Aprendizado de Máquina para Engenharia de Sistemas de Processos: Pré-processamento de Dados - limpeza de dados do processo

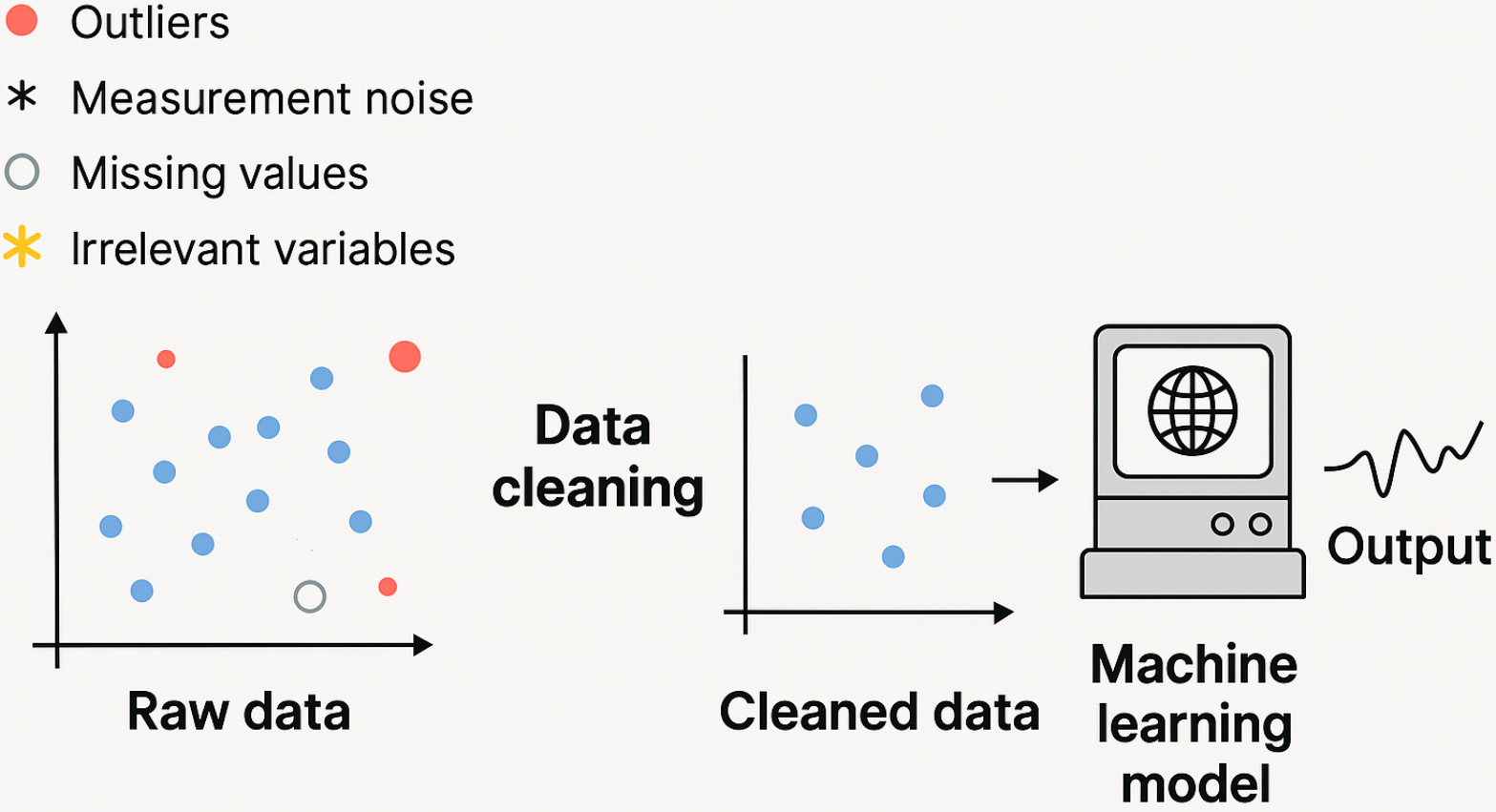
Elizabeth Alves de Oliveira
Prof. Dr. Luiz Gonzaga Sales Vasconcelos

Universidade Federal de Campina Grande - UFCG
Centro de Ciências e Tecnologia - CCT
Programa de Pós-Graduação em Engenharia Química - PPGEQ
Redes Neurais

Abril de 2025

Sumário

Introdução


Outliers
Measurement noise
Missing values
Irrelevant variables
Raw data
Data cleaning
Cleaned data
Machine learning model
Output

Objetivos

- Remoção de ruído de medição;
- Seleção de variáveis relevantes para desempenho aprimorado do modelo;
- Remoção de outliers em configurações univariadas e multivariadas;
- Técnicas comuns para lidar com dados ausentes.

Redução de ruído de sinal

- Medições de processo inevitavelmente apresentam ruído de alta frequência, que, se não for devidamente tratado, pode comprometer a precisão do modelo;
- Esse ruído pode distorcer a estimativa de parâmetros, introduzir erros e afetar negativamente as variáveis previstas, prejudicando a confiabilidade das previsões;
- Duas maneiras comuns de reduzir o ruído dos sinais de processo são suavizá-los usando médias móvel simples (SMA) e filtragem Savitzky-Golay (SG). A Figura 4.1 mostra os sinais de fluxo suavizados.

Redução de ruído de sinal

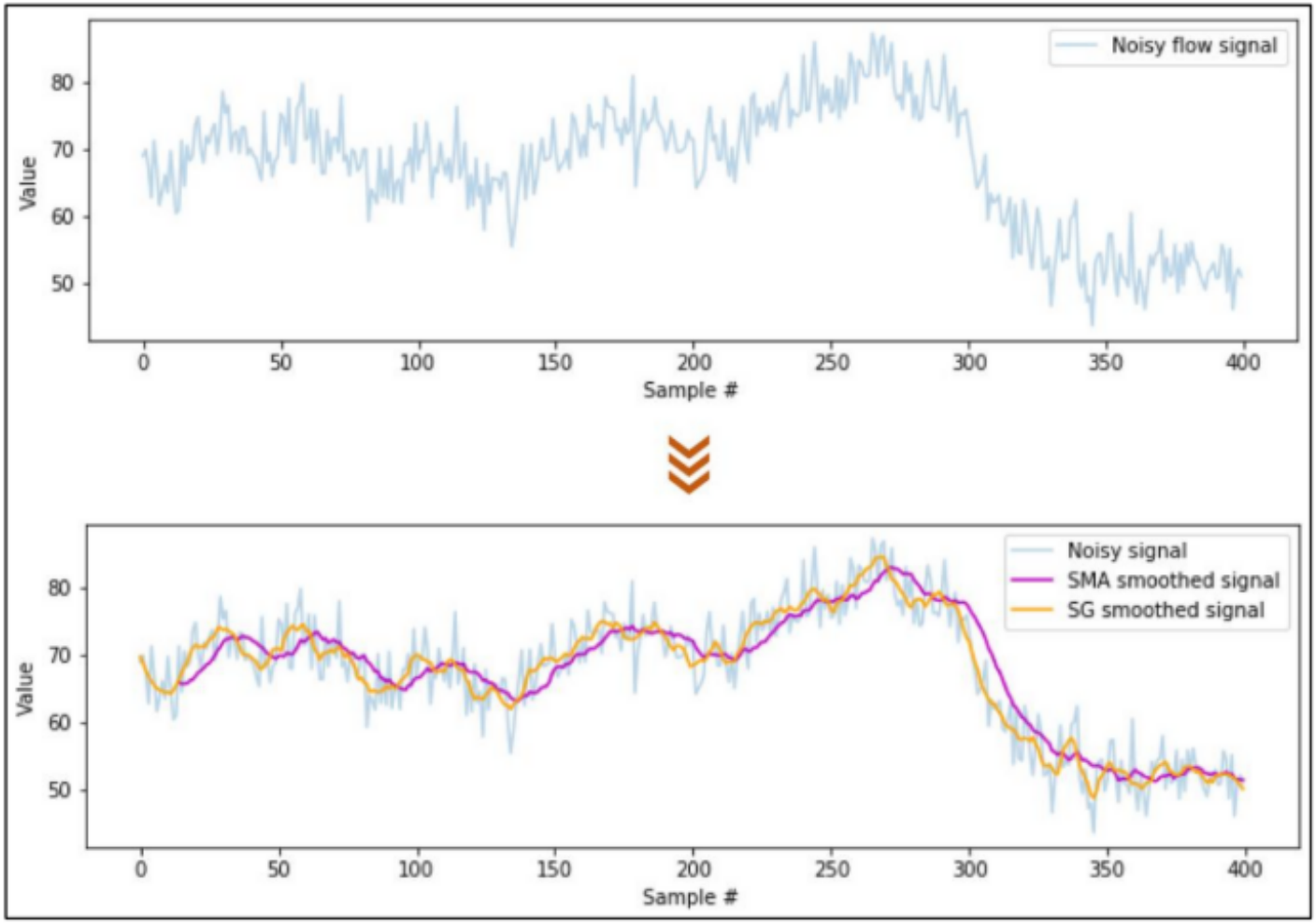


Figure 4.1: Time series signal smoothed by simple moving average (SMA) and SG filters

Redução de ruído de sinal

Filtro de média da janela móvel:

- A média de janela móvel suaviza uma variável x no tempo t ao calcular uma média ponderada de suas medições anteriores dentro de uma janela de tamanho finito ou infinito. A variante mais comum, **média móvel simples (SMA)**, utiliza uma média aritmética dos valores dentro da janela, conforme ilustrado abaixo:

$$x(t)_{smoothed} = \frac{\sum_{j=0}^{W-1} x(t-j)_{raw}}{W} \tag{1}$$

- Onde, W é o tamanho da janela;
- W maior alcança mais suavização.

