

Lista 01 - Machine Learning

IMPA

Pedro Bahia

23 de janeiro de 2026

Conteúdo

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
7.1 i	9
7.2 ii	9
7.3 iii	9
7.4 iv	10
8 Exercício 2c	11
8.1 i	11
8.2 ii	11
8.3 iii	12
9 Exercício 2d	13
9.1 i	13
9.2 ii	15
9.3 iii	16
9.4 iv	17
9.5 v	17
10 Exercício 3a	19

11 Exercício 3b	20
12 Exercício 3c	21
13 Exercício 3d	22
14 Exercício 3e	23
15 Exercício 3f	24
15.1 i) Implementação dos Estimadores	24
15.2 ii) Comparação dos Estimadores	26
15.3 iii) Robustez a Outliers	27
16 Exercício 4a	29
17 Exercício 4b	30
17.1 Implementação dos Algoritmos de Classificação	30
17.2 Treinamento e Avaliação dos Modelos	31
18 Exercício 4c	32
19 Exercício 4d	33
19.1 Análise Específica do k-NN	33
20 Exercício 5a	34
21 Exercício 5b	35
21.1 Métodos de Seleção de Modelos	35
21.2 Implementação dos Algoritmos	35
22 Exercício 5c	44
22.1 Comparação dos Métodos - R^2	44
23 Exercício 5d	45
23.1 Regressão Lasso com Validação Cruzada	45
23.2 Seleção do Parâmetro de Regularização	45
24 Exercício 5e	47
24.1 Comparação de Erros de Teste	47
24.2 Ranking dos Métodos	47
24.3 Análise dos Resultados	48
25 Exercício 5f	49

1 Exercício 1a

Falso. A proximidade entre $\varepsilon_{\text{treino}}$ e $\varepsilon_{\text{teste}}$ pode dar informações sobre o ajuste do modelo aos dados de treino.

Caso $\varepsilon_{\text{treino}}$ seja próximo ao $\varepsilon_{\text{teste}}$, o modelo pode estar *subajustado*, de modo que aumentar a complexidade poderia melhorar sua performance ainda mais.

De maneira análoga, caso o $\varepsilon_{\text{treino}}$ seja menor que o $\varepsilon_{\text{teste}}$, o modelo estará *sobreajustado*, com redução de complexidade podendo resultar em melhorias.

2 Exercício 1b

Verdadeiro. A distribuição t surge do fato de $\mathcal{N}(0, 1)$ dividido por $\sqrt{\frac{K}{N}}$ ter distribuição t com N graus de liberdade.

No nosso caso, $\mathcal{N}(0, 1)$ é a distribuição de $\frac{\hat{\beta}_1 - \beta_1}{\text{Var}(\hat{\beta}_1)}$. Isso é normal, pois $\hat{\beta}_1$ é normal.

Isso, por sua vez, vem do fato de $\hat{\beta}$ ser resultante de uma combinação linear de gaussianas, no caso, $\varepsilon \sim \mathcal{N}(0, \sigma^2)$.

3 Exercício 1c

Falso. Considerando a classe $k = 0$ como as transações fraudulentas, o objetivo do modelo pode ser interpretado como:

$$\sum_{y_i \in k=0} \mathbf{1}_{[y_i \neq \hat{y}_i]} = 0$$

Não há restrições entretanto em relação às transações legítimas, ou seja, para:

$$\sum_{y_i \in k=1} \mathbf{1}_{[y_i \neq \hat{y}_i]}$$

Dado um modelo de acurácia $(1 - \varepsilon)$, têm-se que

$$1 - \frac{1}{n} \left[\sum_{y_i \in k=0} \mathbf{1}_{[y_i \neq \hat{y}_i]} + \sum_{y_i \in k=1} \mathbf{1}_{[y_i \neq \hat{y}_i]} \right] = 1 - \varepsilon$$

Dado uma acurácia

$$1 - \text{epsilon}$$

, há infinitos valores de $\sum_{y_i \in k=0} \mathbf{1}_{[y_i \neq \hat{y}_i]}$ e $\sum_{y_i \in k=1} \mathbf{1}_{[y_i \neq \hat{y}_i]}$ que resolvem essa equação e, portanto, o valor da acurácia não é informativo para o erro individual das classes. Assim, apenas com a acurácia dos Modelos 1 e 2 não é possível determinar qual modelo tem menor erro em transações fraudulentas.

4 Exercício 1d

Verdadeiro

$$L_{ridge}(\beta) = (Y - Yhat)^T(Y - \hat{Y}) + \lambda\beta^T\beta$$

$$L_{ridge}(\beta) = (Y - X\beta)^T(Y - X\beta) + \lambda\beta^T\beta$$

$$L_{linear}(\beta) = (Y - Yhat)^T(Y - \hat{Y})$$

Caso $\lambda = 0$, temos que $L_{ridge}(\beta) = L_{linear}(\beta)$. Logo, a performance de ambos os modelos será a mesma. Então a afirmação será verdadeira para o case de $\lambda = 0$,

5 Exercício 1e

Verdadeira

A equação dada pela fórmula do intervalo de confiança:

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

é derivada do fato de $\hat{\beta}$ ter uma distribuição normal. Isso vem do fato da hipótese de que o erro é normal.

Caso ela não seja feita, os intervalos de confiança gerados via *bootstrap* são mais adequados, pois são derivados a partir da distribuição inferida diretamente dos dados. Neste caso, a hipótese não-paramétrica é mais geral e preferível.

Justificativa:

- **Abordagem paramétrica:** Assume que os erros seguem distribuição normal, permitindo o uso da distribuição t de Student para construir intervalos de confiança analíticos.
- **Abordagem não-paramétrica (bootstrap):** Não assume distribuição específica dos erros, utilizando reamostragem dos dados para estimar a distribuição empírica dos parâmetros.
- **Vantagem do bootstrap:** Mais robusto quando as suposições paramétricas são violadas, especialmente em casos de não-normalidade dos erros.

6 Exercício 2a

Dado que a variância de ε é:

$$\Sigma = \begin{pmatrix} \text{Cov}(\varepsilon_1, \varepsilon_1) & \text{Cov}(\varepsilon_1, \varepsilon_2) & \cdots & \text{Cov}(\varepsilon_1, \varepsilon_n) \\ \text{Cov}(\varepsilon_2, \varepsilon_1) & \text{Cov}(\varepsilon_2, \varepsilon_2) & \cdots & \text{Cov}(\varepsilon_2, \varepsilon_n) \\ \vdots & \vdots & \ddots & \vdots \\ \text{Cov}(\varepsilon_n, \varepsilon_1) & \text{Cov}(\varepsilon_n, \varepsilon_2) & \cdots & \text{Cov}(\varepsilon_n, \varepsilon_n) \end{pmatrix}$$

Tal que $\text{Cov}(\varepsilon_i, \varepsilon_j) = \mathbb{E}[(\varepsilon_i - \mu_i)(\varepsilon_j - \mu_j)]$.

Assumir a independência dos erros implica que $\text{Cov}(\varepsilon_i, \varepsilon_j) = 0$ para $i \neq j$. Isso resulta em uma matriz de covariância diagonal, os elementos fora da diagonal são todos zero.

Já assumir a homocedasticidade implica que a variância dos erros é constante, ou seja,

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

Ou seja, $\text{Var}(\varepsilon_i) = \sigma^2$ para todo i . Assim, os elementos na diagonal da matriz de covariância são todos iguais a σ^2 .

Portanto, sob as suposições de independência e homocedasticidade dos erros, a matriz de covariância Σ assume a forma:

$$\Sigma = \begin{pmatrix} \sigma^2 & 0 & \cdots & 0 \\ 0 & \sigma^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma^2 \end{pmatrix} = \sigma^2 I_n$$

onde I_n é a matriz identidade de ordem n .

7 Exercício 2b

7.1 i

A função de perda ponderada pela matriz de covariância dos erros é dada por:

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

Esta função é convexa em relação a β , pois, sendo Σ^{-1} é uma matriz positiva definida e a função quadrática $(Y - X\beta)^T (Y - X\beta)$ estritamente convexa, seu produto também é estritamente convexo.

