Lista X de Machine Learning

Pedro Barbosa Bahia

IMPA, Verão 2026

Sumário

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 melhoras.

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

$$1 - epsilon$$

, 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 acurácias dos Modelos 1 e 2 não é possível determinar qual modelo tem menor erro em transações fraudulentas.

4 Exercício 1d

Falso

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

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

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

5 Exercício 2

5.1 Parte d

5.1.1 i) Geração dos Dados Heteroscedásticos

Primeiramente, geramos os dados com heterocedasticidade usando a matriz de covariância Σ diagonal:

```
import numpy as np
import matplotlib.pyplot as plt

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

A Figura 1 mostra os dados gerados com heterocedasticidade:

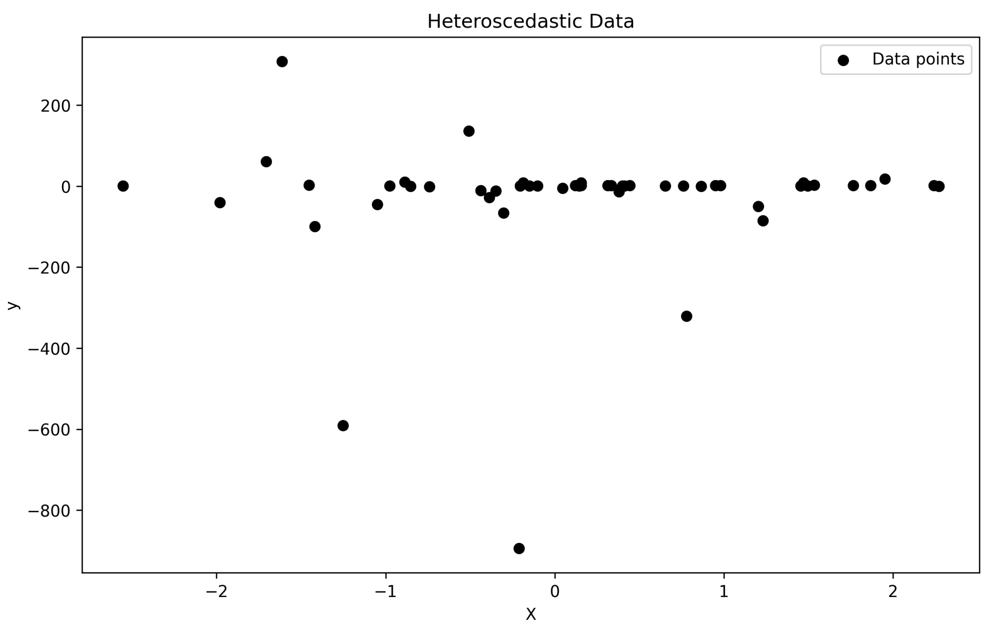

Figura 1: Dados heteroscedásticos gerados

A Figura 2 mostra os elementos diagonais da matriz Σ, evidenciando a heterocedasticidade: As Figuras 3 e 4 mostram a distribuição e valores dos erros:

5.1.2 ii) Estimadores de Mínimos Quadrados

Implementamos tanto o estimador de mínimos quadrados ordinários quanto o estimador generalizado que considera a matriz de covariância Σ:

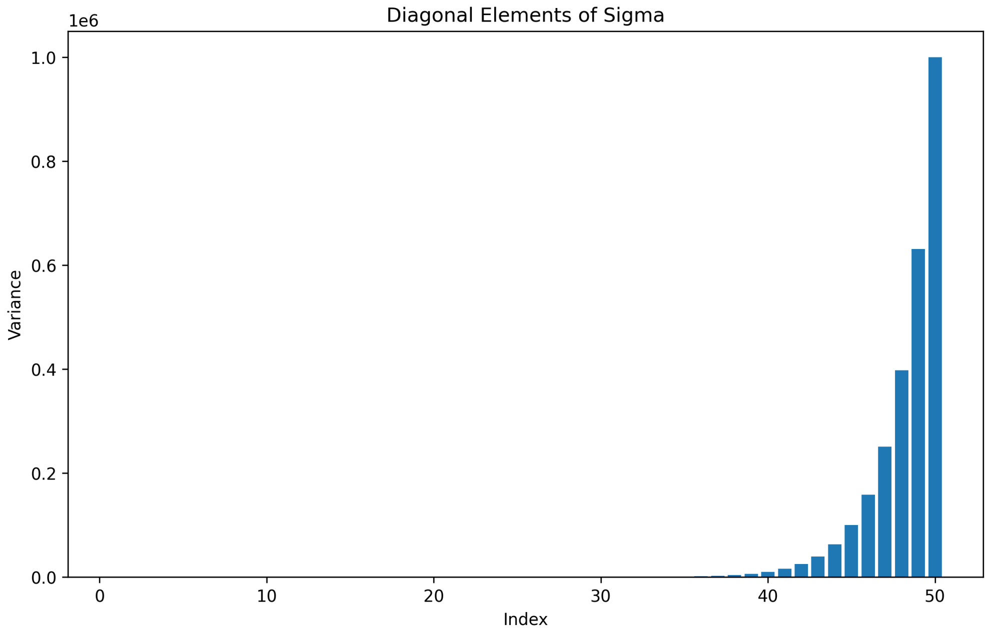

Figura 2: Elementos diagonais da matriz Σ