Filtro SG:

- A filtragem Savitzky-Golay (SG) suaviza os dados ajustando um polinômio de ordem m a uma janela de tamanho ímpar centrada no ponto t*. O valor suavizado é obtido ao avaliar esse polinômio em t*, preservando características como picos e tendências, diferentemente de métodos convencionais de média móvel que podem distorcer detalhes do sinal.

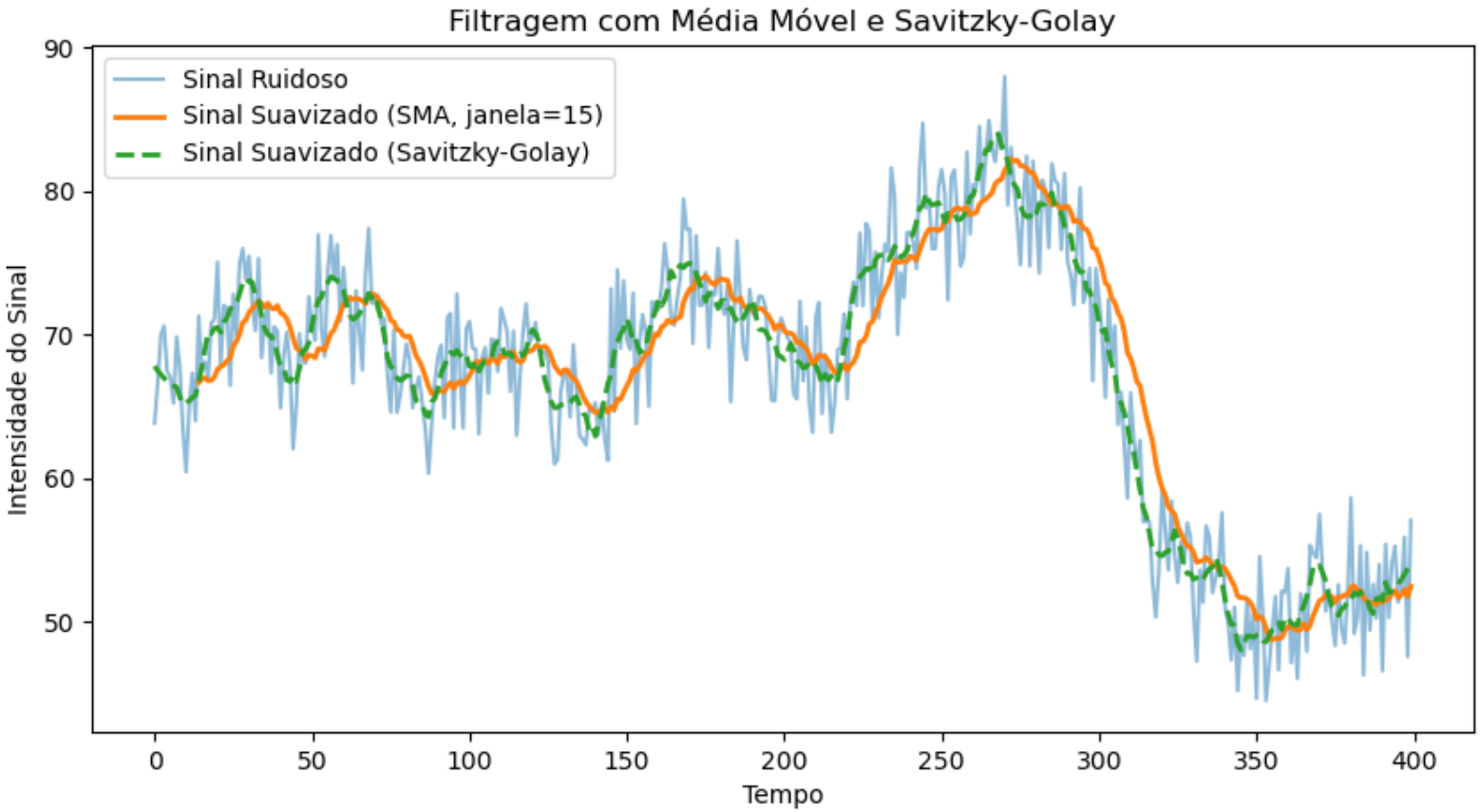
$$x(t^*)_{smoothed} = \sum_{k=1}^{m} b_k (t^*)^k \tag{2}$$

- Em que: b_k são os coeficientes polinomiais estimados.

Redução de ruído de sinal

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.signal import savgol_filter

# Carregar sinal ruidoso
file_path = "noisy_flow_signal.csv"  # Certifique-se de que o arquivo está no mesmo diretório do script
noisy_signal = np.loadtxt(file_path, delimiter=',')

# Aplicar média móvel (SMA)
window_size = 15
smoothed_signal_MA = pd.DataFrame(noisy_signal).rolling(window_size).mean().values

