Detecção de falhas em máquinas industriais utilizando One Class SVM e técnicas de redução de dimensionalidade

Vinícius Araújo Germano Romão
Departamento de Engenharia de Controle e Automação
Escola Politécnica de Pernambuco
Universidade de Pernambuco
Recife-PE
vagr@poli.br

I. Introdução

A Indústria 4.0, também conhecida como a Quarta Revolução Industrial, descreve a integração de tecnologias avançadas no ambiente industrial para criar sistemas inteligentes e autônomos. Essa mudança tecnológica mistura automação, internet das coisas, computação em nuvem, inteligência artificial e outras tecnologias digitais para transformar os processos industriais tradicionais [1].

Nesse sentido, a Inteligência Artificial (IA) e o Machine Learning (ML) desempenham papéis cruciais na Indústria 4.0, especialmente em áreas como automação de processos, otimização de operações e manutenção preditiva [1]. Utilizando algoritmos de ML, a IA pode analisar dados de sensores, históricos de manutenção e condições operacionais para prever falhas em equipamentos antes que ocorram. Isso permite que as empresas realizem intervenções preventivas, melhorando a eficiência operacional, reduzindo os custos de manutenção, garantindo a operação contínua de equipamentos críticos e otimizando a cadeia de suprimentos para a disponibilidade de peças de reposição [2].

No contexto da manutenção preditiva, o algoritmo One Class Support Vector Machine (OCSVM) é utilizado para detecção de anomalias, sendo eficaz em conjuntos de dados desequilibrados, onde há muito mais exemplos de operação normal do que de falhas [3]. Ele aprende a delimitar a fronteira que melhor separa os dados normais do ponto de origem em um espaço de características de alta dimensão e, durante a previsão, classifica novos pontos de dados com base na distância deles em relação a essa fronteira, considerando pontos fora dela como anomalias ou potenciais falhas [3]. Isso torna o sistema robusto para detectar falhas precoces e prevenir paradas inesperadas nas máquinas industriais permitindo intervenções programadas antes que ocorram falhas graves, reduzindo o tempo de inatividade das máquinas e aumentando a vida útil dos seus componentes [4].

Apesar do OCSVM não ser o único método para detecção de falhas e anomalias [5], ele ainda é utilizado em conjunto com algoritmos de aprendizado profundo, redes neurais recorrentes, e métodos baseados em estatísticas para melhorar a robustez e a precisão das detecções [6] [7] [8] .

O objetivo deste trabalho é aprimorar o desempenho do OCSVM em um conjunto de dados de manutenção preditiva, utilizando PCA (Análise de Componentes Principais) utilizando essa técnicas de redução de dimensionalidade para aumentar a precisão na detecção de anomalias. O desempenho do algoritmo

PCA-OCSVM foi comparado com o OCSVM sem técnicas de redução de dimensionalidade, bem como com o OCSVM aplicado em conjunto com a técnica de redução de dimensionalidade por incorporação de vizinhos estocásticos distribuídos (t-SNE-OCSVM).

II. Trabalhos Relacionados

Existem diversos métodos para detecção de anomalias e falhas como é muito mostrado no artigo “Semi-supervised anomaly detection algorithms: A comparative summary and future research directions” [5] onde os autores compararam 29 algoritmos de detecção de anomalia em 95 datasets desbalanceados, entre esses o OCSVM apresentou um bom desempenho geral AUC, mas resultados de classificação ruins em datasets onde o número de objetos foi igual ou maior que 1.460.

Dessa forma, outros artigos abordam algoritmos que podem ser utilizados para melhorar a performance do classificador OCSVM como em “Anomaly detection in telemetry data using a jointly optimal one-class support vector machine with dictionary learning”[6] onde é utilizado o JODL (Jointly Optimal Dictionary Learning) que é um método de aprendizado de máquina para representar dados de forma esparsa e eficiente [6] , esse tratamento serve para melhorar o desempenho do algoritmo de OCSVM, utilizando o JODL-OCSVM os autores obtiveram uma melhora da precisão da detecção, do F1-score, da taxa de falsos positivos e da taxa de falsos negativos em comparação com outros métodos existentes de detecção de anomalias.