Assim, para encontrar o estimador $\hat{\beta}_\Sigma$, derivamos a função de perda em relação a β e igualamos a zero a derivada:

$$\frac{d\hat{\beta}_\Sigma}{d\beta} (Y - X\beta)^T \Sigma^{-1} (Y - X\beta) = [-2X^T \Sigma^{-1} (Y - X\beta)] = 0$$

Resolvendo para β , obtemos:

$$X^T \Sigma^{-1} Y = X^T \Sigma^{-1} X \beta$$

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

7.2 ii

Como $(X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1}$ é uma constante em relação a Y , e sabemos que $Y = X\beta + \varepsilon$, onde $\mathbb{E}[\varepsilon] = 0$ e $\mathbb{E}[Y] = X\beta$, temos:

$$\begin{aligned}\mathbb{E}[\hat{\beta}_\Sigma] &= \mathbb{E}[(X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} Y] \\ \mathbb{E}[\hat{\beta}_\Sigma] &= (X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} \mathbb{E}[Y] \\ \mathbb{E}[\hat{\beta}_\Sigma] &= (X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} X \beta \\ \mathbb{E}[\hat{\beta}_\Sigma] &= \beta\end{aligned}$$

7.3 iii

$(X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1}$ é uma constante em relação a Y e pode ser fatorado para fora da operação de variância como sua trasposta multiplicando pela direita. Além disso, sabemos que $\mathbb{V}(Y) = \varepsilon = \Sigma$, temos:

$$\begin{aligned}\mathbb{V}(\hat{\beta}_\Sigma) &= \mathbb{V}((X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} Y) \\ \mathbb{V}(\hat{\beta}_\Sigma) &= (X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} \mathbb{V}(Y) \Sigma^{-1} X (X^T \Sigma^{-1} X)^{-1} \\ \mathbb{V}(\hat{\beta}_\Sigma) &= (X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} \Sigma \Sigma^{-1} X (X^T \Sigma^{-1} X)^{-1} \\ \mathbb{V}(\hat{\beta}_\Sigma) &= (X^T \Sigma^{-1} X)^{-1}\end{aligned}$$

7.4 iv

Sendo o ε uma variável aleatória distribuída $\varepsilon \sim \mathcal{N}(0, \Sigma)$, Y uma i.i.d com $Y \sim \mathcal{N}(X\beta, \Sigma)$ e Σ e X fixos, o estimador $\hat{\beta}_\Sigma$ é uma combinação linear de variáveis aleatórias normais. Portanto, $\hat{\beta}_\Sigma$ também é uma variável aleatória normal. Assim, temos que:

$$\hat{\beta}_\Sigma \sim \mathcal{N}(\mathbb{E}[\hat{\beta}_\Sigma], \mathbb{V}(\hat{\beta}_\Sigma))$$

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

8 Exercício 2c

8.1 i

Como Σ é uma matriz diagonal e positiva definida, sua inversa Σ^{-1} também será uma matriz diagonal, cujo elementos são os inversos dos elementos diagonais de Σ . Sua fatoração $\Sigma^{-1/2}$ também será uma matriz diagonal, cujos elementos são as raízes quadradas dos elementos de Σ^{-1} . Por fim, a transposta de $\Sigma^{-1/2}$ será igual a $\Sigma^{-1/2}$.

Assim, podemos reescrever a função de perda ponderada como:

$$\begin{aligned}\mathcal{L} &= (y - X\beta)^T \Sigma^{-1} (y - X\beta) \\ &= (y - X\beta)^T \Sigma^{-1/2} \Sigma^{-1/2} (y - X\beta) \\ &= (\Sigma^{-1/2} y - \Sigma^{-1/2} X\beta)^T (\Sigma^{-1/2} y - \Sigma^{-1/2} X\beta)\end{aligned}$$

Novamente, como $\Sigma^{-1/2}$ é uma matriz diagonal, cada elemento é dos vetores é multiplicado pelo respectivo elemento diagonal de $\Sigma^{-1/2}$. Enfim, expandindo a soma temos:

$$\begin{aligned}\mathcal{L} &= \sum_{i=1}^n \left(\frac{1}{\sigma_{ii}} y_i - \frac{1}{\sigma_{ii}} X_i \beta \right)^2 \\ &= \sum_{i=1}^n \frac{1}{\sigma_{ii}^2} (y_i - X_i \beta)^2 \\ &= \sum_{i=1}^n w_i^2 (y_i - X_i \beta)^2 \quad \text{onde } w_i = \frac{1}{\sigma_{ii}} \\ &= \sum_{i=1}^n w_i^2 (y_i - \beta_0 - \sum_{j=1}^p X_{ij} \beta_j)^2\end{aligned}$$

8.2 ii

Os pesos da função de perda $w_i = \frac{1}{\sqrt{\Sigma_{ii}}}$ ponderam a amostra i pelo inverso da variância do erro, de modo a normalizar as contribuições das amostras para a função de perda total. Assim, amostras com maior variância, ou seja, com maior incerteza e dificuldade de ajuste, terão menor impacto na função de perda, enquanto amostras com menor variância terão mais.

8.3 iii

De modo análogo à seção i), dado que Σ é positiva definida, podemos reescrever a função de perda ponderada pela matriz de covariância dos erros

$$\mathcal{L} = (\Sigma^{-1/2}y - \Sigma^{-1/2}X\beta)^T(\Sigma^{-1/2}y - \Sigma^{-1/2}X\beta) = (\tilde{Y} - \tilde{X}\beta)^T(\tilde{Y} - \tilde{X}\beta)$$

onde $\tilde{Y} = \Sigma^{-1/2}y$ e $\tilde{X} = \Sigma^{-1/2}X$.

Entretanto, como os elementos fora das diagonais não são nulos, o somatório é expandido como

$$\mathcal{L} = \sum_{i=1}^n \sum_{j=1}^n w_{ij}(y_i - X_i\beta)^2 = \sum_{i=1}^n \sum_{j=1}^n w_{ij}(y_i - \beta_0 - \sum_{k=1}^p X_{ik}\beta_k)^2$$

