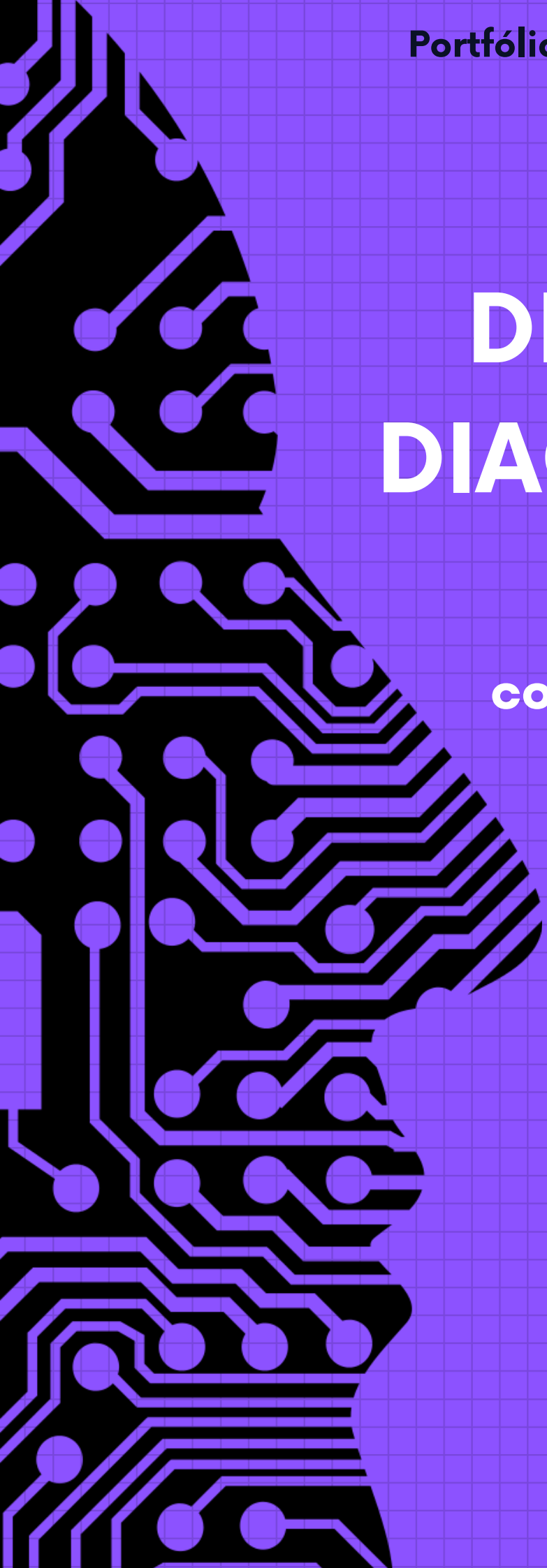
Portfólio Pessoal



DETECÇÃO E DIAGNÓSTICO DE FALHAS

com Machine Learning

Mario Henrique Cosme Juvencio

DETECÇÃO E DIAGNÓSTICO DE FALHAS COM MACHINE LEARNING

Modelos preditivos / Máquina de Vetores de Suporte / Python

1 APRESENTAÇÃO DO PROBLEMA

Este projeto se concentra na implementação de um modelo de detecção e diagnóstico de falhas em sensores físicos para um reator CSTR (Continuous flow Stirred Tank Reactors), operando com uma reação complexa e não-linear, utilizando a técnica de Máquinas de Vetores de Suporte (SVM). O objetivo é desenvolver um algoritmo capaz de identificar rapidamente os desvios e realizar o diagnóstico por meio de informações sobre as demais variáveis do processo.

A técnica de SVM foi escolhida por sua capacidade de lidar com problemas de classificação, sendo uma técnica poderosa para a detecção e diagnóstico de falhas. O modelo proposto neste projeto tem o potencial de contribuir para a segurança e eficiência do processo químico, permitindo a detecção rápida de falhas e fornecendo diagnósticos precisos para auxiliar na tomada de decisões.

A base de dados empregada foi obtida a partir de simulações computacionais do processo químico realizadas a partir de um algoritmo construído em Python.

2 DESCRIÇÃO DA BASE DE DADOS

Na Tabela 1 são apresentados os rótulos e as quantidades de dados em cada classe na base de dados. Cada rótulo representa a falha em um sensor físico específico do reator.

