict(x_fold_validation)

        sum_mse += np.sum((y_fold_validation - y_fold_validation_pred) ** 2) / len
    (y_fold_validation)

    avg_mse = sum_mse / 5
    alpha_and_mse[alpha] = avg_mse

    if avg_mse < min_mse:
        min_mse = avg_mse
        best_alpha = alpha

# Retrain the best model using all training data
lasso_model = Lasso(alpha=best_alpha)
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:

```
plt.plot(
    list(alpha_and_mse.keys()),
    [alpha_and_mse[key] for key in alpha_and_mse.keys()],
    label="Lasso"
)
```


70
60
50
40
30
20
MSE
0
20000
40000
60000
80000
100000
Alpha

28 Exercicio 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:

```
# MSE for lasso
X_test_processed = preprocessing.scale(X_test)
y_test_pred_lasso = best_lasso_model.predict(X_test_processed)
mse_test_lasso = np.sum((y_test - y_test_pred_lasso) ** 2) / len(y_test)

# MSE for best subset selection
X_test_best_bs = X_test[overall_best_params_bs]
y_test_pred_bs = best_bs_model.predict(X_test_best_bs)
mse_test_bs = np.sum((y_test - y_test_pred_bs) ** 2) / len(y_test)

# MSE for forward stepwise selection
X_test_best_forward = X_test[overall_best_params_forward]
y_test_pred_forward = best_forward_model.predict(X_test_best_forward)
mse_test_forward = np.sum((y_test - y_test_pred_forward) ** 2) / len(y_test)

# MSE for backward stepwise selection
X_test_best_backward = X_test[overall_best_params_backward]
y_test_pred_backward = best_backward_model.predict(X_test_best_backward)
mse_test_backward = np.sum((y_test - y_test_pred_backward) ** 2) / len(y_test)

print(
    f"MSE - Lasso: {mse_test_lasso}",
    f"MSE - Best subset selection: {mse_test_bs}",
    f"MSE - Forward stepwise selection: {mse_test_forward}",
    f"MSE - Backward stepwise selection: {mse_test_backward}"
)
```

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

29 Exercicio 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]