9 Exercício 2d

9.1 i

Os dados com heterocedasticidade usando a matriz de covariância Σ diagonal:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 n = 50
5 Sigma = np.diag([10 ** ((i - 20) / 5) for i in range(1, n + 1)])
6 np.random.seed(0)
7 X = np.array([np.ones(n), np.random.normal(0, 1, n)]).T
8 beta = np.array([1, 0.25])
9 epsilon = np.random.multivariate_normal(np.zeros(n), Sigma)
10 y = X @ beta + epsilon
```

A Figura 1 mostra os dados gerados com heterocedasticidade:

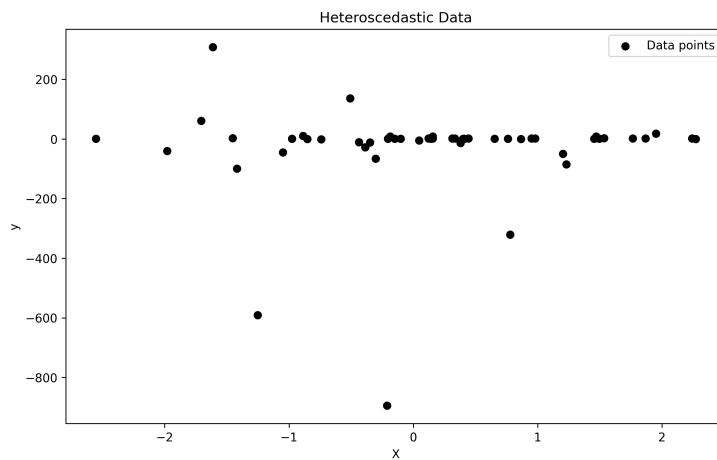


Figura 1: Dados heteroscedásticos gerados

A Figura 2 mostra os elementos diagonais da matriz Σ , evidenciando a heterocedasticidade:

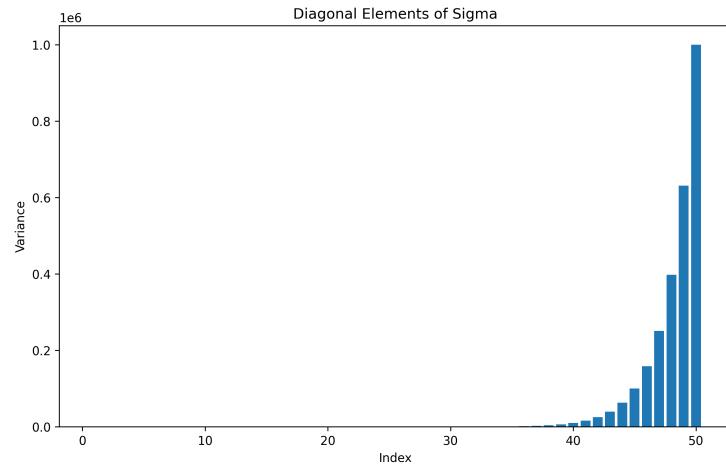


Figura 2: Elementos diagonais da matriz Σ

As Figuras 3 e 4 mostram a distribuição e valores dos erros:

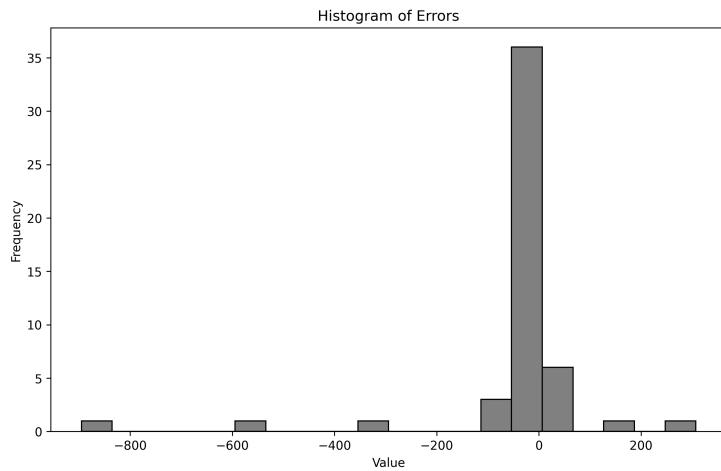


Figura 3: Histograma dos erros

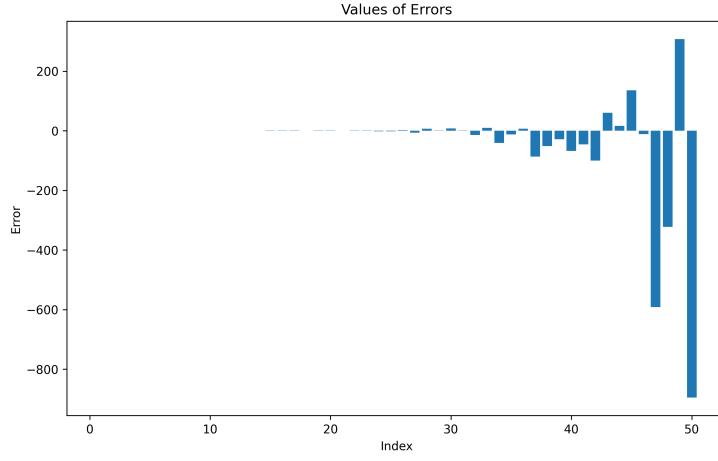


Figura 4: Valores dos erros por índice

9.2 ii

Tanto estimador de mínimos quadrados ordinários quanto o estimador generalizado que considera a matriz de covariância Σ foram implementados com o código à seguir:

```

1 def beta_ordinary(X: np.ndarray, Y: np.ndarray) -> np.
2     ndarray:
3         """
4             Compute the ordinary least squares estimator.
5         """
6         beta = np.linalg.inv(X.T @ X) @ X.T @ Y
7         return beta
8
9 def beta_sigma(X: np.ndarray, Y: np.ndarray, Sigma: np.
10    ndarray) -> np.ndarray:
11        """
12            Compute the generalized least squares estimator
13            considering
14            the covariance matrix Sigma.
15        """
16        Sigma_1 = np.linalg.inv(Sigma)
17        beta = np.linalg.inv(X.T @ Sigma_1 @ X) @ X.T @ Sigma_1
18        @ Y
19        return beta
20
21 beta_hat_ordinary = beta_ordinary(X, y)
22 beta_hat_sigma = beta_sigma(X, y, Sigma)

```

Comparação dos Estimadores:

- Parâmetros Verdadeiros: $\beta = [1.0, 0.25]$

- Estimador Ordinário: $\hat{\beta}_{OLS} = [-34.463, 6.948]$

- Estimador Generalizado: $\hat{\beta}_{\Sigma} = [1.019, 0.244]$

Erro Quadrático dos Estimadores:

- $\|\beta - \hat{\beta}_{OLS}\|_2^2 = 1302.51$
- $\|\beta - \hat{\beta}_{\Sigma}\|_2^2 = 0.000387$

9.3 iii

```

1 def p_value_ordinary_least_square(X: np.ndarray, Y: np.
2     ndarray,
3         beta_ordinary_hat: np.
4             ndarray, j: int) -> float:
5     """
6     Compute the p-value for the j-th coefficient of the
7     ordinary
8     least squares estimator.
9     """
10    Y_hat = X @ beta_ordinary_hat
11    n, p = X.shape
12    dof = n - p
13    errors = Y - Y_hat
14    beta_j = beta_ordinary_hat[j]
15
16    # Z statistic
17    x_j_var = (np.linalg.inv(X.T @ X))[j, j]
18    Z = beta_j / np.sqrt(x_j_var)
19
20    # Estimate of sigma^2
21    sigma2_hat = (1 / dof) * (errors.T @ errors)
22
23    # t statistic and p-value
24    t_statistics = Z / np.sqrt(sigma2_hat)
25    t_statistics = np.abs(t_statistics)
    p_value = 2 * (1 - scipy.stats.t.cdf(t_statistics, dof))

    return p_value

```

Testes de Hipótese - Mínimos Quadrados Ordinários:

- p-valor para β_0 : 0.1544
- p-valor para β_1 : 0.7422

Não podemos descartar a hipótese nula para ambos os coeficientes ao nível de significância de 5%.

9.4 iv

```

1 def calculate_Z_sigma(X: np.ndarray, Sigma: np.ndarray,
2                         Beta_sigma: np.ndarray, j: int) ->
3     float:
4         """
5             Compute the Z statistic for the j-th coefficient of the
6             generalized least squares estimator.
7         """
8         Sigma_inv = np.linalg.inv(Sigma)
9         den = np.linalg.inv(X.T @ Sigma_inv @ X)
10        den = den[j, j]
11        Z = Beta_sigma[j] / (np.sqrt(den))
12        return Z

```

- Estatística Z_Σ para β_0 : 73.67

9.5 v

Condicionado em X , o termo da diagonal i do denominador de Z é a raiz quadrada da variância do estimador generalizado $\hat{\beta}_\Sigma$.

$$\text{Variância de } \hat{\beta} = (X^T \Sigma^{-1} X)^{-1} = \begin{pmatrix} \text{Var}(\beta_0) & \text{Cov}(\beta_0, \beta_1) \\ \text{Cov}(\beta_0, \beta_1) & \text{Var}(\beta_1) \end{pmatrix}$$

Na hipótese H_0 de $\beta_0 = 0$, temos que $\hat{\beta}_\Sigma \sim \mathcal{N}(\beta_0, \text{Var}(\beta_0))$. Assim, a estatística Z é dada por:

$$Z_\Sigma = \frac{\hat{\beta}_\Sigma}{\sqrt{\text{Var}(\hat{\beta}_\Sigma)}}$$

Para obtermos o p-valor, fazemos $Z/\sqrt{\sigma^2}$, onde $\sigma^2 = \frac{1}{n-p-1} \varepsilon^T \Sigma^{-1} \varepsilon$

$$\begin{aligned} \text{Var}(\hat{\beta}_\Sigma) &= \text{Var}((X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} Y) \\ &= (X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} \text{Var}(Y) \Sigma^{-1} X (X^T \Sigma^{-1} X)^{-1} \\ &= (X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} \Sigma \Sigma^{-1} X (X^T \Sigma^{-1} X)^{-1} \\ &= (X^T \Sigma^{-1} X)^{-1} \end{aligned}$$

Calculando o p-valor com o código abaixo:

```

1 def p_value_generalized_least_square(X: np.ndarray, Y: np.
2                                         ndarray,
                                         Sigma: np.ndarray,
                                         ...
                                         )

```