Tabela 1 – Classes e quantidade de dados.

Classe	Quantidade de Dados
Normal	1.196.663
Falha 1	99.834
Falha 2	99.479
Falha 3	99.935
Falha 4	99.480
Falha 5	99.494
Falha 6	99.674

As falhas são desvios do comportamento normal da variável, sendo obtidos a partir da multiplicação do valor real da variável por um número randômico entre -0,9 e 0,9 e adição do resultado ao valor da variável.

O código em Python desenvolvido para a inserção de falhas no processo é apresentado a seguir.

```python
import pandas as pd
import numpy as np

# Ler o arquivo e armazenar em um dataframe chamado "df"
df = pd.read_csv("simulacoes_9s.csv")

# Criar um novo dataframe chamado "Simulacoes_9s_falhas"
Simulacoes_9s_falhas = pd.DataFrame(columns=['CA (kmol/m3)', 'Tr (K)', 'Tc (K)', 'Tr0 (K)',
                                             'CA0 (kmol/m3)', 'Qc (kJ/min)', 'qr (m3/min)',
                                             'Classes_Falhas'])

# Função para realizar a perturbação
def perturbar(valor):
    return valor * (1 + np.random.uniform(-0.9, 0.9))

for index, row in df.iterrows():
    row = pd.Series(row)
    # Gerar um número aleatório (0 ou 1) para decidir se a linha sofrerá perturbação
    perturbacao = np.random.randint(0, 2)

    # Se não houver perturbação
    if perturbacao == 0:
        row['Classes_Falhas'] = 'normal'
        Simulacoes_9s_falhas = pd.concat([Simulacoes_9s_falhas, row.to_frame().T], ignore_index=True)
    else:
        # Escolher uma coluna aleatoriamente para perturbar
        coluna_perturbada = np.random.choice(['CA0 (kmol/m3)', 'Tr0 (K)', 'Qc (kJ/min)',
                                              'qr (m3/min)', 'Tr (K)', 'Tc (K)'])
        row[coluna_perturbada] = perturbar(row[coluna_perturbada])


        # Atribuir a classe de falha correspondente
        if coluna_perturbada == 'CA0 (kmol/m3)':
            row['Classes_Falhas'] = 'Falha_1'
        elif coluna_perturbada == 'Tr0 (K)':
            row['Classes_Falhas'] = 'Falha_2'
        elif coluna_perturbada == 'Qc (kJ/min)':
            row['Classes_Falhas'] = 'Falha_3'
        elif coluna_perturbada == 'qr (m3/min)':
            row['Classes_Falhas'] = 'Falha_4'
        elif coluna_perturbada == 'Tr (K)':
            row['Classes_Falhas'] = 'Falha_5'
        elif coluna_perturbada == 'Tc (K)':
            row['Classes_Falhas'] = 'Falha_6'

        Simulacoes_9s_falhas = pd.concat([Simulacoes_9s_falhas, row.to_frame().T], ignore_index=True)

Simulacoes_9s_falhas.to_csv("Simulacoes_9s_falhas.csv", index=False)
```