# Aplicar filtro de Savitzky-Golay
smoothed_signal_SG = savgol_filter(noisy_signal, window_length=15, polyorder=2)

# Plotar os sinais
plt.figure(figsize=(10, 5))
plt.plot(noisy_signal, label='Sinal Ruidoso', alpha=0.5)
plt.plot(smoothed_signal_MA, label=f'Sinal Suavizado (SMA, janela={window_size})', linewidth=2)
plt.plot(smoothed_signal_SG, label='Sinal Suavizado (Savitzky-Golay)', linewidth=2, linestyle='dashed')
plt.legend()
plt.xlabel('Tempo')
plt.ylabel('Intensidade do Sinal')
plt.title('Filtragem com Média Móvel e Savitzky-Golay')
plt.show()
```

Redução de ruído de sinal



Seleção de Variáveis / Seleção de Características

- A seleção de variáveis busca identificar as entradas mais relevantes para um modelo. Testar todas as combinações seria ideal, mas é inviável para muitos dados devido ao alto custo computacional. Por isso, métodos eficientes foram desenvolvidos para otimizar essa escolha.

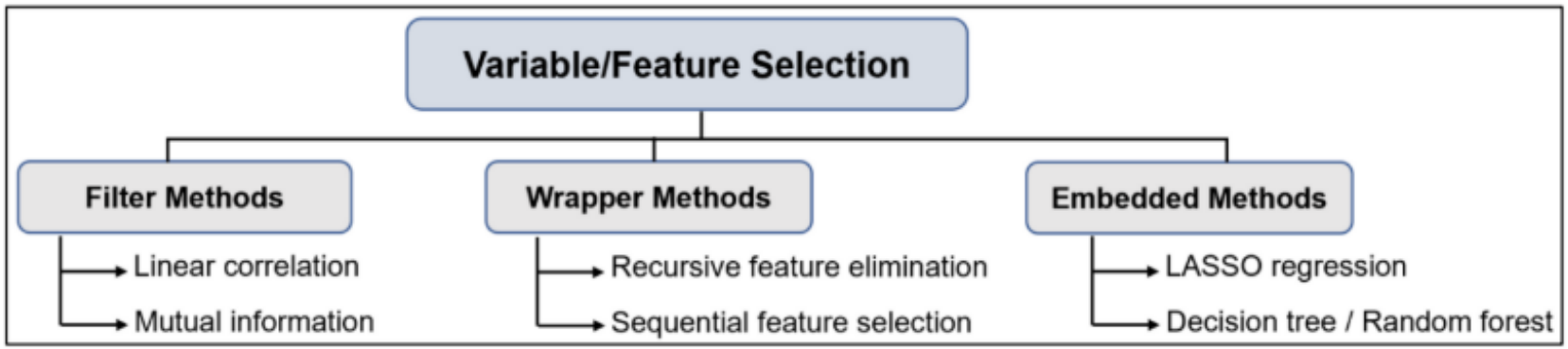


Figure 4.2: An overview of variable selection methods

Sumário

Métodos de filtro

- Na forma mais comum de métodos de filtro, a relação entre cada entrada e o alvo é estimada usando alguma medida estatística e então as entradas são classificadas de acordo com as forças de relacionamento estimadas. Uma vez classificadas, as variáveis mais bem classificadas podem ser escolhidas. A Figura 4.4 mostra uma visão geral da estratégia.

Figure 4.4: Filter method strategy for feature selection

Métodos de filtro: Coeficiente de correlação

- Coeficiente de Pearson

$$R_{xy} = \frac{\sum_i (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_i (x_i - \overline{x})^2 \sum_i (y_i - \overline{y})^2}} \tag{3}$$

- Informação mútua

$$MI(x, y) = \int \int p(x, y) log \frac{p(x, y)}{p(x)p(y)} dxdy \tag{4}$$

Métodos de filtro: Coeficiente de correlação

- Coeficiente de Pearson X Informação mútua

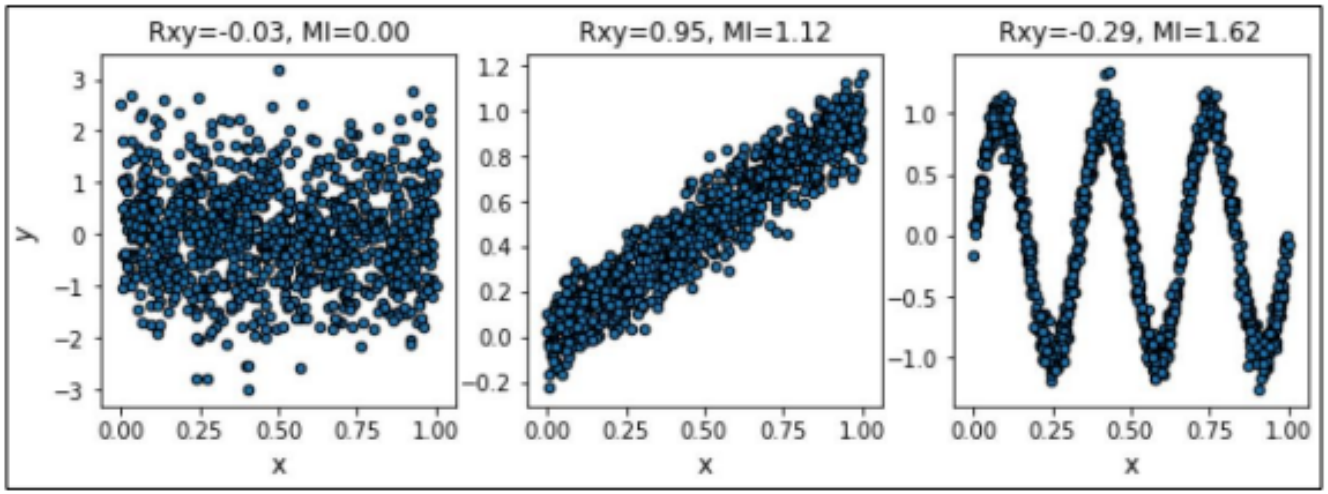
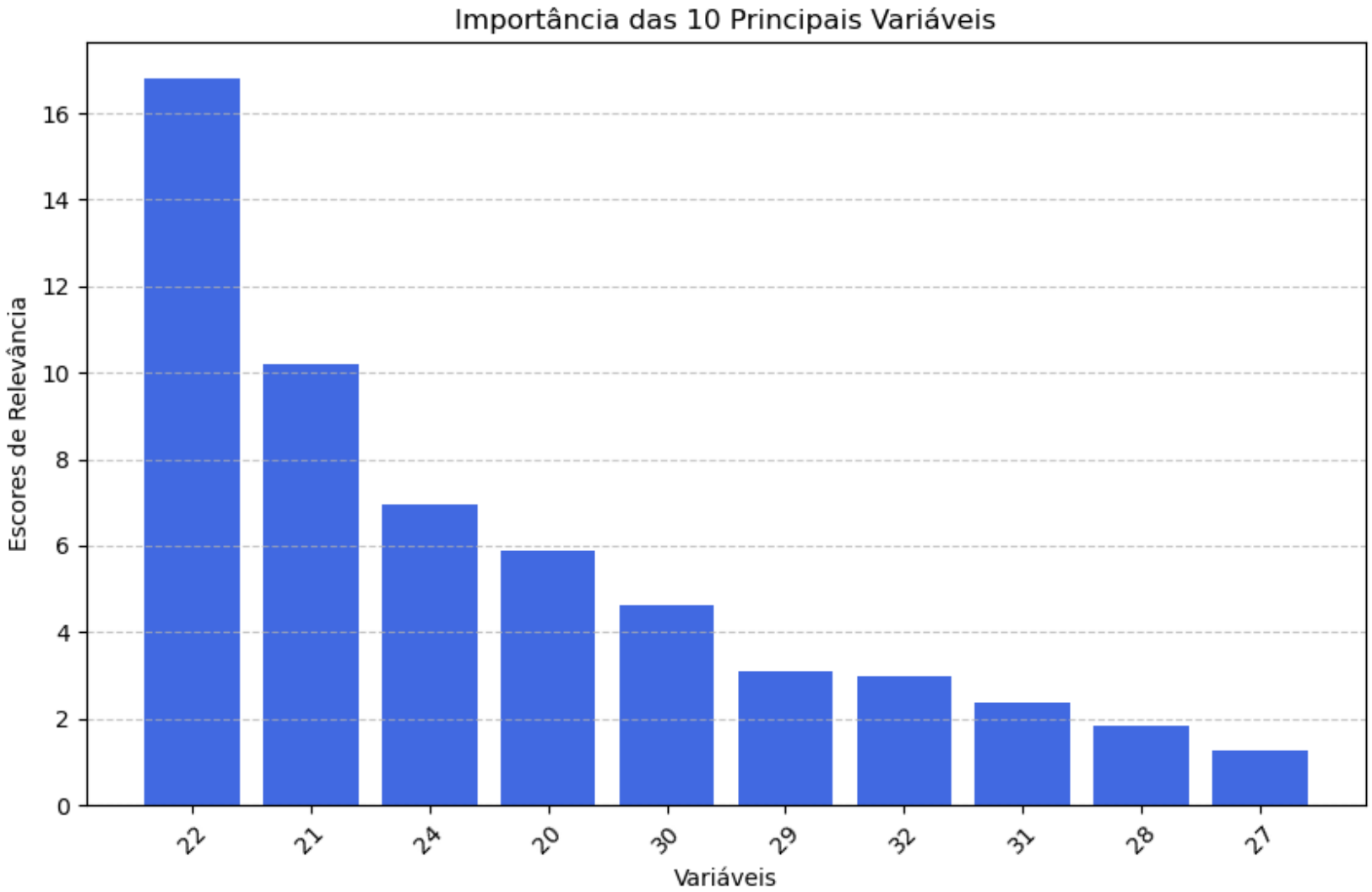


Figure 4.5: Linear correlation and mutual information metrics for zero, linear, and nonlinear dependencies between target and input

Seleção de Variáveis / Seleção de Características

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.preprocessing import StandardScaler

# Carregar os dados
VSdata = np.loadtxt('VSdata.csv', delimiter=',')

# Separar variável alvo (y) e variáveis preditoras (X)
y = VSdata[:, 0]
X = VSdata[:, 1:]

# Normalizar os dados para melhor análise estatística
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Seleção das 10 melhores variáveis com f_regression
k = 10  # Número de variáveis a selecionar
VSmodel = SelectKBest(f_regression, k=k).fit(X_scaled, y)

# Obter escores de importância das variáveis
input_scores = VSmodel.scores_

# Encontrar os 10 melhores preditores
top_k_indices = np.argsort(input_scores)[::-1][:k]
top_k_inputs = top_k_indices + 1  # Ajustando os índices para corresponder às colunas originais

# Reduzir X para apenas as variáveis selecionadas
X_relevant = VSmodel.transform(X_scaled)

# Plotando a importância das variáveis
plt.figure(figsize=(10, 6))
plt.bar(range(k), input_scores[top_k_indices], tick_label=top_k_inputs, color='royalblue')
plt.xlabel("Variáveis")
plt.ylabel("Escores de Relevância")
plt.title("Importância das 10 Principais Variáveis")
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

Seleção de Variáveis / Seleção de Características



Sumário

Método wrapper

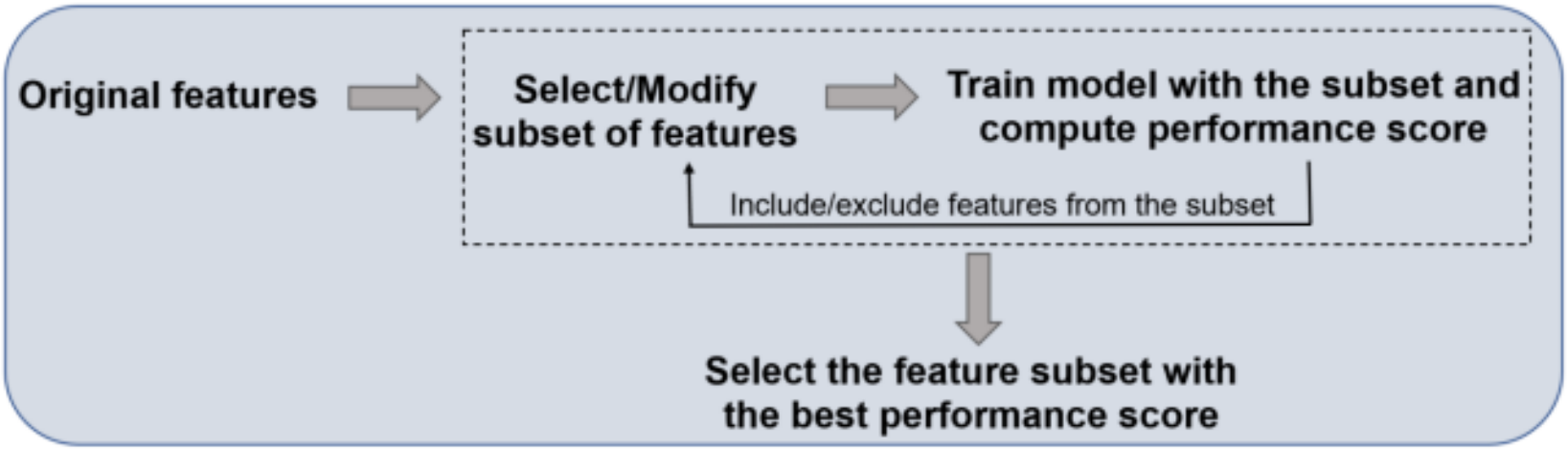


Figure 4.6: Wrapper method strategy for feature selection

Método wrapper

- **Método por eliminação de resursos recursivos (RFE)**: remove iterativamente as variáveis menos importantes com base em atributos do modelo, repetindo o processo até restarem apenas as mais relevantes ou até que a remoção prejudique o desempenho do modelo;
- **Seleção sequencial de recursos (SFS)**: Começa com um subconjunto vazio e adiciona, uma a uma, as variáveis que mais melhoram o modelo. O processo continua até atingir um número definido de variáveis ou até que novas adições não tragam benefícios significativos.

Método wrapper

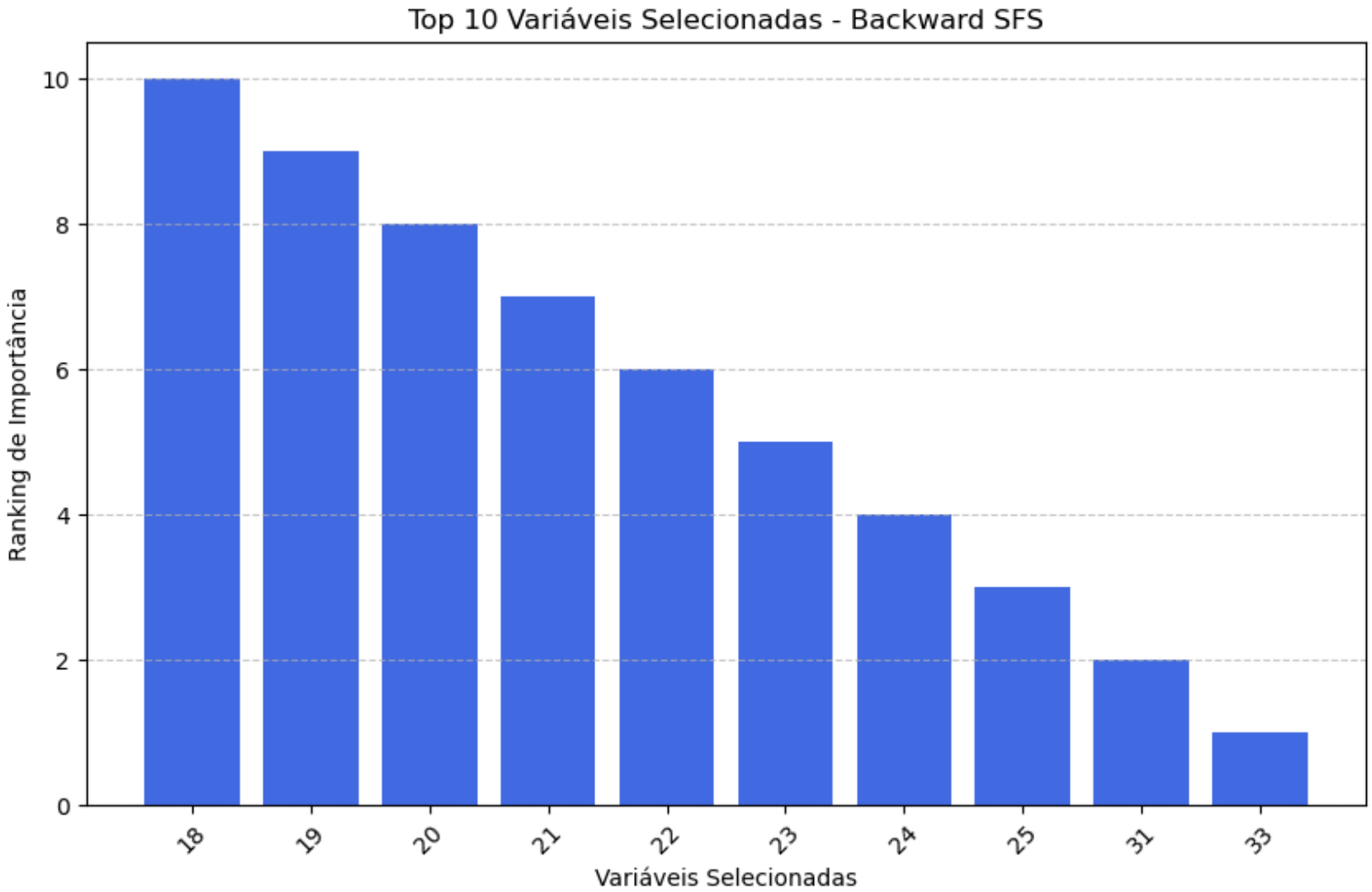
- Implementação do Sklearn

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.linear_model import Ridge

# Carregar os dados
VSdata = np.loadtxt('VSdata.csv', delimiter=',')

# Separar variável alvo (y) e variáveis preditoras (X)
y = VSdata[:, 0]
X = VSdata[:, 1:]

# Normalizar os dados
xscaler = StandardScaler()
X_scaled = xscaler.fit_transform(X)

yscaler = StandardScaler()
print("Antes da normalização:", y[:5])  # Exibe os primeiros valores
y_scaled = yscaler.fit_transform(y[:, None]).ravel()  # ravel() transforma de (n,1) para (n,)
print("Depois da normalização:", y_scaled[:5])  # Exibe os valores transformados

# Seleção de variáveis com Backward SFS usando Ridge Regression
k = 10  # Número de variáveis a selecionar
model = Ridge(alpha=1.0)  # Regularização para evitar overfitting

BSFS = SequentialFeatureSelector(model,
                                 n_features_to_select=k,
                                 direction='backward',
                                 cv=5).fit(X_scaled, y_scaled)
```