Outro artigo que aborda melhorias na extração de dados para melhorar a performance do OCSVM é o “KDE-OCSVM model using Kullback-Leibler divergence to detect anomalies in medical claims” [7] que explora a divergência Kullback-Leibler e o método KDE para estimar o parâmetro do modelo OCSVM para melhorar o desempenho do modelo.

Já o artigo “Two methods of selecting Gaussian kernel parameters for one-class SVM and their application to fault detection”[8] propõe dois métodos para selecionar o parâmetros do kernel no OCSVM, esses métodos são baseados na análise dos vizinhos mais distantes e mais próximos e na "abertura" das fronteiras de decisão. Os resultados mostraram que esses métodos podem ajudar a selecionar parâmetros adequados, melhorando o desempenho dos classificadores na detecção de falhas.

III. Metodologia

1. Dataset

Devido a dificuldade de obter dados reais de máquinas para detecção de falhas\anomalias, foi utilizado nesse projeto um conjunto de dados do UCI Machine Learning Repository [9], esse conjunto de dados sintético foi feito de maneira que reflete a manutenção preditiva e detecção de anomalias\falhas encontrada na indústria, abrangendo 10.000 pontos de dados com 14 características em colunas. Essas características incluem:

- **UID**: identificador único variando de 1 a 10.000.
- **ProductID:** Identificador do tipo de produto em questão
- **Type** : Consiste nas letras L, M ou H para baixo (50% de todos os produtos), médio (30%) e alto (20%) como variantes de qualidade dos produtos da máquina.
- **Air temperature [K]**: gerada usando um processo de caminhada aleatória normalizado para um desvio padrão de 2 K em torno de 300 K.
- **Process temperature [K]**: gerada usando um processo de caminhada aleatória normalizado para um desvio padrão de 1 K, adicionado à temperatura do ar mais 10 K.
- **Rotational speed [rpm]**: calculada a partir de uma potência de 2860 W, sobreposta com um ruído normalmente distribuído.
- **Torque [Nm]**: valores de torque são normalmente distribuídos em torno de 40 Nm com um σ = 10 Nm e sem valores negativos.
- **Tool wear [min]**: as variantes de qualidade H/M/L adicionam 5/3/2 minutos de desgaste da ferramenta no processo.

Além disso, o conjunto de dados inclui um rótulo de "falha da máquina", que indica se a máquina falhou ou não. Esse rótulo está normalizado, com 0 representando a ausência de falha e 1 indicando a ocorrência de uma falha. Além disso, há uma coluna adicional que especifica o tipo de falha, que pode ser:

- Heat Dissipation Failure
- Power Failure

- Overstrain Failure
- Tool Wear Failure
- Random Failures

2. Análise e Transformação dos Dados

Analisando nosso target podemos observar que ele é um caso de dados desbalanceados pois, temos 96.61% dos dados como não falha (0) e 3.39% dos dados como falha (1). Como podemos observar na Figura 1.

```
# DESBALANCEAMENTO DO TARGET


distribuicao = dados['Target'].value_counts()
pct = dados['Target'].value_counts(normalize=True) * 100

print(pct)
```

```
Target
0    96.61
1     3.39
```

Figura 1

Analisando os valores dos atributos do dataset, conforme mostrado na Figura 2, o atributo UDI atua apenas como índice dos dados, mostrando a sequência de medições, sendo irrelevante para determinar a ocorrência de falhas na máquina. Da mesma forma, o atributo PRODUCT ID na Figura 2 funciona apenas como um identificador específico do produto, sem relevância na análise de falhas. Em relação ao atributo Failure Type, embora informe o tipo de falha como foi mostrado na seção III.1 , foi também desconsiderado na análise, uma vez que nosso foco está em determinar a ocorrência de falhas, não suas causas. Os atributos acima então foram excluídos como consta na Figura 3.

```
for coluna in dados.columns:
    valores_unicos = dados[coluna].unique()
    print(f"'{coluna}': {valores_unicos}")
```

```
'UDI': [    1     2     3 ...  9998  9999 10000]
'Product ID': ['M14860' 'L47181' 'L47182' ... 'M24857' 'H39412' 'M24859']
'Type': ['M' 'L' 'H']
'Air temperature [K]': [298.1 298.2 298.3 298.5 298.4 298.6 298.7 298.8 29
```