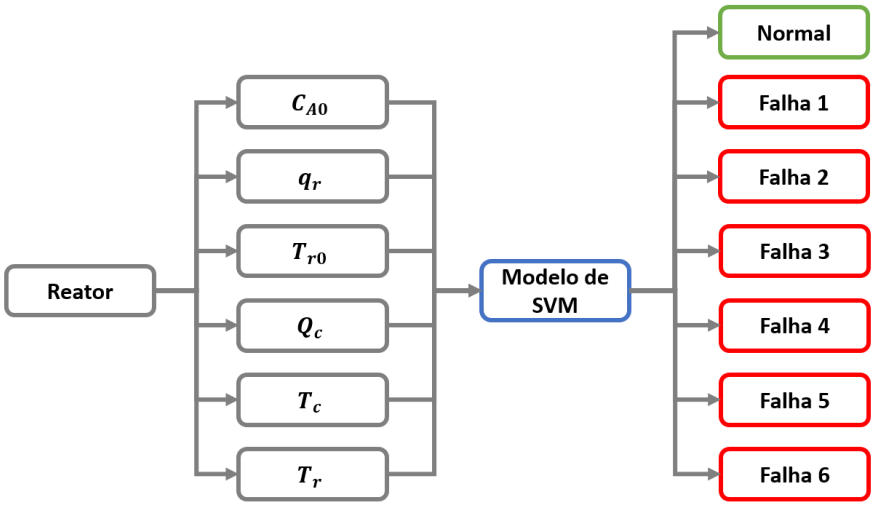
3 DESENVOLVIMENTO DO MODELO

No contexto deste projeto, a SVM foi utilizada para detecção e diagnóstico de falhas na operação do reator CSTR operando com a reação de Van de Vusse. O classificador SVM, implementado pela classe SVC da biblioteca Scikit-Learn, foi treinado utilizando um conjunto de treinamento. Para avaliar o desempenho do modelo, foram realizadas previsões com o modelo treinado, empregando um conjunto de teste.

Diversas métricas de desempenho foram calculadas, tais como acurácia, recall, F1 score e taxa de erro. Além disso, foi gerada uma matriz de confusão para comparar os resultados verdadeiros com os previstos pelo modelo.

A partir dos resultados da matriz de confusão, o desempenho do modelo foi avaliado tanto na detecção quanto no diagnóstico de falhas, de maneira individual. Essa abordagem permitiu verificar a capacidade do modelo em detectar as falhas e, após a detecção, sua eficácia em identificar corretamente o tipo específico de falha. O fluxograma da Figura 1 ilustra o esquema de funcionamento do modelo, apresentando os atributos previsores e as classes.

Figura 1 – Fluxograma de funcionamento do modelo.



No fluxograma temos a representação dos atributos previsores e os rótulos das classes. As etapas de desenvolvimento do modelo são descritas nos próximos tópicos.

3.1 Importação das Bibliotecas Python

Foram importadas as seguintes bibliotecas:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
```

A biblioteca “pandas foi utilizada para facilitar a manipulação e análise de dados, fornecendo a estrutura de dados para o desenvolvimento do projeto. As bibliotecas “matplotlib” e “seaborn” foram utilizadas para a criação de visualizações gráficas. A função “train_test_split” da biblioteca Scikit-Learn foi empregada para dividir a matriz de dados em dois subconjuntos: um para treinamento e outro para teste do modelo. Além disso, a partir do módulo “metrics” da Scikit-Learn, foram importadas as funções responsáveis pelo cálculo das métricas de avaliação do modelo, tais como acurácia, precisão, recall, F1 score e matriz de confusão. Em seguida, a classe “StandardScaler” do módulo “preprocessing” da Scikit-Learn foi importada com o objetivo de implementar a transformação de escala nas variáveis de entrada do modelo de aprendizado de máquina. Por fim, a classe “SVC” foi importada para a implementação do algoritmo de SVM.

3.2 Preparação dos Dados

Primeiramente, os dados foram carregados a partir do arquivo “Simulacoes_9s_falhas.csv”. As colunas contendo os atributos previsores e a variável contendo os rótulos foram extraídas para serem utilizadas no treinamento e teste do modelo.

```
# Lendo o arquivo e armazenando as colunas desejadas em um dataframe
file_path = "Simulacoes_9s_falhas.csv"
columns = ['Tr (K)', 'Tc (K)', 'Tr0 (K)', 'CA0 (kmol/m3)', 'Qc (kJ/min)', 'qr (m3/min)', 'Classes_Falhas']
df = pd.read_csv(file_path, usecols=columns)
```

Após a extração e armazenamento em um dataframe, os dados foram divididos em dois subconjuntos. O primeiro subconjunto continha os atributos previsores e o segundo as classes com os rótulos conhecidos. Em seguida, os dados foram separados em conjuntos de treinamento e teste, com 70% dos dados destinados para treinamento e 30% para teste. É importante ressaltar que a divisão foi realizada de forma aleatória a partir da utilização da função “train_test_split” e definição do parâmetro “test_size”.