Seleção de Variáveis / Seleção de Características

```
# Obter índices das variáveis selecionadas
selected_indices = BSFS.get_support(indices=True)
selected_features = selected_indices + 1  # Ajuste para refletir os números originais das colunas

# Reduzir X para apenas as variáveis escolhidas
X_relevant = BSFS.transform(X)

# Visualizar as variáveis selecionadas
plt.figure(figsize=(10, 6))
plt.bar(range(k), np.arange(k)[::-1] + 1, tick_label=selected_features, color='royalblue')
plt.xlabel("Variáveis Selecionadas")
plt.ylabel("Ranking de Importância")
plt.title("Top 10 Variáveis Selecionadas - Backward SFS")
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

print("Rodando o código...")  # Mensagem para saber se a execução começou
# Exibir as variáveis escolhidas no terminal
print(f"Variáveis Selecionadas: {selected_features}")
```

Seleção de Variáveis / Seleção de Características



Sumário

Métodos incorporados/integrados

- Realizam a seleção de variáveis durante o próprio ajuste do modelo, eliminando a necessidade de um processo separado. Um exemplo é a **Regressão Lasso**, que atribui coeficientes zero a variáveis irrelevantes, removendo-as automaticamente. Outro exemplo são as **árvores de decisão** e **florestas aleatórias**, que calculam diretamente a importância de cada variável após o treinamento, permitindo identificar quais recursos mais influenciam as previsões;
- Esses métodos são computacionalmente menos dispendiosos do que métodos wrapper e funcionam melhor quando o número de amostras é muito maior do que o número de entradas.