Figura 2

```
dados.drop(columns=['UDI','Product ID','Failure Type'], inplace=True)
```

Figura 3

Ainda observando a Figura 2 vemos que o atributo Type tem três letras M, L e H, essas letras como explicadas na seção III.1 tem a ver com a qualidade do produto das máquinas sendo M para Medium (média qualidade), L para Low (baixa qualidade) e H para High (Alta qualidade). Dessa forma foi preciso realizar a transformação simbólico-numérico codificação 1-de-c [10] onde que para um atributo que possui c valores possíveis, são gerados c novos atributos binários. Cada um desses atributos binários representa uma posição específica e indica se um determinado valor do atributo original está presente ou não. Assim, apenas uma posição na nova sequência binária será igual a 1, representando o valor correspondente do atributo original para um objeto específico, dessa forma utilizamos o método pandas.get_dummies para realizar essa conversão, como ilustrado nas Figuras 4 e 5.

```
dados = pd.get_dummies(dados, columns=['Type'], prefix='Type')
dados['Type_H'] = dados['Type_H'].astype(int)
dados['Type_M'] = dados['Type_M'].astype(int)
dados['Type_L'] = dados['Type_L'].astype(int)
```

Figura 4

Type_H	Type_L	Type_M
0	0	1
0	1	0
0	1	0
0	1	0
0	1	0
...	...	...
0	0	1
1	0	0
0	0	1
1	0	0
0	0	1

Figura 5

Uma vez feito a exclusão e transformação dos dados simbólicos-numéricos podemos observar os outliers dos atributos numéricos, analisando a Figura 6 vemos a presença de outliers no atributos Rotational Speed e Torque o que pode nos indicar que esses atributos podem ser os maiores responsáveis pela falha do equipamento.

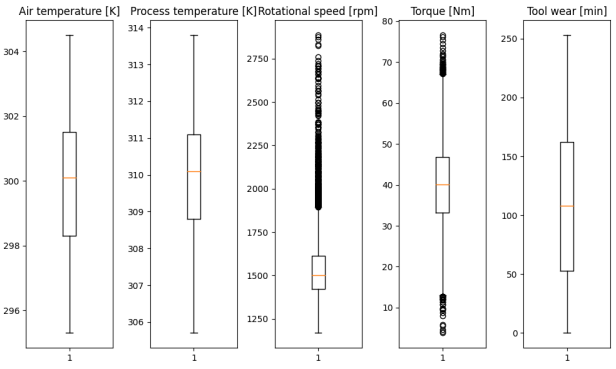

Figura 6

Vale mencionar que devido ao código ser sintético não foram encontrados linhas duplicadas, valores ausentes e nem colunas completamente duplicadas sendo essas informações confirmadas pelas Figura 7 e 8.

```python
# Detecção de Dados Incompletos

missing_values = dados.isnull().sum()

print(f"\nValores ausentes por feature:\n")
print(missing_values)
print(f"\n")

# Detecção de dados redundantes

duplicated_rows = dados[dados.duplicated()]

if  duplicated_rows.empty:
    print("Não tem linhas duplicadas.")
else:
    print("Linhas duplicadas:")
    print(duplicated_rows)

duplicated_columns = dados.columns[dados.T.duplicated()]

if len(duplicated_columns) > 0:
    print("Colunas com valores completamente duplicados:")
    print(duplicated_columns)
else:
    print("Não tem colunas com valores completamente duplicados.")
```

Figura 7

```
Valores ausentes por feature:

Air temperature [K]        0
Process temperature [K]    0
Rotational speed [rpm]     0
Torque [Nm]                0
Tool wear [min]            0
Type_H                     0
Type_L                     0
Type_M                     0
Target                     0
dtype: int64


Não tem linhas duplicadas.
Não tem colunas com valores completamente duplicados.
```

Figura 8

Podemos gerar outros insights, como a matriz de correlação, que é uma tabela mostrando a relação entre os atributos. Cada valor na matriz indica a força e a direção da relação linear entre dois atributos, variando de -1 (correlação negativa perfeita) a 1 (correlação positiva perfeita), enquanto valores próximos de 0 indicam pouca ou nenhuma correlação. Ao analisar a matriz de correlação na Figura 9 observamos uma alta correlação de 0.88 entre os atributos de Process Temperature e Air Temperature. No entanto, essa correlação não é alta o suficiente para justificar a exclusão de qualquer um desses atributos.