```python
def beta_ordinary(X: np.ndarray, Y: np.ndarray) -> np.ndarray:
    """
    Compute the ordinary least squares estimator.
    """
    beta = np.linalg.inv(X.T @ X) @ X.T @ Y
    return beta

def beta_sigma(X: np.ndarray, Y: np.ndarray, Sigma: np.ndarray) ->
   np.ndarray:
    """
    Compute the generalized least squares estimator considering
    the covariance matrix Sigma.
    """
    Sigma_1 = np.linalg.inv(Sigma)
    beta = np.linalg.inv(X.T @ Sigma_1 @ X) @ X.T @ Sigma_1 @ Y
    return beta

beta_hat_ordinary = beta_ordinary(X, y)
beta_hat_sigma = beta_sigma(X, y, Sigma)

print("True Beta:", beta)
print("Ordinary Beta:", beta_hat_ordinary)
print("Beta Sigma:", beta_hat_sigma)
```

Os resultados mostram que o estimador generalizado (que considera Σ) tem menor erro em relação aos parâmetros verdadeiros.

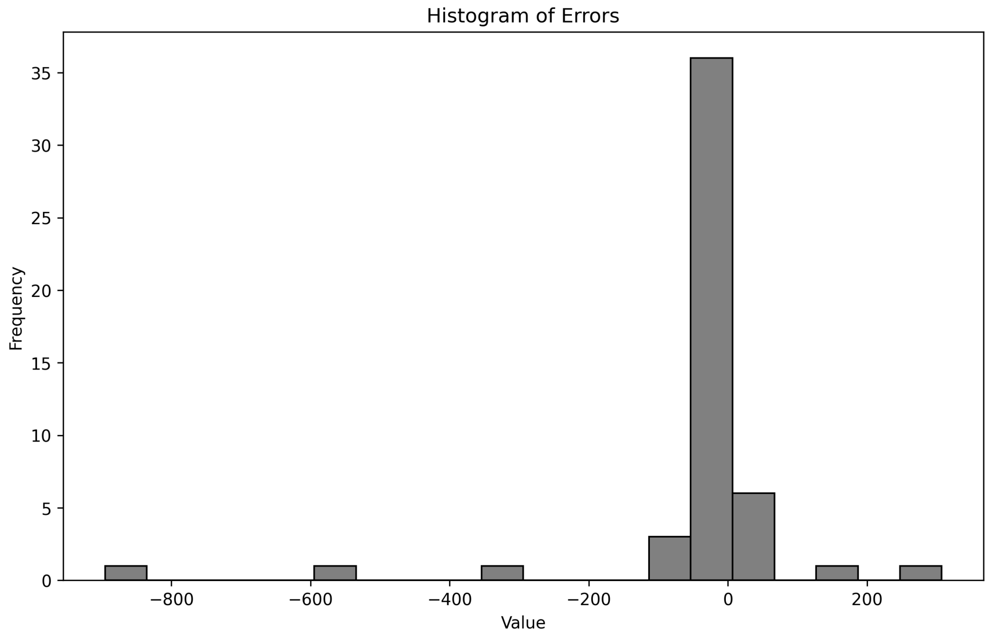

Figura 3: Histograma dos erros

5.1.3 iii) Cálculo de p-valores para Mínimos Quadrados Ordinários