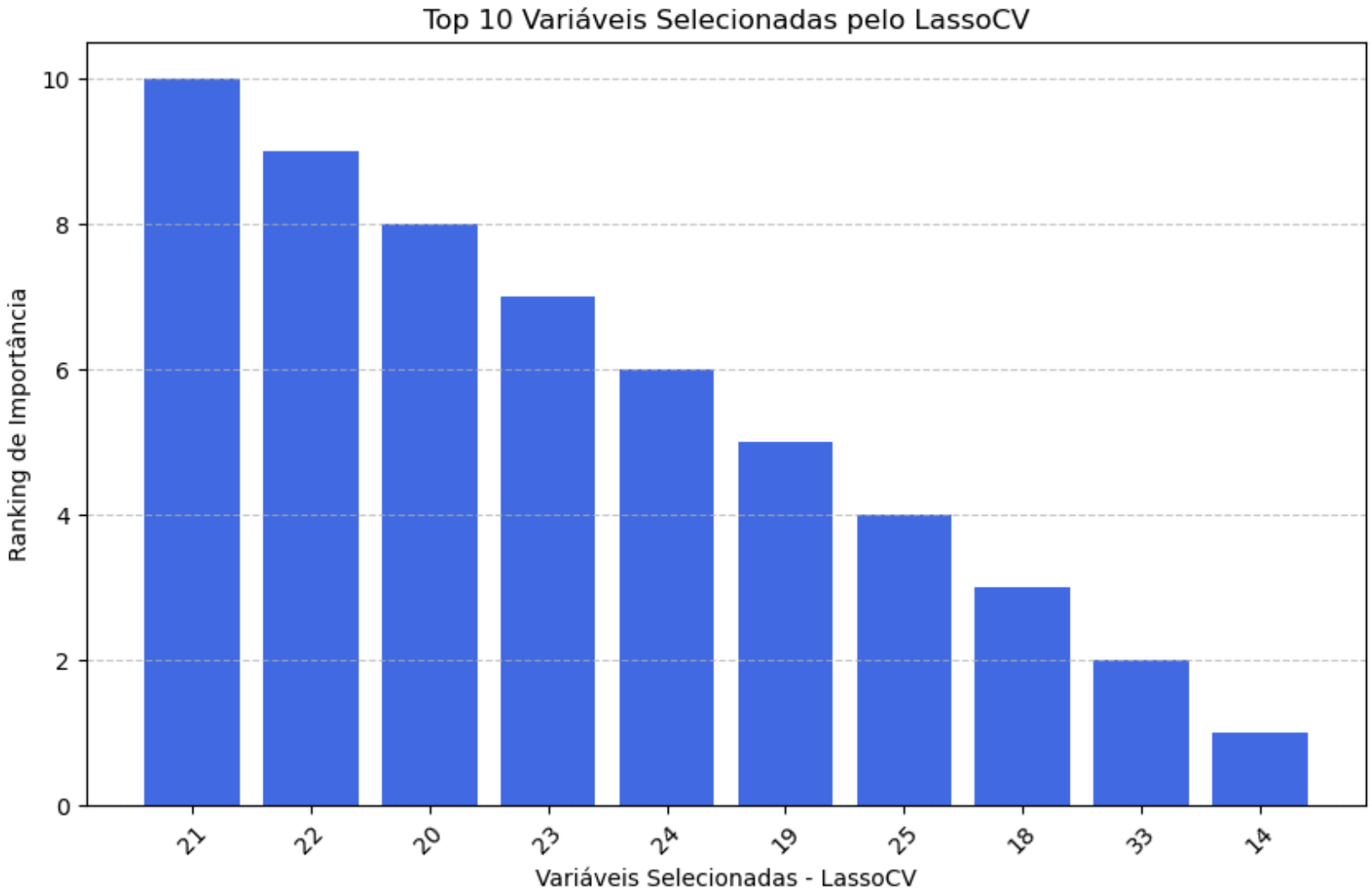
Implementação do Sklearn

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LassoCV, Ridge
from sklearn.feature_selection import SequentialFeatureSelector

# Carregar os dados
VSdata = np.loadtxt('VSdata.csv', delimiter=',')

# Separar variável alvo (y) e variáveis preditoras (X)
y = VSdata[:, 0]
X = VSdata[:, 1:]

# Normalizar os dados
xscaler = StandardScaler()
X_scaled = xscaler.fit_transform(X)

yscaler = StandardScaler()
print("Antes da normalização:", y[:5])  # Exibe os primeiros valores
y_scaled = yscaler.fit_transform(y[:, None]).ravel()  # ravel() transforma de (n,1) para (n,)
print("Depois da normalização:", y_scaled[:5])  # Exibe os valores transformados

# ------------------------------------------------
# Parte 1: Seleção de Variáveis com LassoCV

# Ajusta o modelo LassoCV com validação cruzada (5 folds)
lasso_model = LassoCV(cv=5).fit(X_scaled, y_scaled)

# Obter os coeficientes absolutos das variáveis e identificar as mais relevantes
top_k_inputs_lasso = np.argsort(np.abs(lasso_model.coef_))[-10:][::-1]  # Top 10 coeficientes mais importantes
selected_features_lasso = top_k_inputs_lasso + 1  # Ajuste para refletir os números originais das colunas

# Exibir as variáveis selecionadas pelo LassoCV
print("Variáveis Selecionadas pelo LassoCV:", selected_features_lasso)
```

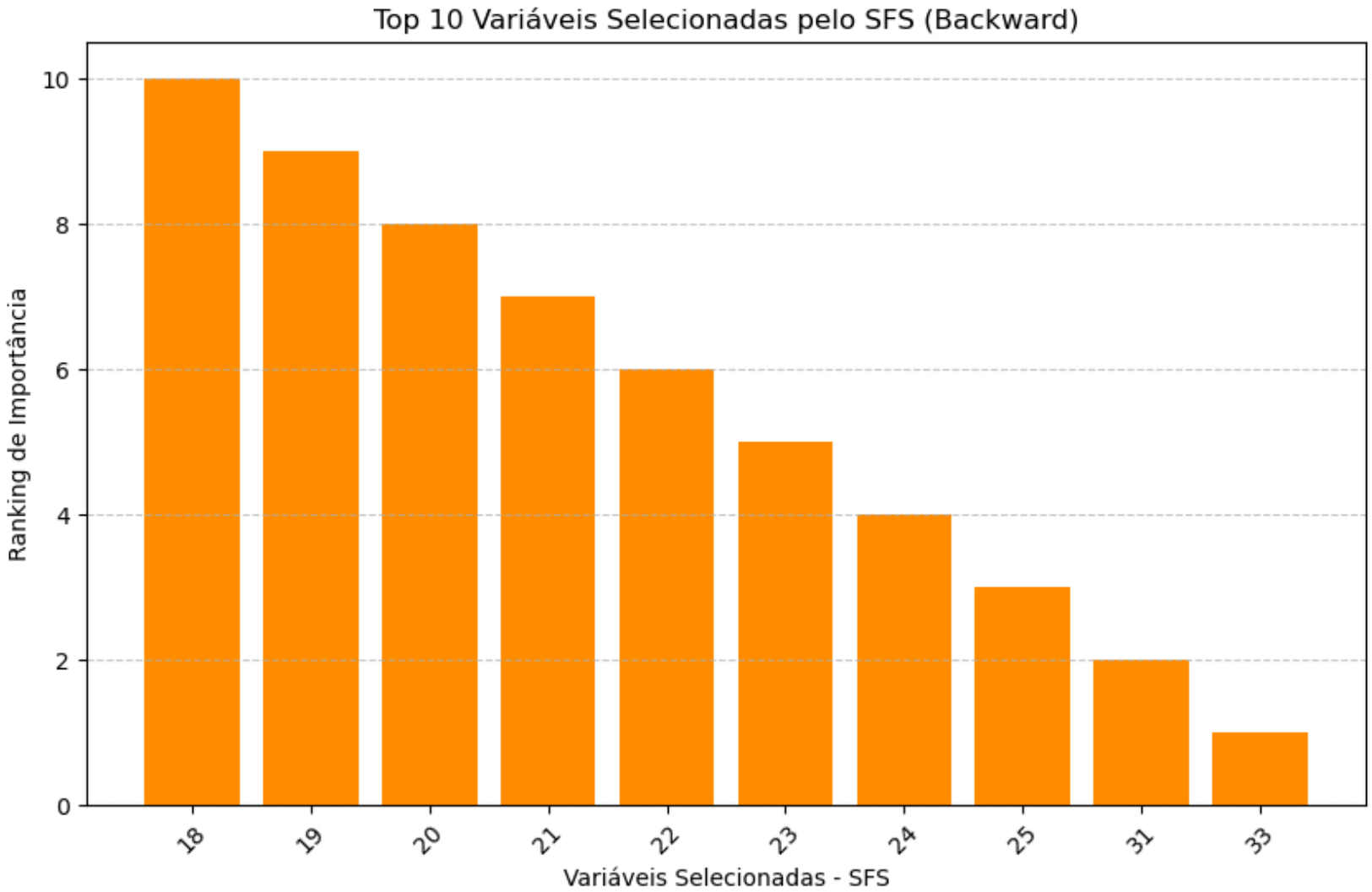
Implementação do Sklearn

```python
# Parte 2: Seleção de Variáveis com Sequential Feature Selector (SFS) - Backward

# Número de variáveis a selecionar
k = 10

# Usando Ridge Regression como modelo para a seleção de variáveis
model_ridge = Ridge(alpha=1.0)

# Realizar a seleção de variáveis com SFS (Backward)
BSFS = SequentialFeatureSelector(model_ridge, n_features_to_select=k, direction='backward', cv=5)
BSFS.fit(X_scaled, y_scaled)

# Obter índices das variáveis selecionadas pelo SFS
selected_indices_sfs = BSFS.get_support(indices=True)
selected_features_sfs = selected_indices_sfs + 1  # Ajuste para refletir os números originais das colunas

# Exibir as variáveis selecionadas pelo SFS
print("Variáveis Selecionadas pelo SFS (Backward):", selected_features_sfs)

# ------------------------------------------------
# Visualização das variáveis selecionadas pelo LassoCV
plt.figure(figsize=(10, 6))
plt.bar(range(k), np.arange(k)[::-1] + 1, tick_label=selected_features_lasso, color='royalblue')
plt.xlabel("Variáveis Selecionadas - LassoCV")
plt.ylabel("Ranking de Importância")
plt.title("Top 10 Variáveis Selecionadas pelo LassoCV")
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

# Visualização das variáveis selecionadas pelo SFS
plt.figure(figsize=(10, 6))
plt.bar(range(k), np.arange(k)[::-1] + 1, tick_label=selected_features_sfs, color='darkorange')
plt.xlabel("Variáveis Selecionadas - SFS")
plt.ylabel("Ranking de Importância")
plt.title("Top 10 Variáveis Selecionadas pelo SFS (Backward)")
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

Seleção de Variáveis / Seleção de Características



Seleção de Variáveis / Seleção de Características



Sumário

Métodos univariados

- Métodos univariados analisam e limpam cada variável individualmente, sem considerar interações entre elas. A regra 3-sigma e o identificador de Hampel são técnicas comuns nesse grupo, usadas para identificar e tratar outliers, baseando-se em medidas estatísticas como média e desvio padrão, ou medindo a diferença de cada valor em relação à mediana.
- **Regra 3-sigma:** A regra 3-sigma estabelece que qualquer valor além do intervalo ± 3 (média $\pm$ três vezes o desvio padrão) é considerado um outlier. Isso se baseia na distribuição normal, onde 99,87% dos dados estão dentro desse intervalo.