```
# Dividindo os dados em atributos previsores e classes
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

# Dividindo os dados em conjuntos de treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Como os atributos previsores apresentam escalas diferentes entre si, foi necessário realizar a padronização usando a função “StandardScaler”. Essa etapa é crucial para garantir que o desempenho do modelo não seja negativamente influenciado pela diferença de escala entre as variáveis.

```
# Padronizando os dados
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

3.3 Treinamento e Teste

Após a preparação dos dados, o modelo de SVM foi treinado utilizando os conjuntos de treinamento “X_train” e “y_train”, que representam os atributos previsores e as classes, respectivamente. Em seguida, o modelo treinado foi aplicado ao conjunto de teste para fazer previsões, permitindo avaliar a capacidade de generalização do modelo. Em outras palavras, foi avaliado o desempenho do modelo em relação a dados que não foram “vistos” durante a etapa de treinamento.

```
# Treinando o classificador LinearSVC
classifier = SVC(random_state=42)
classifier.fit(X_train, y_train)

# Realizando as previsões
y_pred = classifier.predict(X_test)
```

3.4 Cálculo das Métricas de Desempenho

Com as previsões realizadas no conjunto de teste, foram calculadas as métricas de desempenho para o classificador. Para obter uma visão geral da proporção de previsões corretas do modelo, a acurácia foi calculada, comparando os rótulos previstos com os rótulos reais. A matriz de confusão foi construída com o objetivo de permitir uma visualização mais detalhada do desempenho do modelo em diferentes classes, comparando a quantidade de previsões corretas e incorretas para cada uma. Além disso, a partir da matriz de confusão é possível avaliar

o modelo na tarefa de detecção e na tarefa de diagnóstico, de forma individual. Por fim, foram calculadas as métricas de precisão, recall e f1-score para cada classe.

```
# Calculando as métricas de desempenho
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Exibindo a matriz de confusão como uma figura
class_labels = classifier.classes_
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, cmap="YlGnBu", fmt='g', cbar=False, xticklabels=class_labels,
            yticklabels=class_labels)
plt.xlabel('Classe prevista')
plt.ylabel('Classe verdadeira')
plt.title('Matriz de Confusão')
plt.show()

print("Acurácia: {:.2f}".format(accuracy))