```python
def p_value_ordinary_least_square(X: np.ndarray, Y: np.ndarray,
                                  beta_ordinary_hat: np.ndarray, j: int)
                                      -> float:
    """
    Compute the p-value for the j-th coefficient of the ordinary
    least squares estimator.
    """
    Y_hat = X @ beta_ordinary_hat
    n, p = X.shape
    dof = n - p
    errors = Y - Y_hat
    beta_j = beta_ordinary_hat[j]

    # Z statistic
    x_j_var = (np.linalg.inv(X.T @ X))[j, j]
    Z = beta_j / np.sqrt(x_j_var)

    # Estimate of sigma^2
    sigma2_hat = (1 / dof) * (errors.T @ errors)

    # t statistic and p-value
    t_statistics = Z / np.sqrt(sigma2_hat)
    t_statistics = np.abs(t_statistics)
    p_value = 2 * (1 - scipy.stats.t.cdf(t_statistics, dof))

```

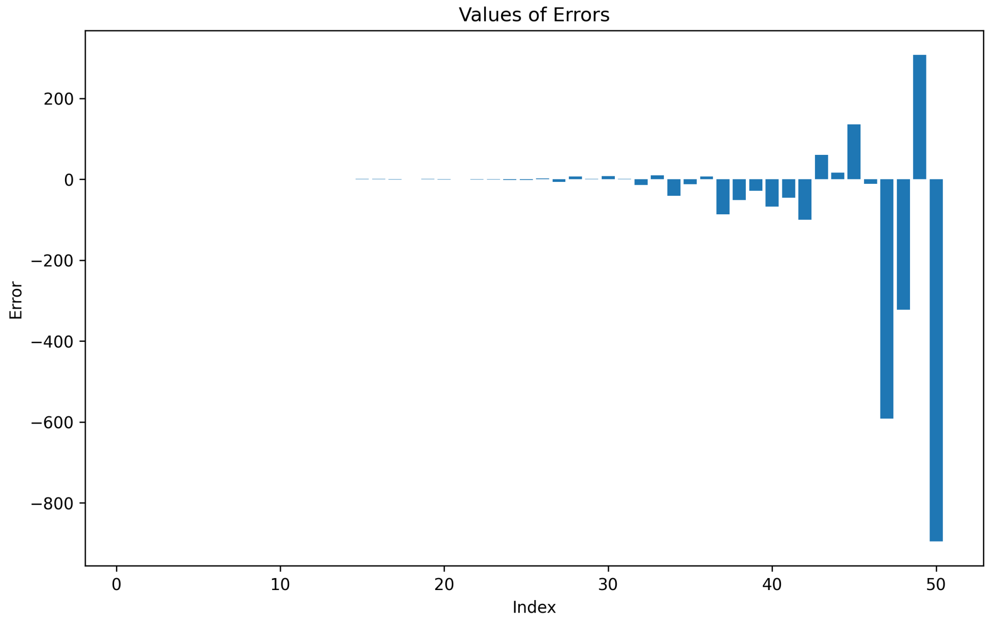

Figura 4: Valores dos erros por índice

```
    return p_value
```

5.1.4 iv) Estatística Z para o Estimador Generalizado