Tratamento de Outlier

Métodos univariados

- **Identificador Hampel:** É uma alternativa robusta à regra 3-sigma para detecção de outliers. Enquanto a regra 3σ usa a **média(μ)** e o **desvio padrão(σ)**, que são sensíveis a valores extremos, o Identificador Hampel substitui esses parâmetros pela **mediana** e pelo **Desvio Absoluto Mediano (MAD)**, que são mais resistentes a outliers;
- Uma observação é marcada como outlier se estiver fora do intervalo:

$$mediana(x) \pm 3\sigma_{MAD} \tag{5}$$

- Em que:

$$\sigma_{MAD} = 1,4826\ x\ mediana(|x - mediana(x)|) \tag{6}$$

- Esse método é mais confiável quando os dados contêm valores anômalos significativos, pois a mediana e o MAD não são facilmente influenciados por outliers.

Métodos univariados

- 3σ X Identificador Hampel

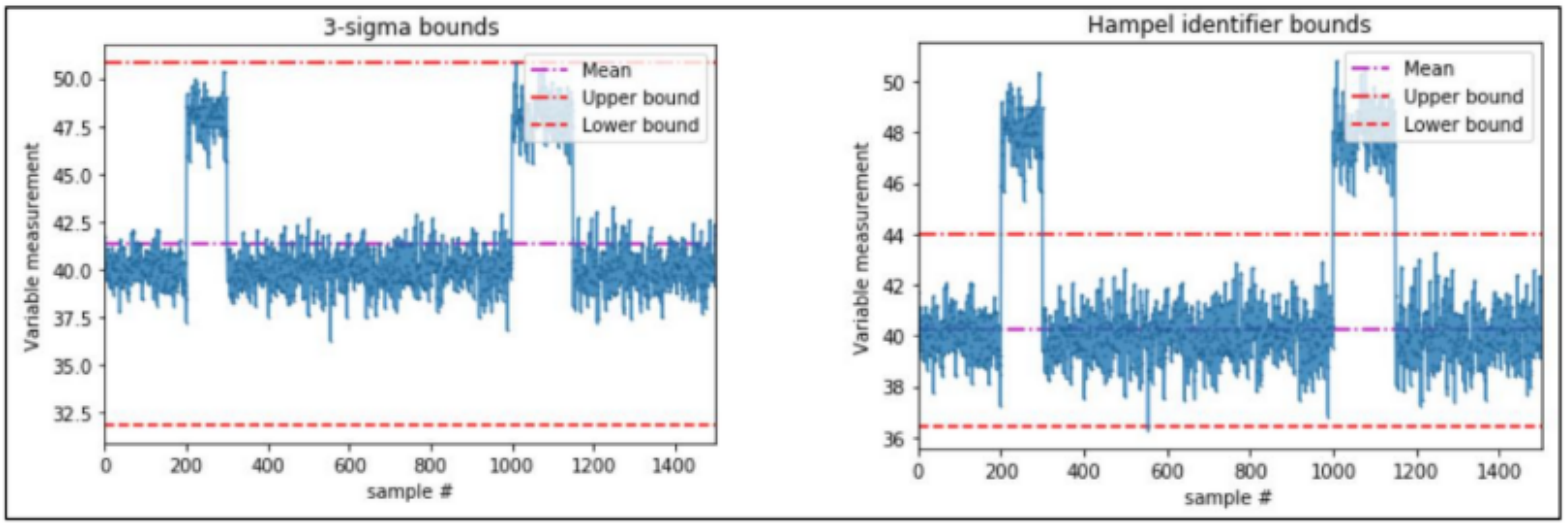


Figure 4.9: Univariate bounds obtained with 3-sigma and Hampel identifier rules for data contaminated with outliers

Sumário

Métodos multivariados

- Métodos multivariados de detecção de outliers analisam a estrutura conjunta dos dados, considerando sua distribuição no espaço multidimensional;
- Utilizam métricas de distância, como a distância de Mahalanobis ou técnicas baseadas em aprendizado de máquina, para identificar pontos anômalos;
- No exemplo da Figura 4.10, um método multivariado reconheceria o ponto vermelho como um outlier porque ele não segue o padrão esperado da distribuição conjunta das variáveis.

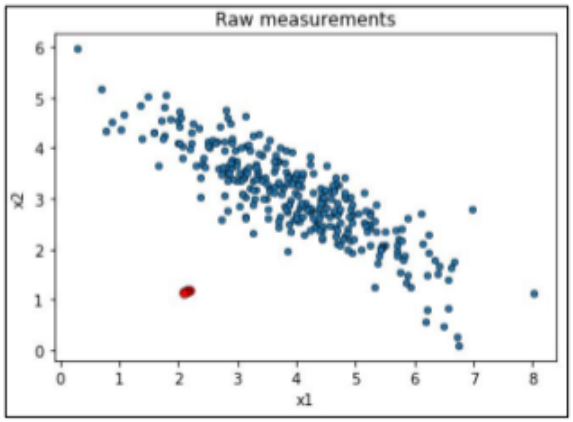


Figure 4.10: Outliers in 2D space difficult to detect via univariate methods

Métodos multivariados

- **A Distância de Mahalanobis (MD):** Mede a distância entre um ponto e o centro da distribuição dos dados, levando em conta a forma da distribuição e as correlações entre variáveis, convertendo um problema de detecção de outliers multivariados em um problema univariado. A MD de qualquer observação x_n é dada por:

$$MD(x_n) = \sqrt{(x_n - \overline{x} \sum^{-1} (x_n - \overline{x})^T} \tag{7}$$

- Em que: $\overline{x}$ e $\sum$ são médias e covariâncias das observações amostradas.

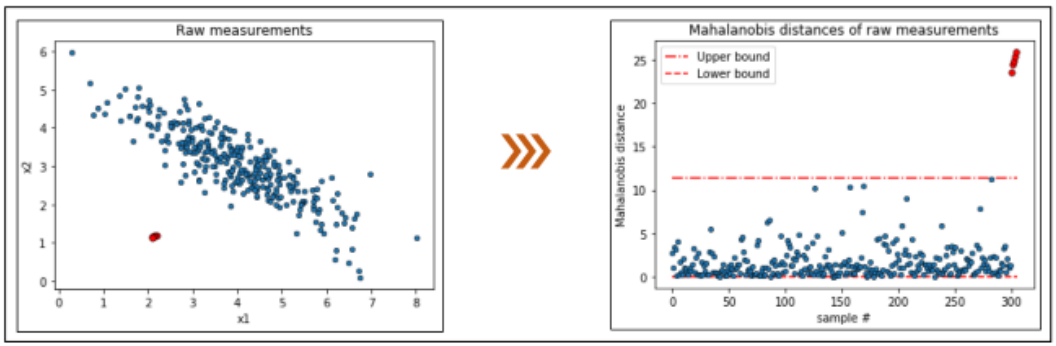
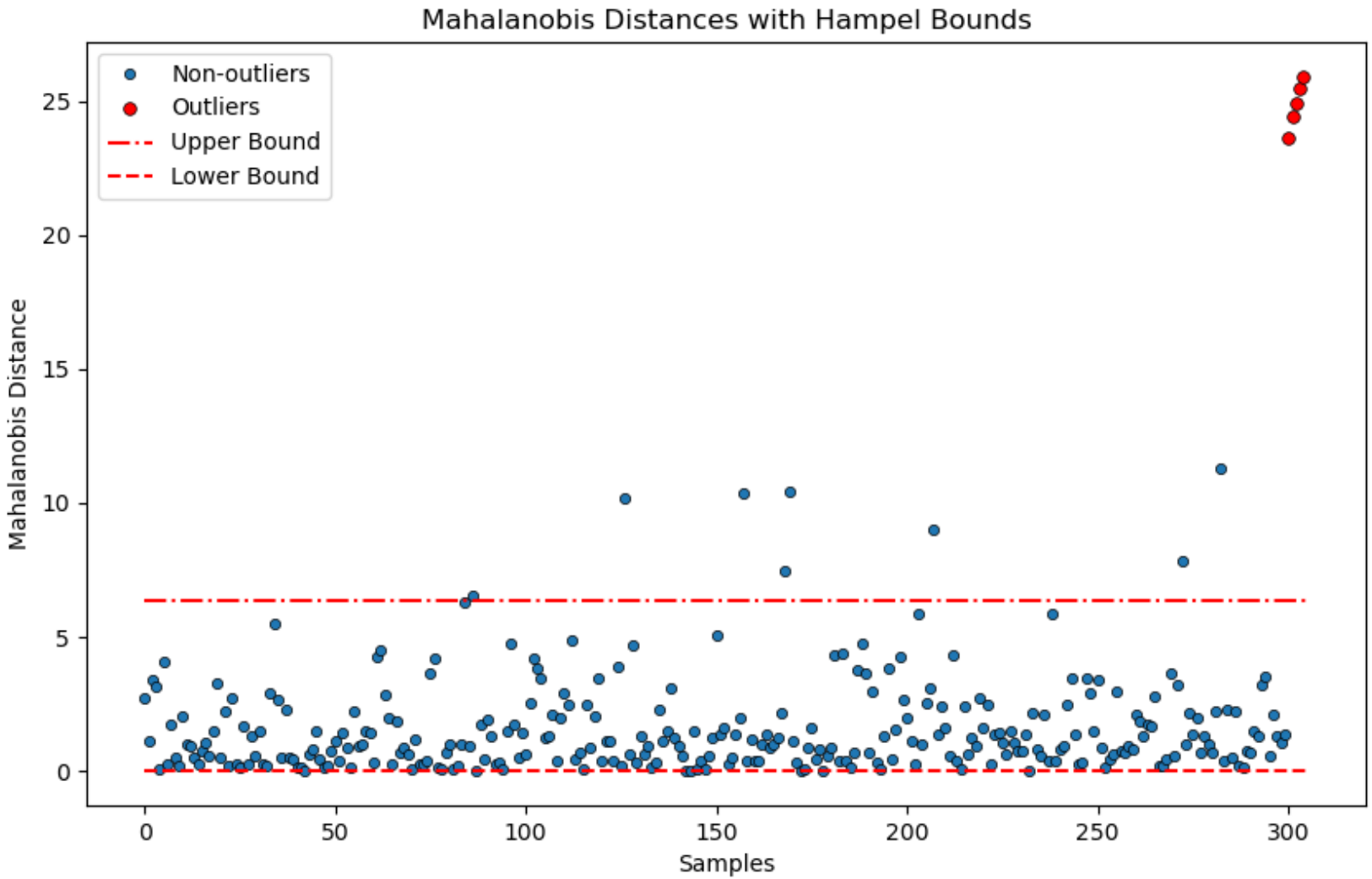