```

3             beta_ordinary_hat: np.
4         ndarray, j: int) -> float:
5         """
6         Compute the p-value for the j-th coefficient of the
7         generalized least squares estimator.
8         """
9         Y_hat = X @ beta_ordinary_hat
10        n, p = X.shape
11        dof = n - p
12        errors = Y - Y_hat
13
14        # Z statistic
15        Z_sigma = calculate_Z_sigma(X, Sigma, beta_hat_sigma, j)
16
17        # Estimate of sigma^2
18        inverse_Sigma = np.linalg.inv(Sigma)
19        sigma2_hat = (1 / dof) * (errors.T @ inverse_Sigma @
20        errors)
21
22        # t statistic and p-value
23        t_statistics = Z_sigma / np.sqrt(sigma2_hat)
24        t_statistics = np.abs(t_statistics)
25        p_value = 2 * (1 - scipy.stats.t.cdf(t_statistics, dof))

    return p_value

```

Esta implementação permite comparar os dois estimadores e calcular a significância estatística dos coeficientes em ambos os casos.

Testes de Hipótese - Estimador Generalizado:

- p-valor para β_0 : 0.0
- p-valor para β_1 : 0.0

10 Exercício 3a

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

$$\epsilon \sim \text{Laplace}(0, b)$$

$$Y_i|X_i = Y = \beta^T X + \epsilon$$

$$\begin{aligned}\mathbb{E}(Y) &= \mathbb{E}(\beta X + \epsilon) = \mathbb{E}(\beta^T X) + \mathbb{E}(\epsilon) \\ &= \beta^T X + 0\end{aligned}$$

$$\begin{aligned}\text{Var}(Y) &= \text{Var}(\beta^T X + \epsilon) = \text{Var}(\beta^T X) + \text{Var}(\epsilon) \\ &= 0 + b = b\end{aligned}$$

Como toda combinação linear de variáveis aleatórias com distribuição de Laplace resulta em uma distribuição de Laplace, Y é uma variável aleatória de Laplace com parâmetros $\text{Laplace}(\beta^T X, b)$.

Função de perda (likelihood):

Dado que Y é iid, temos que:

$$P(Y|X, \beta) = \prod_{i=1}^n \frac{1}{2b} \exp\left(-\frac{|Y_i - \beta^T X_i|}{b}\right)$$

$$\begin{aligned}\log P(Y|X, \beta) &= \log \prod_{i=1}^n \frac{1}{2b} \exp\left(-\frac{|Y_i - \beta^T X_i|}{b}\right) \\ &= \sum_{i=1}^n \log\left(\frac{1}{2b} \exp\left(-\frac{|Y_i - \beta^T X_i|}{b}\right)\right) \\ &= \sum_{i=1}^n \left(\log \frac{1}{2b} - \frac{|Y_i - \beta^T X_i|}{b}\right) \\ &= n \log \frac{1}{2b} - \frac{1}{b} \sum_{i=1}^n |Y_i - \beta^T X_i|\end{aligned}$$

11 Exercício 3b

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

$$\epsilon \sim \text{Laplace}(0, b)$$

$$Y_i|X_i = Y = \beta^T X + \epsilon$$

$$\begin{aligned}\mathbb{E}(Y) &= \mathbb{E}(\beta X + \epsilon) = \mathbb{E}(\beta^T X) + \mathbb{E}(\epsilon) \\ &= \beta^T X + 0\end{aligned}$$

$$\begin{aligned}\mathbb{V}(Y) &= \mathbb{V}(\beta X + \epsilon) = \mathbb{V}(\beta X) + \mathbb{V}(\epsilon) \\ &= 0 + b\end{aligned}$$

Toda combinação linear de Laplace são Laplace:

Item b

$$\mathcal{L}(\beta) = -\log(\ell(\beta|x, y, b))$$

$$= -\log\left(\frac{1}{2b} \cdot e^{-\sum_i^n \frac{|y_i - x_i \beta|}{b}}\right)$$

$$= -\log(2b) + \sum_i^n \frac{|y_i - x_i \beta|}{b}$$

$$= -\log(2b) + \frac{1}{b} \sum_i^n |y_i - x_i \beta|$$

- $-\log(2b)$ é constante e pode ser desconsiderado na otimização
- o mesmo pode ser dito sobre o termo que multiplica o somatório

12 Exercício 3c

$$\begin{aligned}\epsilon &\stackrel{iid}{\sim} \mathcal{N}(0, 1) \\ Y &= X\beta + \epsilon\end{aligned}$$

$$\begin{aligned}\ell(y|X\beta, \sigma^2) &= \prod_i^n \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{1}{2}\left(\frac{y_i - X_i\beta}{\sigma}\right)^2} \\ &= \frac{1}{\sigma^n (\sqrt{2\pi})^n} \cdot e^{-\frac{1}{2\sigma^2} \sum (y_i - X_i\beta)^2}\end{aligned}$$

$$\begin{aligned}\mathcal{L}(y|\mu, \sigma^2) &= -\log(\ell(y|\mu, \sigma^2)) \\ &= -\log \left[\frac{1}{\sigma^n (\sqrt{2\pi})^n} \cdot e^{-\frac{1}{2\sigma^2} \sum_i^n (y_i - X_i\beta)^2} \right] \\ &= \log(\sigma^n (\sqrt{2\pi})^n) + \frac{1}{2\sigma^2} \sum_i^n (y_i - X_i\beta)^2 \\ \mathcal{L}(y|\mu, \sigma^2) &\approx \sum_i^n (y_i - X_i\beta)^2\end{aligned}$$

Vantagem: em caso de outliers, erro muito alto, o erro para a amostra i na função (ℓ) será elevado ao quadrado, resultando em maiores impactos na função de perda.

O mesmo não ocorre em b.

13 Exercício 3d

14 Exercício 3e

15 Exercício 3f

15.1 i) Implementação dos Estimadores

Neste exercício, comparamos estimadores baseados em diferentes suposições sobre a distribuição dos erros. Implementamos estimadores que minimizam diferentes funções de perda:

```
1 import numpy as np
2 import scipy
3 import matplotlib.pyplot as plt
4
5 def beta_ordinary(X: np.ndarray, Y: np.ndarray) -> np.
6     ndarray:
7         """
8             Compute the ordinary least squares estimator.
9         """
10        beta = np.linalg.inv(X.T @ X) @ X.T @ Y
11        return beta
12
13 def calculate_beta_hat(X: np.ndarray, Y: np.ndarray,
14     error_distribution: str) -> np.ndarray:
15     """
16         Compute the estimator beta_hat based on the specified
17         error distribution.
18     """
19     np.random.seed(0)
20     p = X.shape[1]
21
22     if error_distribution == "gaussian":
23         def loss_function(beta):
24             return np.sum((Y - X @ beta) ** 2)
25     elif error_distribution == "laplacian":
26         def loss_function(beta):
27             return np.sum(np.abs(Y - X @ beta))
28     else:
29         raise ValueError("Unsupported error distribution")
30
31     beta_0 = np.random.uniform(size=p)
32     beta_hat = scipy.optimize.minimize(loss_function, beta_0)
33
34     return beta_hat["x"]
```

Geramos dados sintéticos para testar os estimadores:

```
1 np.random.seed(1)
2 beta = np.array([-1.5, 2.0])
3 input_range = np.linspace(-1, 1, 100)
4 X = np.vstack([np.ones(100), input_range]).T
5 y = X @ beta + np.random.normal(0, 0.3, 100)
```

A Figura 5 mostra os dados gerados:

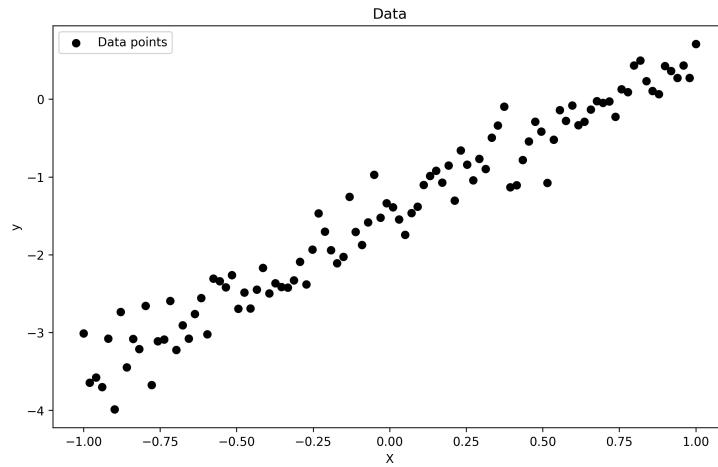


Figura 5: Dados sintéticos para comparação dos estimadores

As Figuras 6 e 7 mostram a análise dos erros:

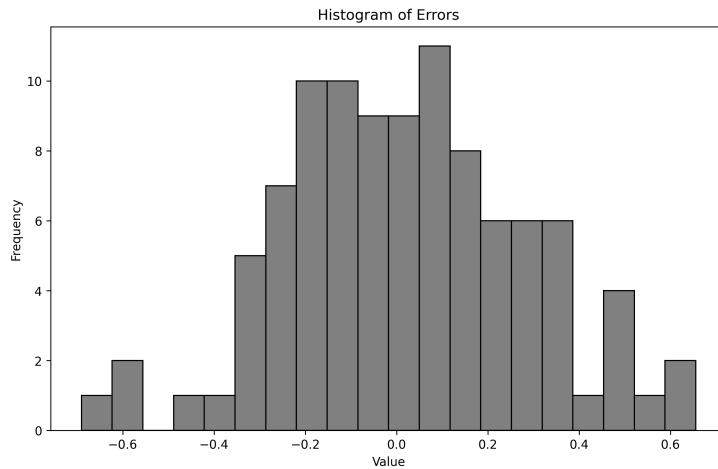


Figura 6: Histograma dos erros

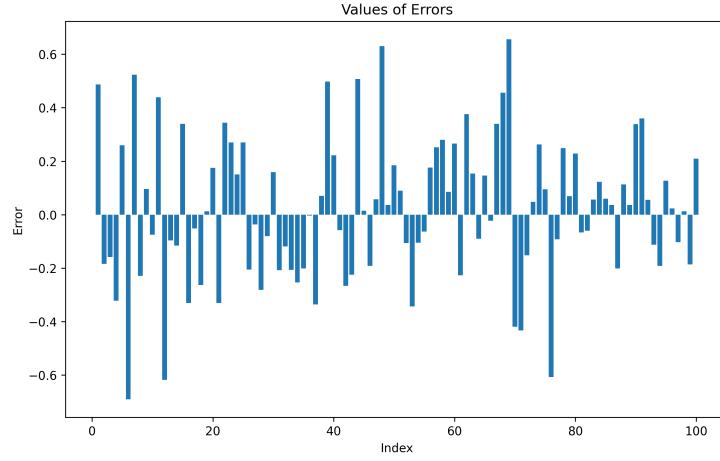


Figura 7: Valores dos erros por índice

15.2 ii) Comparação dos Estimadores

Calculamos os estimadores para ambas as distribuições de erro:

Resultados sem Outliers:

- Parâmetros Verdadeiros: $\beta = [-1.5, 2.0]$
- Estimador Gaussiano (minimize): $\hat{\beta}_{Gauss} = [-1.482, 2.050]$
- Estimador Gaussiano (forma fechada): $\hat{\beta}_{OLS} = [-1.482, 2.050]$
- Estimador Laplaciano: $\hat{\beta}_{Lap} = [-1.498, 2.082]$

Análise de Erro (Norma L2):

- Erro do Estimador Gaussiano: 0.053
- Erro do Estimador Laplaciano: 0.082

Sem outliers, o estimador gaussiano (mínimos quadrados) tem melhor performance, como esperado quando os erros seguem distribuição normal.

A Figura 8 compara visualmente os ajustes:

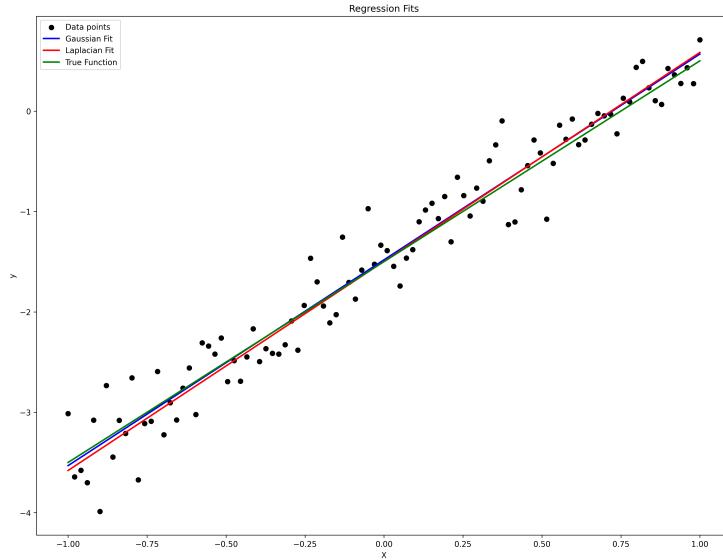


Figura 8: Comparação dos ajustes de regressão sem outliers

15.3 iii) Robustez a Outliers

Para testar a robustez, adicionamos um outlier extremo ao dataset:

```

1 # Regenerate data and add outlier
2 y[80] = 10 # Extreme outlier

```

Resultados com Outlier:

- Parâmetros Verdadeiros: $\beta = [-1.5, 2.0]$
- Estimador Gaussiano com outlier: $\hat{\beta}_{Gauss} = [-1.378, 2.237]$
- Estimador Laplaciano com outlier: $\hat{\beta}_{Lap} = [-1.498, 2.083]$

Análise de Erro com Outlier (Norma L2):

- Erro do Estimador Gaussiano: 0.266 (aumento de 5x)
- Erro do Estimador Laplaciano: 0.083 (praticamente inalterado)

A Figura 9 mostra o impacto do outlier:

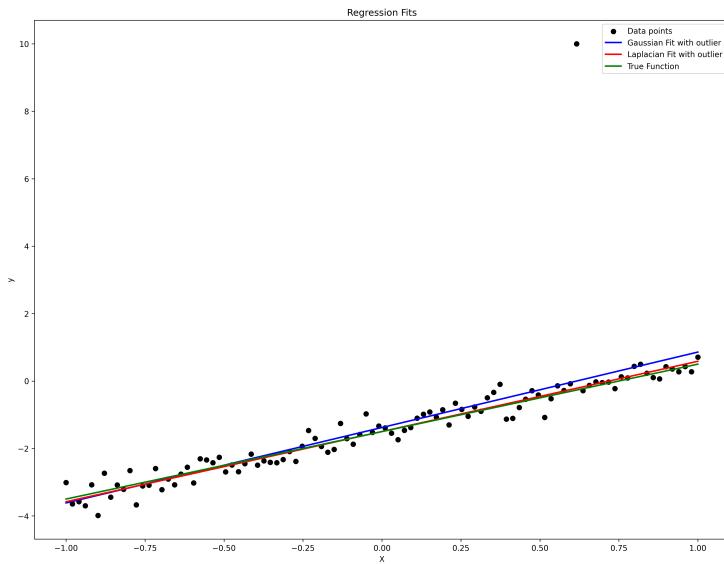


Figura 9: Comparação dos ajustes de regressão com outlier

Conclusões:

1. **Eficiência:** Quando os erros são gaussianos e não há outliers, o estimador de mínimos quadrados (gaussiano) é mais eficiente.
2. **Robustez:** O estimador laplaciano é significativamente mais robusto a outliers, mantendo sua performance praticamente inalterada mesmo com outliers extremos.
3. **Trade-off:** Existe um trade-off entre eficiência (mínimos quadrados) e robustez (estimador laplaciano). A escolha depende das características esperadas dos dados.
4. **Aplicação Prática:** Em situações onde outliers são esperados ou a distribuição dos erros tem caudas pesadas, o estimador laplaciano (regressão com norma L1) é preferível.
5. **Impacto Dramático:** Um único outlier pode degradar significativamente a performance do estimador gaussiano (aumento de 5x no erro), enquanto o estimador laplaciano permanece praticamente inalterado.

16 Exercício 4a

O modelo knn necessida que as features sejam normalizadas pois ele usa uma métrica de distância para encontrar os vizinhos mais próximos. Se as features não forem normalizadas, aquelas com escalas maiores podem dominar a métrica de distância, fazendo com que o algoritmo não funcione corretamente.

17 Exercício 4b

17.1 Implementação dos Algoritmos de Classificação

Neste exercício, implementamos e comparamos cinco diferentes algoritmos de classificação usando o dataset de futebol:

- **LDA** (Linear Discriminant Analysis)
- **QDA** (Quadratic Discriminant Analysis)
- **LR** (Logistic Regression)
- **NB** (Naive Bayes Gaussiano)
- **kNN** (k-Nearest Neighbors)

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import pandas as pd
4 from sklearn.discriminant_analysis import
5     LinearDiscriminantAnalysis as LDA
6 from sklearn.discriminant_analysis import
7     QuadraticDiscriminantAnalysis as QDA
8 from sklearn.linear_model import LogisticRegression as LR
9 from sklearn.naive_bayes import GaussianNB as NB
10 from sklearn.neighbors import KNeighborsClassifier as kNN
11 from sklearn import preprocessing
12
13 # Load and prepare data
14 df = pd.read_csv("../data/soccer.csv")
15 X = df.drop("target", axis=1)
16 y = df[["target"]]
17
18 # Split dataset
19 X_train, y_train = X.iloc[:2560], y.iloc[:2560]
20 X_test, y_test = X.iloc[2560:], y.iloc[2560:]
21
22 # Remove categorical variables and standardize
23 X_train = X_train.drop(["home_team", "away_team"], axis=1)
24 X_test = X_test.drop(["home_team", "away_team"], axis=1)
25 scaler = preprocessing.StandardScaler()
26 X_train = scaler.fit_transform(X_train)
27 X_test = scaler.transform(X_test)
```