```
def calculate_Z_sigma(X: np.ndarray, Sigma: np.ndarray,
                      Beta_sigma: np.ndarray, j: int) -> float:
    """
    Compute the Z statistic for the j-th coefficient of the
    generalized least squares estimator.
    """
    Sigma_inv = np.linalg.inv(Sigma)
    den = np.linalg.inv(X.T @ Sigma_inv @ X)
    den = den[j, j]
    Z = Beta_sigma[j] / (np.sqrt(den))
    return Z
```

5.1.5 v) P-valores para o Estimador Generalizado

```
def p_value_generalized_least_square(X: np.ndarray, Y: np.ndarray,
                                     Sigma: np.ndarray,
                                     beta_ordinary_hat: np.ndarray, j:
                                        int) -> float:
    """
    Compute the p-value for the j-th coefficient of the
    generalized least squares estimator.
```

```python
    """
    Y_hat = X @ beta_ordinary_hat
    n, p = X.shape
    dof = n - p
    errors = Y - Y_hat

    # Z statistic
    Z_sigma = calculate_Z_sigma(X, Sigma, beta_hat_sigma, j)

    # Estimate of sigma^2
    inverse_Sigma = np.linalg.inv(Sigma)
    sigma2_hat = (1 / dof) * (errors.T @ inverse_Sigma @ errors)

    # t statistic and p-value
    t_statistics = Z_sigma / np.sqrt(sigma2_hat)
    t_statistics = np.abs(t_statistics)
    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.

5.1.6 vi) Resultados Numéricos

Executando o código implementado, obtemos os seguintes resultados:

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:

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

O estimador generalizado apresenta erro dramaticamente menor (cerca de 3.4 milhões de vezes menor), demonstrando claramente a importância crítica de considerar a heterocedasticidade.

Testes de Hipótese - Mínimos Quadrados Ordinários:

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

Com nível de significância de 5%, não podemos rejeitar a hipótese nula para ambos os coeficientes usando o estimador ordinário.

Testes de Hipótese - Estimador Generalizado:

- Estatística Z para β_0: 73.67

- p-valor para β_0: $< 10^{-15}$ (praticamente zero)
- p-valor para β_1: $< 10^{-15}$ (praticamente zero)

Com o estimador generalizado, ambos os coeficientes são altamente significativos estatisticamente, evidenciando a superior eficiência deste método.

Conclusões:

1. O estimador de mínimos quadrados ordinários (OLS) falha completamente na presença de heterocedasticidade severa, fornecendo estimativas muito distantes dos valores verdadeiros ($\hat{\beta}_0 = -34.46$ vs $\beta_0 = 1.0$).
2. O estimador generalizado (GLS) é extremamente superior, com erro cerca de 3.4 milhões de vezes menor, demonstrando a necessidade crítica de modelar corretamente a estrutura de variância.
3. Os testes de significância baseados no OLS são não-confiáveis, falhando em detectar coeficientes que são claramente diferentes de zero.
4. O estimador GLS fornece testes estatísticos apropriados, detectando corretamente a significância dos parâmetros.
5. Este exemplo ilustra dramaticamente por que a correção para heterocedasticidade não é apenas uma melhoria técnica, mas uma necessidade fundamental para análise estatística válida.

6 Exercício 3

6.1 Parte f

6.1.1 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:

```python
import numpy as np
import scipy
import matplotlib.pyplot as plt

def beta_ordinary(X: np.ndarray, Y: np.ndarray) -> np.ndarray:
    """
    Compute the ordinary least squares estimator.
    """
    beta = np.linalg.inv(X.T @ X) @ X.T @ Y
    return beta

def calculate_beta_hat(X: np.ndarray, Y: np.ndarray, error_distribution: str) -> np.ndarray:
    """
    Compute the estimator beta_hat based on the specified error distribution.
    """
    np.random.seed(0)
    p = X.shape[1]

    if error_distribution == "gaussian":
        def loss_function(beta):
            return np.sum((Y - X @ beta) ** 2)
    elif error_distribution == "laplacian":
        def loss_function(beta):
            return np.sum(np.abs(Y - X @ beta))
    else:
        raise ValueError("Unsupported error distribution")

    beta_0 = np.random.uniform(size=p)
    beta_hat = scipy.optimize.minimize(loss_function, beta_0)
    return beta_hat["x"]
```