# Imprimindo o relatório de classificação completo
print("\nRelatório de classificação:")
print(classification_report(y_test, y_pred))
```

4 AVALIAÇÃO DO MODELO

Os valores das métricas de desempenho para cada classe são fornecidas na Tabela 2.

Tabela 2 – Métricas de desempenho por classe.

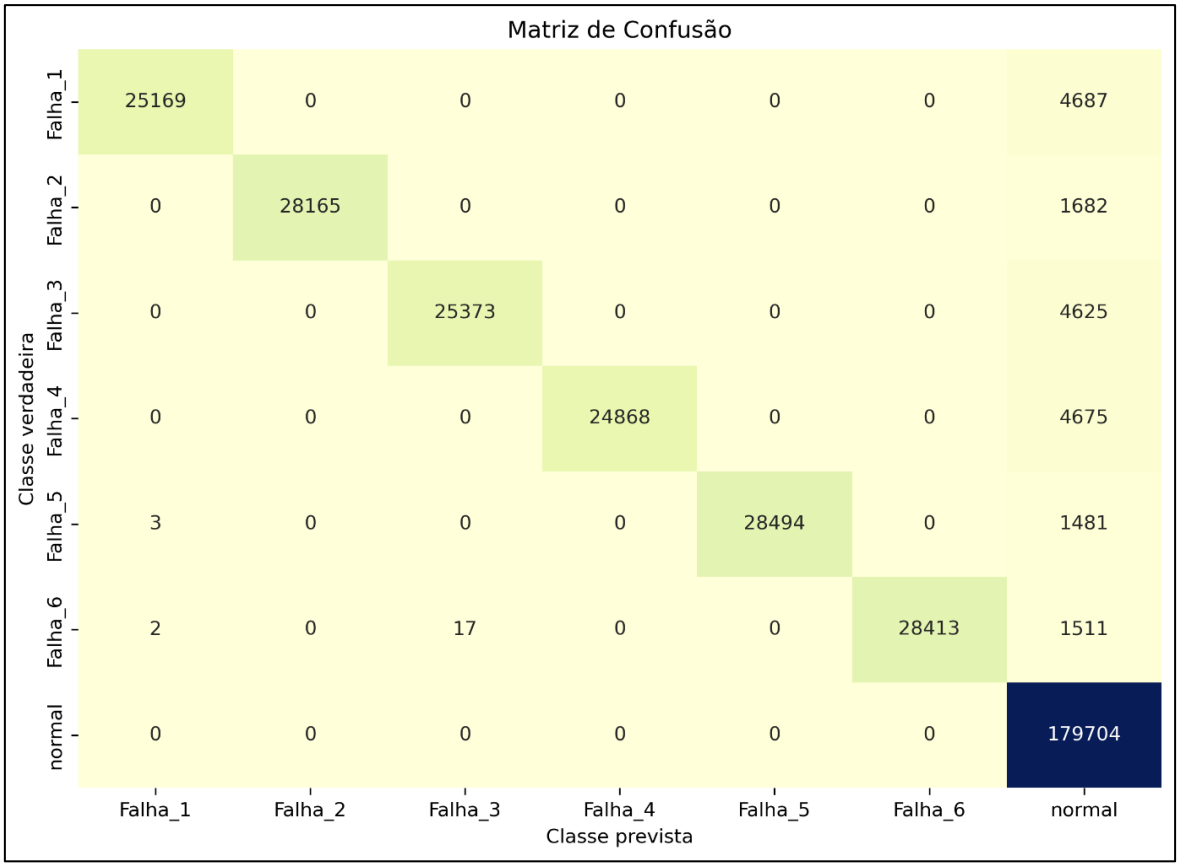
Classe	Precisão	Recall	F1-Score	Support
Falha 1	1,00	0,84	0,91	29.856
Falha 2	1,00	0,94	0,97	29.847
Falha 3	1,00	0,85	0,92	29.998
Falha 4	1,00	0,84	0,91	29.543
Falha 5	1,00	0,95	0,97	29.978
Falha 6	1,00	0,95	0,97	29.943
Normal	0,91	1,00	0,95	179.704

Ao analisar os resultados, podemos constatar que o modelo treinado apresenta um excelente desempenho ao lidar com dados das classes de falhas. No entanto, observa-se um desempenho inferior para a classe “Normal”. Esse desempenho inferior pode ser atribuído aos dados utilizados no treinamento do modelo, uma vez que pequenos desvios nas variáveis de entrada do processo também são classificados como falha, dentro do contexto dessa base de

dados. Assim, o modelo teve dificuldade em diferenciar dados provenientes do processo normal e dados provenientes de um processo com falhas, devido à similaridade entre eles.

A matriz de confusão na Figura 2 fornece um resumo das previsões corretas e incorretas realizadas pelo modelo na etapa de teste.

Figura 2 – Matriz de confusão do modelo de SVM.



Ao analisar a matriz de confusão, é possível observar o ótimo desempenho do modelo ao classificar dados do processo com falhas. No entanto, como mencionado anteriormente, o modelo teve dificuldade em diferenciar dados do processo normal e dados do processo com falhas devido à similaridade de parte desses dados, como demonstrado na coluna à direita da matriz de confusão.

A partir da matriz de confusão, é possível determinar o desempenho do modelo de classificação na detecção da ocorrência de falhas e, após a detecção, a capacidade de diagnosticar o tipo de falha. A Tabela 3 e 4 mostram os resultados obtidos.

Tabela 3 – Desempenho do modelo na tarefa de detecção de falhas.

Desempenho na Detecção de Falhas	
Número de dados com falhas	179.165
Taxa de erro (%)	10,41

A Tabela 3 indica que o modelo apresenta uma taxa de erro de 10,41%, classificado incorretamente parte dos dados do processo normal como sendo provenientes de um processo com falhas, devido à similaridade. No entanto, o modelo possui alta eficiência em identificar o tipo de falha existente no processo, apresentando uma taxa de erro de aproximadamente 0,013%, conforme mostrado na Tabela 4.

Tabela 4 – Desempenho do modelo na tarefa de diagnóstico de falhas.

Desempenho no Diagnóstico de Falhas	
Diagnósticos realizados	160.504
Taxa de erros (%)	0,013

Uma possível solução para aumentar a eficiência do modelo na tarefa de detecção das falhas seria a definição de um intervalo aceitável de valores em torno dos atributos previsores. Dentro desse intervalo, pequenas variações (ruídos) não seriam classificadas incorretamente como falhas, sendo atribuídas a ruídos. Dessa forma, o problema de similaridade de uma fração dos dados do processo normal com os dados do processo com falhas seria eliminado.