Informações do Dataset:

- Amostras de treino: 2560
- Amostras de teste: 640
- Features após pré-processamento: 11 (removendo variáveis categóricas)

17.2 Treinamento e Avaliação dos Modelos

Implementamos um loop para treinar todos os modelos e comparar suas performances:

```
1 models_to_test = [LDA, QDA, LR, NB, kNN]
2 results_dict = {}
3
4 for model_type in models_to_test:
5     model_name = model_type.__name__
6     params = {}
7     if model_type in [LDA, QDA]:
8         params.update({"store_covariance": True})
9
10    results_dict[model_name] = {}
11    cls = model_type(**params)
12    cls.fit(X_train, y_train.values.ravel())
13
14    # Store predictions and model
15    results_dict[model_name]["in_sample_predictions"] = cls.
16    predict(X_train)
17    results_dict[model_name]["test_predictions"] = cls.
18    predict(X_test)
19    results_dict[model_name]["model"] = cls
```

Linear Discriminant Analysis (LDA):

- Coeficientes: [0.81, -0.26, -0.026, 0.017, 0.23, 0.069, 0.37, -0.050, -0.083, 0.043, 0.047]
- Intercepto: 0.109
- Utiliza covariância comum entre as classes

18 Exercício 4c

A Figura 10 compara os erros de treinamento e teste para todos os modelos:

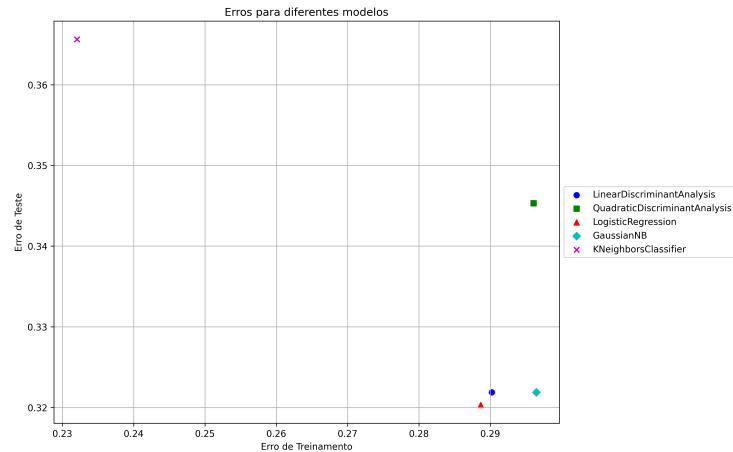


Figura 10: Comparação dos erros de treinamento vs teste para diferentes modelos

19 Exercício 4d

19.1 Análise Específica do k-NN

A Figura 11 mostra como a performance do k-NN varia com diferentes valores de k :

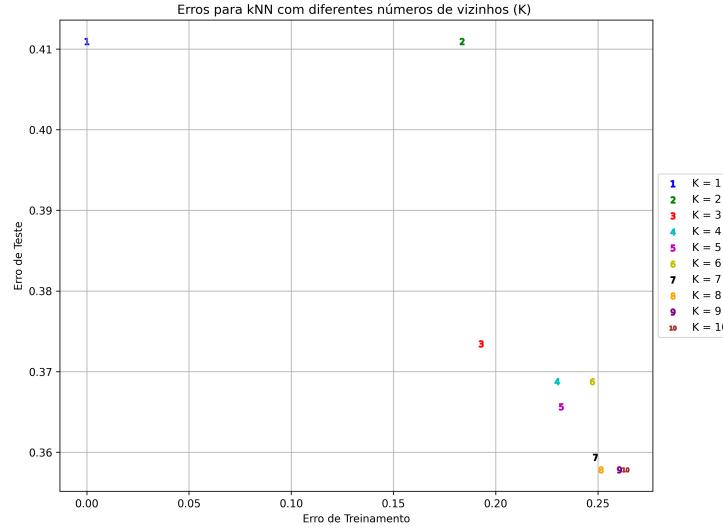


Figura 11: Eros do k-NN para diferentes números de vizinhos ($K=1$ a $K=10$)

Para $k=1$, o modelo apresenta overfitting, com erro de treino nulo já que ele simplesmente repete o dado do vizinho mais próximo, a própria amostra, e erro de teste alto. Conforme k aumenta, o modelo generaliza melhor, aumentando o erro de treino e reduzindo o erro de teste, já que mais vizinhos são considerados na decisão de classe. Para os k analisados, não houve underfitting, mas se k fosse muito grande (próximo ao número total de amostras), o modelo tenderia a classificar todas as amostras na classe majoritária, aumentando ambos os erros.

20 Exercício 5a

```
1 import statsmodels.api as sm
2 import numpy as np
3 import pandas as pd
4 from sklearn.model_selection import train_test_split, KFold
5 from sklearn.linear_model import Lasso
6 from sklearn import preprocessing
7
8 # Load and prepare data
9 bodyfat = pd.read_csv("../data/bodyfat.csv")
10 X = bodyfat.drop(columns=["BodyFat", "Density"])
11 y = bodyfat["BodyFat"]
12
13 # Split data
14 X_train, X_test, y_train, y_test = train_test_split(
15     X, y, test_size=0.2, random_state=10
16 )
17
18 # Setup cross-validation
19 kf = KFold(n_splits=5, shuffle=True, random_state=10)
```

21 Exercício 5b

21.1 Métodos de Seleção de Modelos

Neste exercício, implementamos e comparamos diferentes métodos de seleção de modelos para regressão linear usando o dataset de composição corporal (bodyfat). Os métodos implementados incluem:

- **Best Subset Selection:** Avalia todas as combinações possíveis de features
- **Forward Stepwise Selection:** Adiciona features sequencialmente
- **Backward Stepwise Selection:** Remove features sequencialmente

21.2 Implementação dos Algoritmos

Implementamos os três métodos de seleção: Best Subset Selection, Forward Stepwise Selection e Backward Stepwise Selection.

```
1     def best_subset_selection(
2         X_train: np.ndarray, Y_train: np.ndarray
3     ) -> dict:
4         """
5             Perform best subset selection for linear regression.
6             This function evaluates all possible combinations of
7             features
8             and selects the best model for each subset size based on
9             R-squared.
10
11            Parameters:
12                - X_train (np.ndarray): Training feature data. Shape (
13                    n_samples, n_features).
14                - Y_train (np.ndarray): Training target data. Shape (
15                    n_samples,).
16
17            Returns:
18                - dict: A dictionary where each key is the subset size (
19                    as a string) and the value is another dictionary
20                    containing:
21                        - 'best_model': The statsmodels OLS regression
22                            results object for the best model.
23                        - 'best_r2': The R-squared value of the best model.
24                        - 'best_features': A tuple of feature indices used
25                            in the best model.
26
27
28            """
29
30        # Useful dimensions
```

```

22     n, p = X_train.shape
23
24     # Initialize dictionary to store the best model for each
25     # subset size
26     best_model_k = {}
27
28     # Create model for no feature
29     best_model_k["0"] = {}
30     best_model_k["0"]["best_model"] = sm.OLS(
31         Y_train, np.ones((len(Y_train), 1))
32     ).fit()
33     best_model_k["0"]["best_r2"] = best_model_k["0"]["best_model"].rsquared
34     best_model_k["0"]["best_features"] = []
35     tests_made = [()]
36
37     for k in range(1, p + 1):
38         best_r2 = 0
39         best_features = []
40         best_model = []
41
42         # Create every model with k features (excluding
43         # intercept)
44         for feature_combination in itertools.combinations(
45             range(p), k):
46             X_train_aux = sm.add_constant(X_train[:, feature_combination])
47             model = sm.OLS(Y_train, X_train_aux).fit()
48             tests_made.append(feature_combination)
49             if model.rsquared > best_r2:
50                 best_r2 = model.rsquared
51                 best_model = model
52                 best_features = feature_combination
53
54         # Store the best model for this subset size
55         best_model_k[str(k)] = {
56             "best_model": best_model,
57             "best_r2": best_r2,
58             "best_features": list(best_features),
59         }
60
61     # Sanity checks
62     assert len(tests_made) == 2**p, (
63         "Number of feature subsets evaluated does not match
64         expected count."
65     )
66     assert len(tests_made) == len(set(tests_made)), (
67         "Duplicate feature subsets were evaluated."
68     )
69     assert len(best_model_k.keys()) == p + 1, (