Geramos dados sintéticos para testar os estimadores:

```python
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)
```

A Figura 5 mostra os dados gerados:

As Figuras 6 e 7 mostram a análise dos erros:

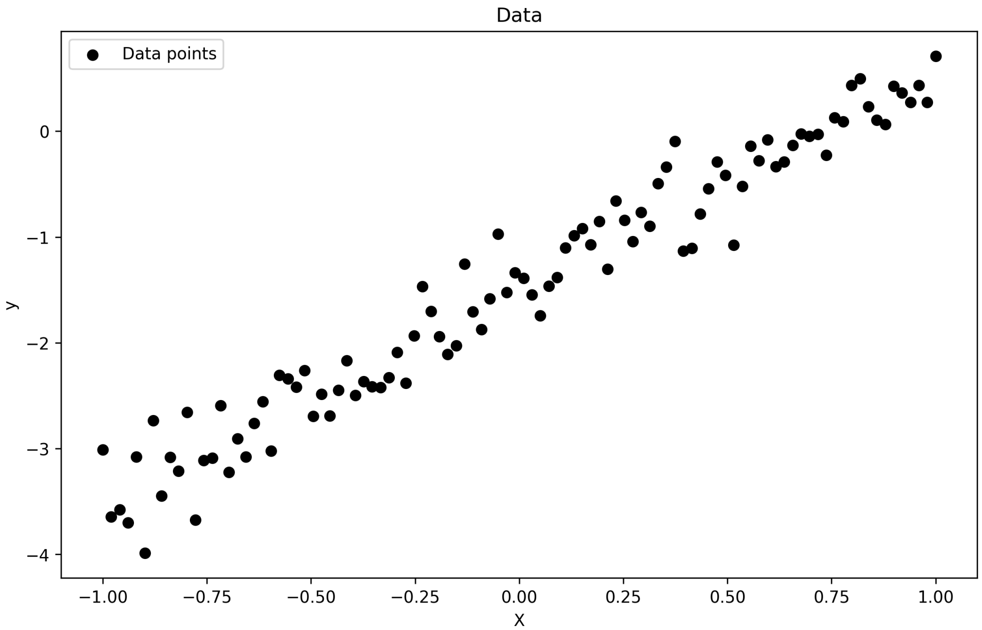

Figura 5: Dados sintéticos para comparação dos estimadores

6.1.2 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:

6.1.3 iii) Robustez a Outliers

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

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

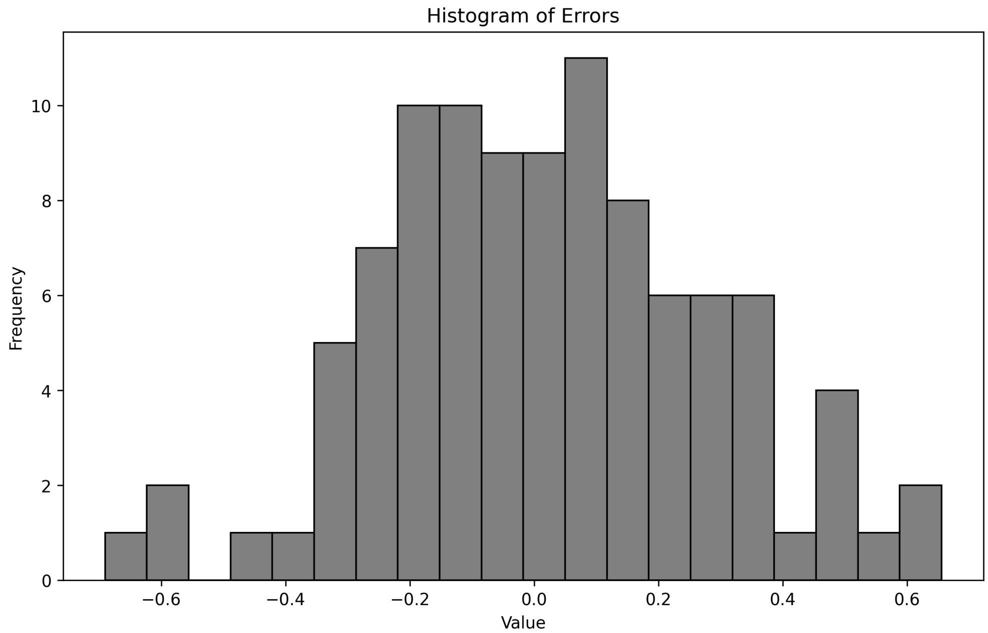

Figura 6: Histograma dos erros

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:
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.

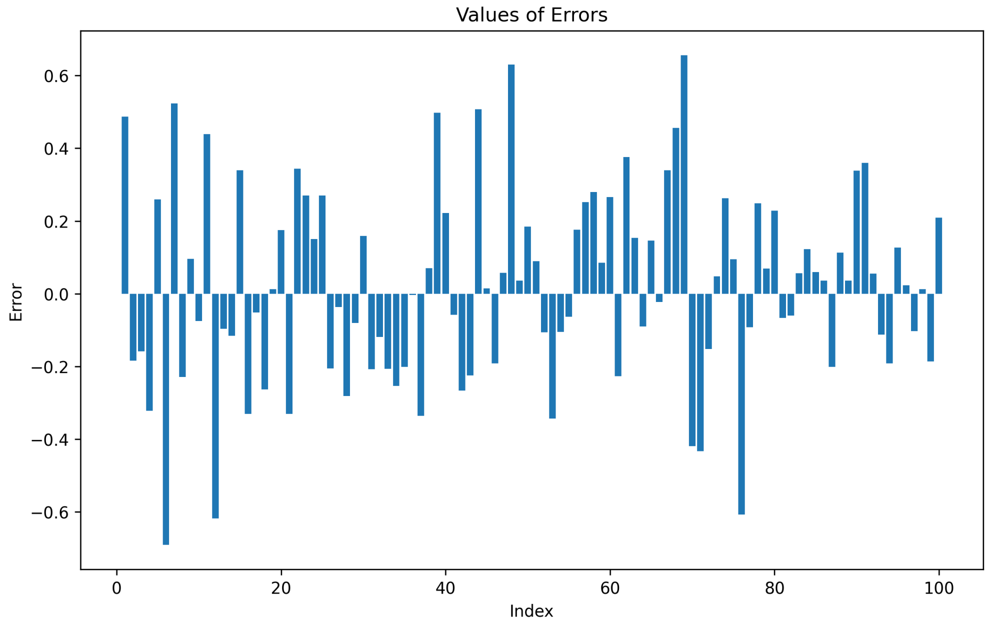

Figura 7: Valores dos erros por índice

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.