Figure 4.11: Multivariate outlier detection via Mahalanobis distance and Hampel identifier

Tratamento de Outlier

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.covariance import EmpiricalCovariance
from scipy import stats

# 1. Carregar os dados
data_2Doutlier = np.loadtxt('simple2D_outlier.csv', delimiter=',')

# 2. Calcular as distâncias de Mahalanobis e transformá-las para distribuição Gaussiana usando raiz cúbica
emp_cov = EmpiricalCovariance().fit(data_2Doutlier)
mahalanobis_distances = emp_cov.mahalanobis(data_2Doutlier)
mahalanobis_cube_root = np.power(mahalanobis_distances, 1/3)  # Raiz cúbica

# 3. Calcular os limites de Hampel (upper e lower bounds)
median = np.median(mahalanobis_cube_root)
mad = stats.median_abs_deviation(mahalanobis_cube_root)
upper_bound = np.power(median + 3 * mad, 3)
lower_bound = np.power(median - 3 * mad, 3)

# 4. Plotar as distâncias de Mahalanobis com os limites (outliers destacados em vermelho)
plt.figure(figsize=(10, 6))

# Plot dos dados não-outliers (excluindo os últimos 5)
plt.plot(mahalanobis_distances[:-5], '.', markeredgecolor='k', markeredgewidth=0.5, ms=9, label='Non-outliers')

# Plot dos últimos 5 outliers em vermelho
plt.plot(np.arange(len(mahalanobis_distances)-5, len(mahalanobis_distances)), mahalanobis_distances[-5:], '.r',
         markeredgecolor='k', markeredgewidth=0.5, ms=11, label='Outliers')

# Adicionar os limites superiores e inferiores
plt.hlines(upper_bound, 0, len(mahalanobis_distances), colors='r', linestyles='dashdot', label='Upper Bound')
plt.hlines(lower_bound, 0, len(mahalanobis_distances), colors='r', linestyles='dashed', label='Lower Bound')

# Adicionar título e rótulos
plt.title('Mahalanobis Distances with Hampel Bounds')
plt.xlabel('Samples')
plt.ylabel('Mahalanobis Distance')

# Exibir legenda
plt.legend()

# Exibir o gráfico
plt.show()
```

Tratamento de Outlier



Métodos multivariados

- **Estimador de Determinante de Covariância Mínima (MCD):** É um método robusto para estimar a matriz de covariância de um conjunto de dados, minimizando o impacto de outliers. Ele busca um subconjunto de tamanho h (menor que o total de amostras n) que apresenta a menor determinante da matriz de covariância;
- Esse subconjunto representa a parte mais central dos dados, garantindo uma estimativa mais confiável para distribuições contaminadas.

Métodos multivariados

- Estimador de Determinante de Covariância Mínima (MCD)

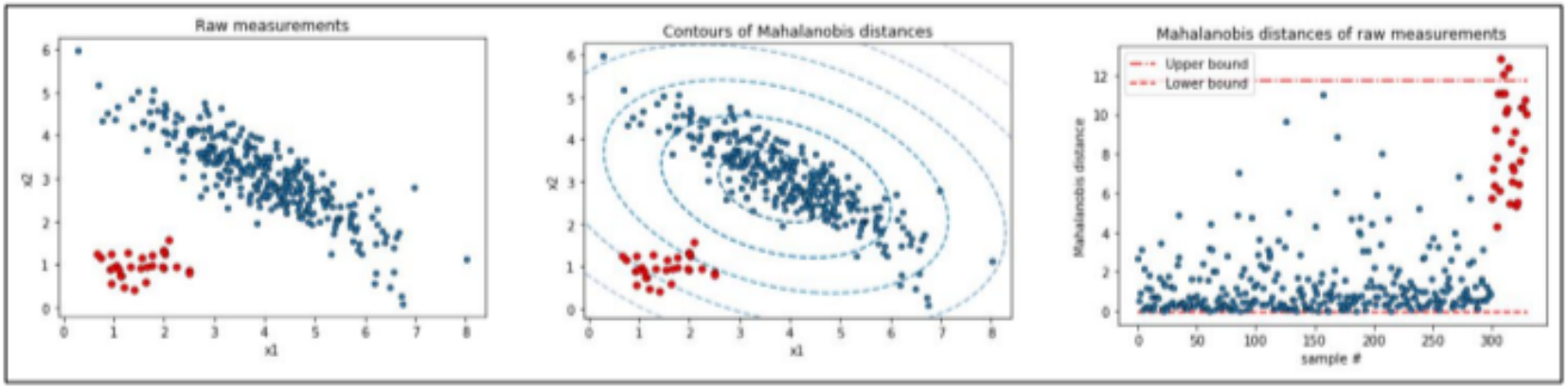


Figure 4.12: : Outliers in 2D space difficult to detect via classical Mahalanobis distance method

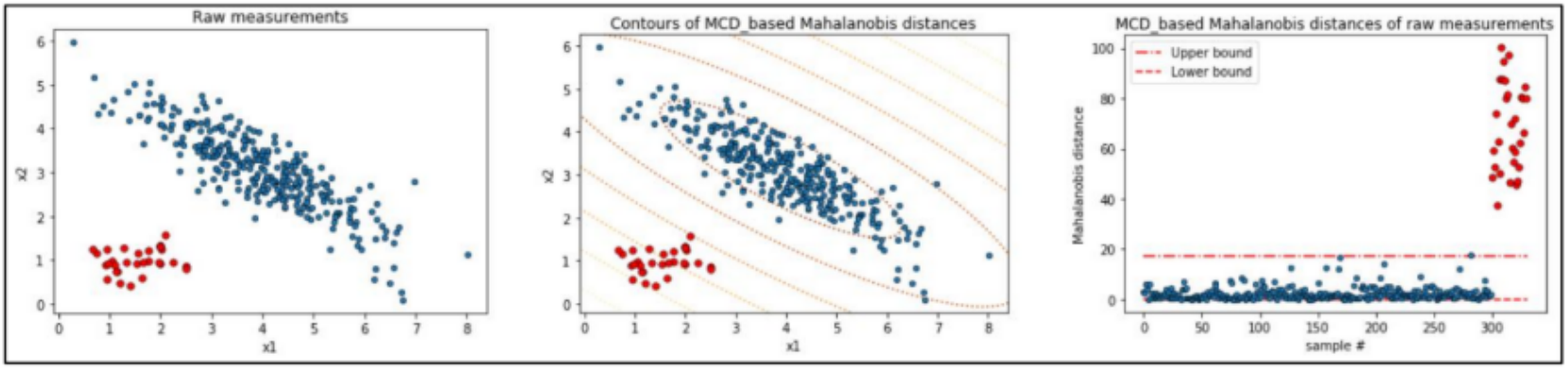


Figure 4.13: Multivariate outlier detection via MCD-based robust Mahalanobis distances

Métodos multivariados

- **Estimador de Determinante de Covariância Mínima (MCD)**: O código abaixo mostra o cálculo de MDs baseados em MCD.