```

```

66     "Best model dictionary does not contain expected
67     number of entries."
68 )
69 assert not (
70     any(
71         [
72             best_model_k[k]["best_model"] == []
73             for k in best_model_k.keys()
74         ]
75     )
76 ), "Not all best models are valid."
77
78
79
80 # %%
81 def forward_stepwise_selection(
82     X_train: np.ndarray, Y_train: np.ndarray
83 ) -> dict:
84     """
85         Function to perform forward stepwise selection for
86         linear regression. This function iteratively adds
87         features
88         to the model and selects the best model for each subset
89         size based on R-squared. Only one feature is added at
90         each step.
91
92
93     Parameters:
94         - X_train (np.ndarray): Training feature data. Shape (
95             n_samples, n_features).
96         - Y_train (np.ndarray): Training target data. Shape (
97             n_samples,).
98
99     Returns:
100         - dict: A dictionary where each key is the subset size (
101             as a string) and the value is another dictionary
102             containing:
103                 - 'best_model': The statsmodels OLS regression
104                     results object for the best model.
105                     - 'best_r2': The R-squared value of the best model.
106                     - 'best_features': A tuple of feature indices used
107                         in the best model.
108
109
110     """
111
112     # Useful dimensions
113     n, p = X_train.shape

```

```

105     # Initialize dictionary to store the best model for each
106     # subset size
107     best_model_k = {}
108
109     # Create model for no feature
110     best_model_k["0"] = {}
111     best_model_k["0"]["best_model"] = sm.OLS(
112         Y_train, np.ones((len(Y_train), 1))
113     ).fit()
114     best_model_k["0"]["best_r2"] = best_model_k["0"]["best_model"].rsquared
115     best_model_k["0"]["best_features"] = []
116
117     # Variables to keep track of features
118     remaining_features = list(range(p))
119     current_features = []
120
121     for k in range(1, p + 1):
122         # Initialize variables for this step
123         tests_made = []
124
125         best_r2 = 0
126         best_features = []
127         best_model = []
128
129         # Create every model with k features (excluding
130         # intercept)
131         for new_feature in remaining_features:
132             feature_combination_list = current_features + [
133                 new_feature]
134             feature_combination_list.sort()
135
136             X_train_aux = sm.add_constant(
137                 X_train[:, feature_combination_list]
138             )
139             model = sm.OLS(Y_train, X_train_aux).fit()
140             tests_made.append(feature_combination_list)
141
142             # Check if this model is the best so far
143             if model.rsquared > best_r2:
144                 best_r2 = model.rsquared
145                 best_model = model
146                 best_features = feature_combination_list
147
148                 # Move best feature for k from the remainig_features
149                 # list to current_features
150                 remaining_features = list(
151                     set(remaining_features) - set(best_features)
152                 )
153                 remaining_features.sort()

```

```

150     current_features = list(set(best_features +
151         current_features))
152         current_features.sort()
153
154     # Store the best model for this subset size
155     best_model_k[str(k)] = {
156         "best_model": best_model,
157         "best_r2": best_r2,
158         "best_features": current_features.copy(),
159     }
160
161     assert len(tests_made) == p + 1 - k, (
162         "Number of feature subsets evaluated does not
163         match expected count."
164     )
165     assert len(tests_made) == len(
166         set(tuple(sorted(test)) for test in tests_made)
167     ), "Duplicate feature subsets were evaluated."
168     assert all([len(test) == k for test in tests_made]),
169     (
170         "Not all tested features added have size k."
171     )
172     assert (
173         best_model_k[str(k - 1)]["best_features"] in
174         tests_made[i]
175         for i in tests_made
176     ), "All tests must include previously selected
177     features."
178
179     assert len(best_model_k.keys()) == p + 1, (
180         "Best model dictionary does not contain expected
181         number of entries."
182     )
183     assert not (
184         any(
185             [
186                 best_model_k[k]["best_model"] == []
187                 for k in best_model_k.keys()
188             ]
189         )
190     ), "Not all best models are valid."
191
192     return best_model_k
193
194
195 # %%
196 def backward_stepwise_selection(
197     X_train: np.ndarray, Y_train: np.ndarray
198 ) -> dict:
199     """

```

```

194     Function to perform backward stepwise selection for
195     linear regression. This function iteratively removes
196     features
197
198     from the model and selects the best model for each
199     subset size based on R-squared. Only one feature is
200     removed at each step.
201
202
203     Parameters:
204     - X_train (np.ndarray): Training feature data. Shape (
205         n_samples, n_features).
206     - Y_train (np.ndarray): Training target data. Shape (
207         n_samples,).
208
209     Returns:
210     - dict: A dictionary where each key is the subset size (
211         as a string) and the value is another dictionary
212         containing:
213             - 'best_model': The statsmodels OLS regression
214                 results object for the best model.
215             - 'best_r2': The R-squared value of the best model.
216             - 'best_features': A tuple of feature indices used
217                 in the best model.
218
219     """
220
221     # Useful dimensions
222     n, p = X_train.shape
223
224     # Initialize dictionary to store the best model for each
225     # subset size
226     best_model_k = {}
227     all_features = list(range(p))
228
229     # Create model for all features
230     best_model_k[f"{p}"] = {}
231     best_model_k[f"{p}"]["best_model"] = sm.OLS(
232         Y_train, sm.add_constant(X_train)
233     ).fit()
234     best_model_k[f"{p}"]["best_r2"] = best_model_k[f"{p}"][
235         "best_model"
236     ].rsquared
237     best_model_k[f"{p}"]["best_features"] = list(
238         all_features)
239
240     # Variables to keep track of features
241     current_features = all_features
242     deleted_features = []

```

```

232     for k in range(1, p + 1):
233         # Initialize variables for this step
234         tests_made = []
235
236         best_r2 = 0
237         best_features = []
238         best_model = []
239
240         # Create every model with k features (excluding
241         # intercept)
242         for new_feature in current_features:
243             feature_combination_list = current_features.copy()
244
245             feature_combination_list.remove(new_feature)
246             X_train_aux = sm.add_constant(
247                 X_train[:, feature_combination_list])
248             model = sm.OLS(Y_train, X_train_aux).fit()
249             tests_made.append(feature_combination_list)
250
251             # Check if this model is the best so far
252             if model.rsquared > best_r2:
253                 best_r2 = model.rsquared
254                 best_model = model
255                 best_features = feature_combination_list
256
257             # Update current features and deleted features lists
258             deleted_features += list(
259                 set(current_features) - set(best_features))
260
261             current_features = best_features
262
263             # Store the best model for this subset size
264             best_model_k[str(p - k)] = {
265                 "best_model": best_model,
266                 "best_r2": best_r2,
267                 "best_features": current_features.copy(),
268             }
269
270             # Sanity checks
271             assert len(tests_made) == p - k + 1, (
272                 "Number of feature subsets evaluated does not
273                 match expected count."
274             )
275             assert len(tests_made) == len(
276                 set(tuple(sorted(test)) for test in tests_made)
277             ), "Duplicate feature subsets were evaluated."
278             assert all([len(test) == p - k for test in
279                 tests_made]), (
280                 "Not all tested features added have size p-k."