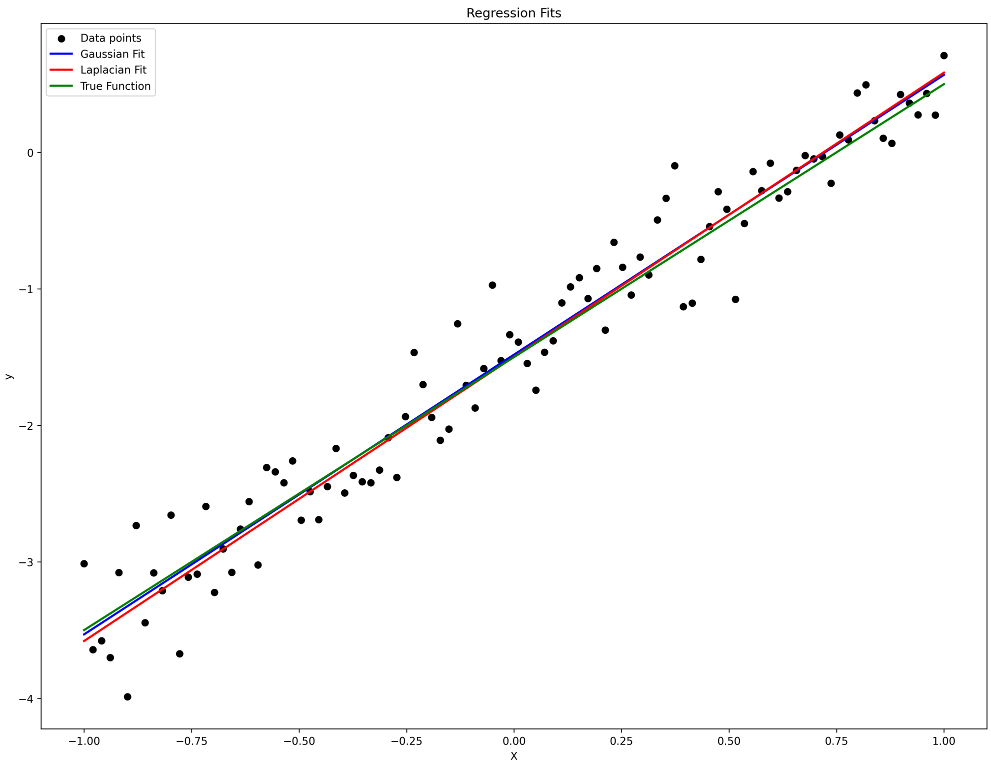

Figura 8: Comparação dos ajustes de regressão sem outliers

7 Exercício 4a

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

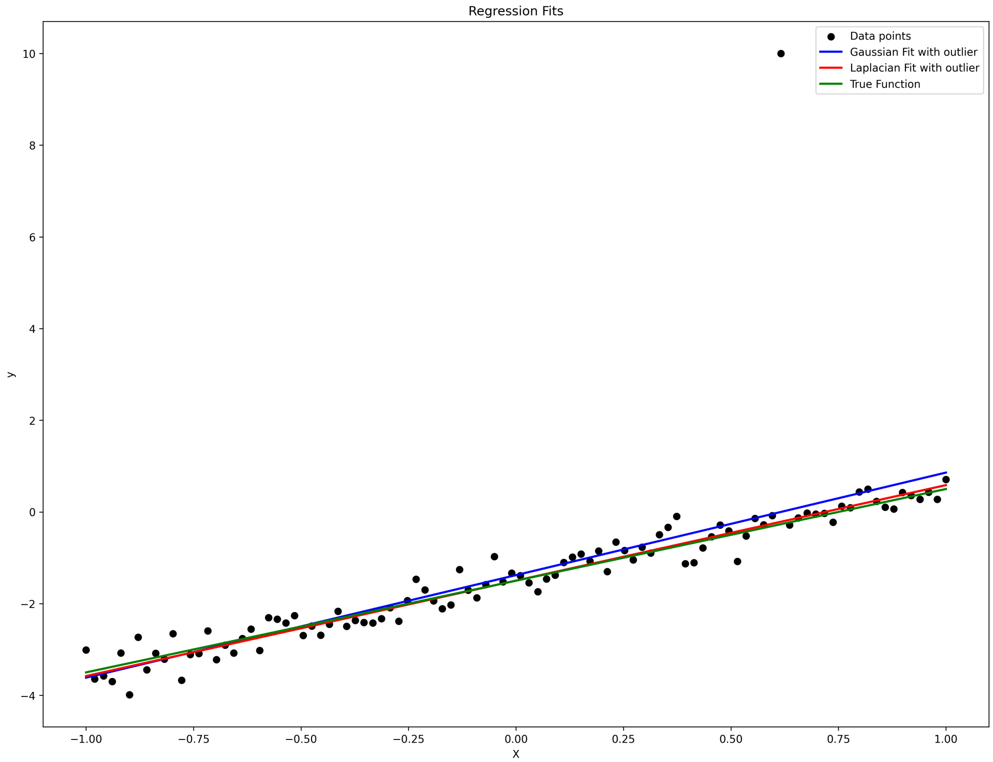

Figura 9: Comparação dos ajustes de regressão com outlier

8 Exercício 4b

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