```python
import numpy as np
from sklearn.covariance import MinCovDet

def load_data(file_path):
    """
    Função para carregar dados de um arquivo CSV.
    :param file_path: Caminho para o arquivo CSV.
    :return: Dados carregados como um array NumPy.
    """
    try:
        data = np.loadtxt(file_path, delimiter=',')
        print(f"Dados carregados com sucesso do arquivo {file_path}")
        return data
    except Exception as e:
        print(f"Erro ao carregar os dados: {e}")
        return None

def compute_mahalanobis_distances(data):
    """
    Função para calcular as distâncias de Mahalanobis usando a estimativa MCD (Minimum Covariance Determinant).
    :param data: Dados de entrada.
    :return: Distâncias de Mahalanobis baseadas no MCD.
    """
    MCD_cov = MinCovDet().fit(data)
    MD_MCD = MCD_cov.mahalanobis(data)
    return MD_MCD

# Carregar dados do arquivo
data_2Doutlier = load_data('complex2D_outlier.csv')

# Verificar se os dados foram carregados corretamente antes de continuar
if data_2Doutlier is not None:
    # Calcular as distâncias de Mahalanobis usando MCD
    MD_MCD = compute_mahalanobis_distances(data_2Doutlier)
    print("Distâncias de Mahalanobis calculadas com sucesso.")
else:
    print("Falha ao carregar os dados. O código não prosseguiu.")
```

Métodos multivariados

- **Detecção de Outliers Baseada em Análise de Componentes Principais (PCA)**
- Nem todos os outliers devem ser removidos. Na Figura 4.14, as amostras A desviam da correlação principal, enquanto as amostras B seguem a correlação, mas estão distantes da maioria dos dados.
- A remoção das amostras B depende do contexto. A abordagem baseada em MD não distingue os tipos de outliers, tornando a PCA uma alternativa para reter as amostras B.

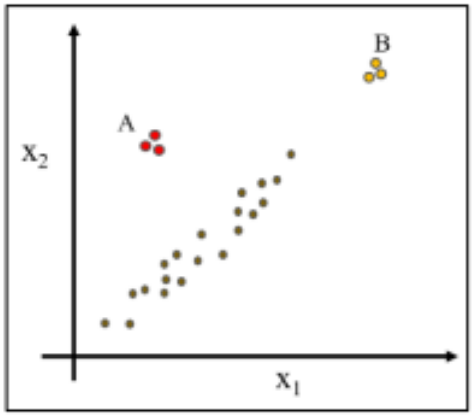


Figure 4.14: Different types of outliers which can be distinguished via PCA

Métodos de Mineração de Dados

- Os métodos anteriores assumem uma distribuição Gaussiana ou unimodal, o que nem sempre ocorre. A Figura 4.15 mostra três clusters distintos com outliers, onde MD e PCA falham por não considerarem múltiplos agrupamentos.

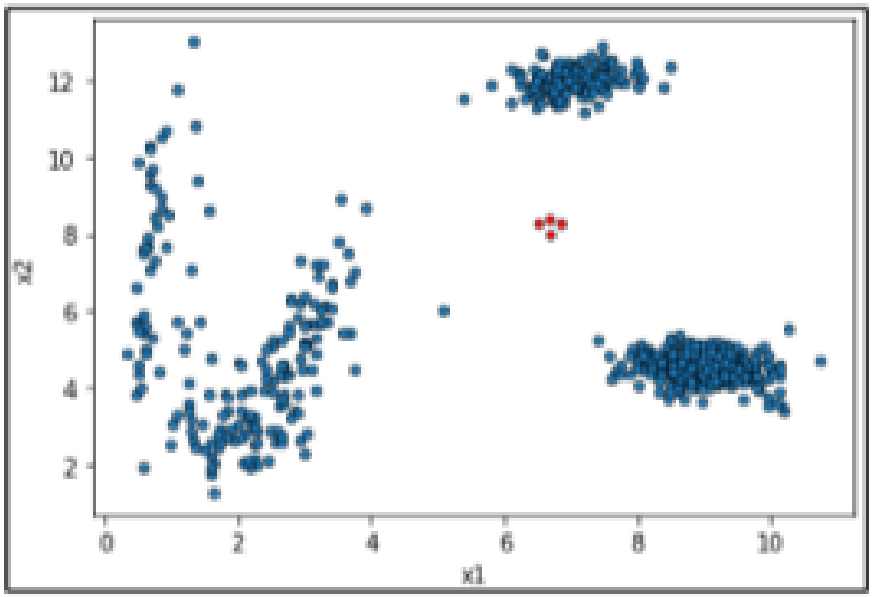


Figure 4.15: Complex multivariate data distribution with outliers

Tratamento de Outlier

Métodos de Mineração de Dados

- **Baseados em densidade (KDE)** – Estimam a densidade da distribuição dos dados, classificando como outliers as observações em regiões de baixa densidade;
- **Baseados em cluster (GMM, K-means)** – Identificam agrupamentos distintos, sinalizando como outliers as amostras pertencentes a clusters pequenos ou irrelevantes;
- **Baseados em vizinhos (KNN, LOF)** – Determinam a proximidade entre pontos de dados, identificando como outliers aqueles com grandes distâncias de seus vizinhos ou densidade local baixa.

Técnicas Robustas Resistentes a Outliers

- Os métodos robustos são uma abordagem alternativa para lidar com outliers ao ajustar modelos diretamente em dados contaminados.
- Esses algoritmos são projetados para minimizar a influência dos outliers, garantindo que apenas os inliers contribuam significativamente para a determinação dos parâmetros do modelo.
- Dessa forma, o impacto dos outliers é reduzido, permitindo um ajuste mais preciso e representativo da estrutura real dos dados.
- Métodos utilizados: Algoritmo RANSAC, IRPLS (PLS iterativamente reponderado).

Técnicas Robustas Resistentes a Outliers

- O método robusto é capaz de ignorar com sucesso os outliers para encontrar o ajuste correto entre o alvo e o preditor.

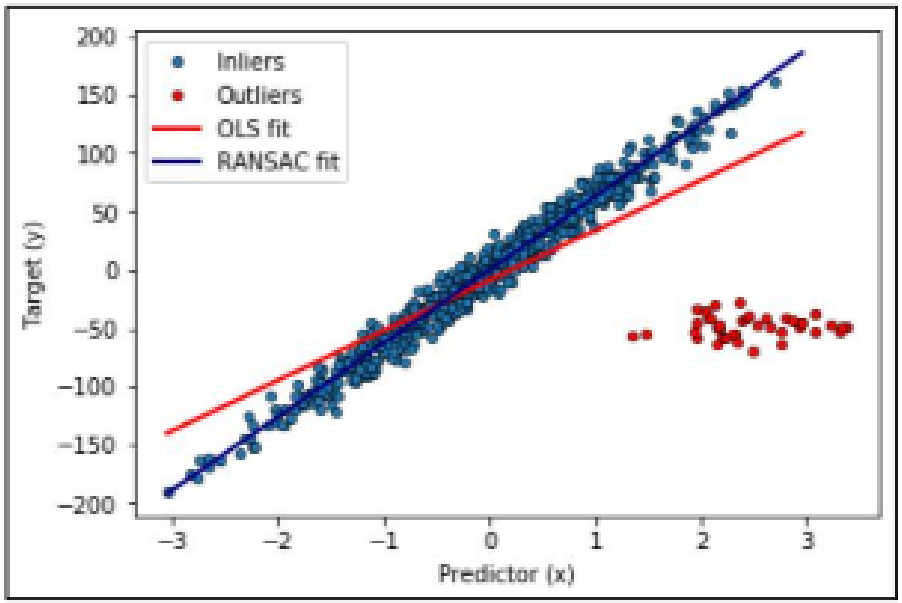


Figure 4.16: Robust model fitting with outlier-infested data

Tratamento de Dados Ausentes

- Problemas de dados ausentes ocorrem quando faltam valores em uma ou mais variáveis dentro de um conjunto de dados. Se a quantidade de amostras com dados ausentes for pequena em relação ao total de amostras, essas podem ser descartadas sem impactar significativamente o modelo.
- Contudo, se a quantidade de dados ausentes for significativa ou o descarte prejudicar a análise, pode-se utilizar técnicas de imputação. Essas técnicas buscam preencher os valores ausentes com base em informações disponíveis, seja da mesma variável ou de outras variáveis relacionadas, mantendo a integridade do modelo e evitando viés nas análises.

Tratamento de Dados Ausentes

```python
import numpy as np
from sklearn.impute import SimpleImputer

def imputar_dados_medianos(dados):
    """
    Função para realizar imputação de dados ausentes usando a média.
    :param dados: Dados de entrada com valores ausentes (np.nan).
    :return: Dados com valores ausentes substituídos pela média.
    """
    imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
    dados_imputados = imputer.fit_transform(dados)
    return dados_imputados

# Dados de exemplo com valores ausentes (np.nan)
dados_amostra = [[1, 2, np.nan], [3, 4, 3], [np.nan, 6, 5], [8, 8, 7]]

# Imputar os dados usando a média
dados_imputados = imputar_dados_medianos(dados_amostra)

# Exibir os dados antes e depois da imputação
print("Dados originais:")
print(np.array(dados_amostra))
print("\nDados após imputação de média:")
print(dados_imputados)
```

Tratamento de Dados Ausentes

```python
import numpy as np
from sklearn.impute import KNNImputer

def imputar_dados_knn(dados, n_neighbors=2):
    """
    Função para realizar imputação de dados ausentes usando o algoritmo KNN (K-Nearest Neighbors).
    :param dados: Dados de entrada com valores ausentes (np.nan).
    :param n_neighbors: Número de vizinhos a serem considerados para a imputação.
    :return: Dados com valores ausentes imputados.
    """
    imputer = KNNImputer(n_neighbors=n_neighbors)
    dados_imputados = imputer.fit_transform(dados)
    return dados_imputados

# Dados de exemplo com valores ausentes (np.nan)
dados_amostra = [[1, 2, np.nan], [3, 4, 3], [np.nan, 6, 5], [8, 8, 7]]

# Imputar os dados usando KNN
dados_imputados_knn = imputar_dados_knn(dados_amostra, n_neighbors=2)

# Exibir os dados antes e depois da imputação
print("Dados originais:")
print(np.array(dados_amostra))
print("\nDados após imputação KNN:")
print(dados_imputados_knn)
```

Aprendizado de Máquina para Engenharia de Sistemas de Processos: Pré-processamento de Dados - limpeza de dados do processo

Elizabeth Alves de Oliveira
Prof. Dr. Luiz Gonzaga Sales Vasconcelos

Universidade Federal de Campina Grande - UFCG
Centro de Ciências e Tecnologia - CCT
Programa de Pós-Graduação em Engenharia Química - PPGEQ
Redes Neurais

Abril de 2025