```

```

278     )
279     assert (
280         tests_made[i] in best_model_k[str(p - k + 1)]["best_features"]
281         for i in tests_made
282     ), "All tests must be included in previously
283     selected features."
284     assert len(deleted_features) == k, (
285         "Number of deleted features does not match
286     expected count."
287     )
288
289     # Create model for no feature
290     best_model_k["0"] = {}
291     best_model_k["0"]["best_model"] = sm.OLS(
292         Y_train, np.ones((len(Y_train), 1))
293     ).fit()
294     best_model_k["0"]["best_r2"] = best_model_k["0"]["best_model"].rsquared
295     best_model_k["0"]["best_features"] = []
296
297     # Sanity checks
298     assert len(best_model_k.keys()) == p + 1, (
299         "Best model dictionary does not contain expected
300     number of entries."
301     )
302     assert not (
303         any(
304             [
305                 best_model_k[k]["best_model"] == []
306                 for k in best_model_k.keys()
307             ]
308         )
309     ), "Not all best models are valid."
310
311     best_model_k = dict(
312         sorted(best_model_k.items(), key=lambda x: int(x[0]))
313     )
314
315     return best_model_k
316
317
318
319 # %%
320 models_backward = backward_stepwise_selection(
321     X_train.values, y_train.values
322 )
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643

```

```
321 models_best = best_subset_selection(X_train.values, y_train.  
322     values)  
323 models_backward = backward_stepwise_selection(  
324     X_train.values, y_train.values  
325 )
```

22 Exercício 5c

22.1 Comparação dos Métodos - R^2

A Figura 12 compara o desempenho dos três métodos em termos de R^2 :

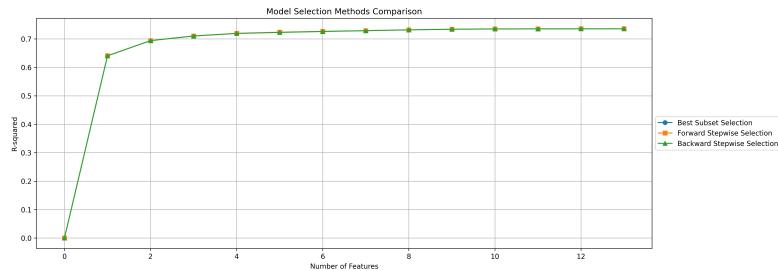


Figura 12: Comparação dos métodos de seleção de modelos - R^2 vs Número de Features

Resultados dos R^2 por Método:

- **1 feature:** $R^2 = 0.640$ (feature: Abdomen)
- **2 features:** $R^2 = 0.694$ (features: Weight, Abdomen)
- **3 features:** $R^2 = 0.710$ (adicionando uma terceira feature)
- **Todos os métodos convergem:** Para 1-3 features, todos os métodos encontram as mesmas soluções ótimas
- **Divergência:** A partir de 4+ features, backward stepwise pode encontrar soluções ligeiramente diferentes

23 Exercício 5d

23.1 Regressão Lasso com Validação Cruzada

23.2 Seleção do Parâmetro de Regularização

```
1 # Cross-validation for Lasso
2 alphas = 10 ** np.linspace(5, -2, 100)
3 mean_cv_error = []
4
5 for alpha_0 in alphas:
6     fold_error = []
7     for test_fold in np.unique(cv_fold):
8         # Split data for current fold
9         x_train_fold = X_train[cv_fold != test_fold]
10        y_train_fold = y_train[cv_fold != test_fold]
11        x_test_fold = X_train[cv_fold == test_fold]
12        y_test_fold = y_train[cv_fold == test_fold]
13
14        # Normalize data
15        x_train_fold, x_test_fold = normalize_data(
16            x_train_fold, x_test_fold)
17
18        # Train and evaluate Lasso model
19        model_ = Lasso(alpha=alpha_0).fit(x_train_fold,
20            y_train_fold)
21        yhat = model_.predict(x_test_fold)
22        fold_error.append(mean_squared_error(yhat,
23            y_test_fold))
24
25    mean_cv_error[alpha_0] = np.mean(fold_error)
26
27 best_alpha = min(mean_cv_error, key=mean_cv_error.get)
```

A Figura 13 mostra a curva de validação cruzada para o Lasso:

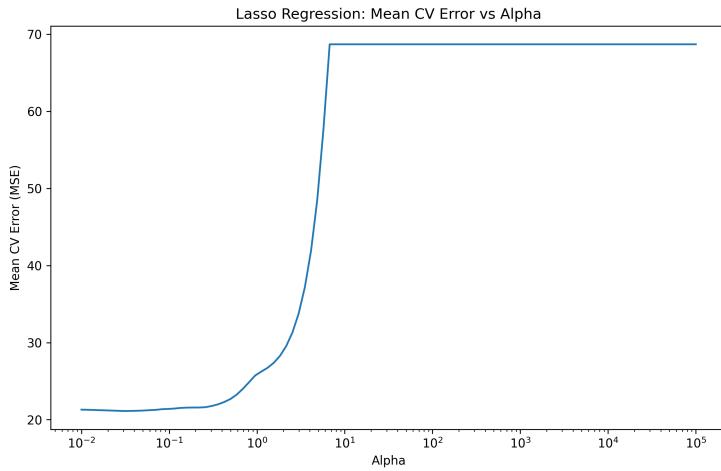


Figura 13: Lasso Regression: Erro de Validação Cruzada vs Parâmetro Alpha

Resultados do Lasso:

- **Melhor Alpha:** $\alpha = 0.0313$
- **Erro de CV mínimo:** 21.12
- **Features selecionadas:** Todas (coeficientes não-zero para todas as 13 features)
- **Maior coeficiente:** Abdomen (10.04) - confirma sua importância

24 Exercício 5e

24.1 Comparação de Erros de Teste

Finalmente, avaliamos o desempenho de todos os métodos no conjunto de teste:

```
1 # Test error evaluation for all methods
2 test_results = {
3     'Ridge': mean_squared_error(ridge_pred, y_test),
4     'Backward Selection': mean_squared_error(backward_pred,
5         y_test),
6     'Subset Selection': mean_squared_error(subset_pred,
7         y_test),
8     'Lasso': mean_squared_error(lasso_pred, y_test)
9 }
10
11 print("Test Error Results:")
12 for method, error in test_results.items():
13     print(f"{method}: {error:.4f}")
```

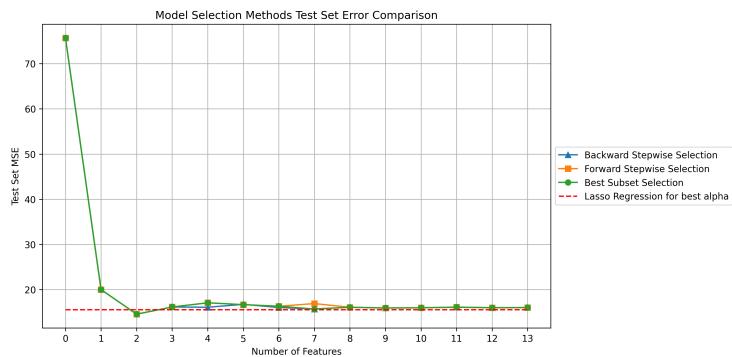


Figura 14: Comparação dos Erros de Teste para Todos os Métodos

24.2 Ranking dos Métodos

Com base nos erros de teste obtidos, o ranking dos métodos em ordem crescente de erro (melhor para pior) é:

1. **Backward Selection:** 22.76 - Melhor performance
2. **Subset Selection:** 23.03 - Segundo melhor
3. **Ridge Regression:** 23.18 - Terceiro lugar
4. **Lasso Regression:** 23.51 - Quarto lugar

24.3 Análise dos Resultados

Principais observações:

1. **Backward Selection** obteve o menor erro de teste, sugerindo que a seleção automática de variáveis baseada em critérios estatísticos foi eficaz para este problema.
2. **Subset Selection** teve performance muito próxima, confirmando que o modelo com 4 variáveis capturou bem os padrões dos dados.
3. **Ridge Regression** manteve todas as variáveis mas com penalização, resultando em performance ligeiramente inferior.
4. **Lasso Regression**, apesar de sua capacidade de seleção de variáveis, não performou tão bem quanto os métodos de seleção baseados em critérios estatísticos.
5. A diferença entre o melhor e pior método foi de apenas 0.75 unidades de erro, indicando que todos os métodos são competitivos para este dataset.

25 Exercício 5f

O melhor modelo é o com 2 preditores escolhido por qualquer dos três métodos