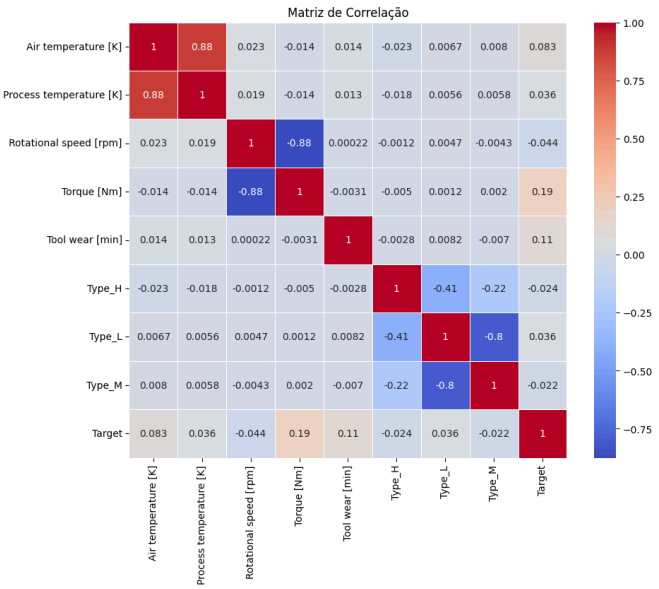

Figura 9

3. Divisão dos dados para aplicação do modelo

Uma vez feita a análise e transformação dos dados podemos passar para o próximo passo onde criamos a função que divide o nosso dataset em treinamento, validação e teste. Esse processo é fundamental para treinarmos, avaliarmos e testarmos o modelo de aprendizado de máquina [11]. A divisão é realizada de modo que separamos nosso dados em dois dataframes: input (entradas ou atributos) e target (o que queremos classificar, as falhas da máquina), dessa forma dividimos cada um desses dois dataframes em 60% para treinamento, 20% para validação e 20% para teste, como podemos observar na Figura 10.

Nessa mesma função ainda realizamos a normalização dos dados que ajusta a escala dos valores das variáveis de um conjunto de dados para que se situem dentro da faixa de 0 e 1. Esse processo é crucial porque garante uniformidade nas variáveis, melhorando a performance dos modelos de aprendizado de máquina que podem ser prejudicados por escalas diferentes [12]. A normalização é feita identificando o valor mínimo e máximo da variável que deseja normalizar, em seguida, para cada valor na sua variável, subtrai-se o valor mínimo encontrado, ajustando

os dados para que o menor valor se torne zero. Depois, é calculada a faixa original, determinando a diferença entre o valor máximo e o valor mínimo da variável, essa diferença representa a faixa original dos dados. Por fim, você pega o resultado da subtração e divide pela faixa original calculada. Isso ajusta os dados para a nova faixa desejada, de 0 a 1, todo esse processo pode ser bem identificado na Figura 11.
No final a função retorna os dados de entrada e targets de treinamento, validação e teste todos já normalizados, como pode ser observado na Figura 11.

```
def divisão_dataset(dados_filtrados):

# Divisão das colunas (target e entradas) - No caso desse

  x_input = dados_filtrados.iloc[:,:-1]
  x_target = dados_filtrados.iloc[:,-1]

  tr = 0.6 # Treinamento - 60%
  vl = 0.2 # Validação - 20%
         # Teste - 20%

# Divisão das linhas do dataset

  train_i = int(tr*len(x_input))
  valid_i = int(vl*(len(x_input)))

  train_input = x_input.iloc[0:train_i,:]
  train_target = x_target.iloc[0:train_i]

  valid_input = x_input.iloc[train_i:train_i+valid_i,:]
  valid_target = x_target.iloc[train_i:train_i+valid_i]

  test_input = x_input.iloc[train_i+valid_i:,:]
  test_target = x_target.iloc[train_i+valid_i:]
```

Figura 10

```
# Normalização dos dados

  (l,cols)=np.shape(train_input)
  minmaxs=[]
  for i in range(cols): # Percorre todas as colunas do dataset
    min = np.min(train_input.iloc[:,i]) # Seleciona todas as linhas da coluna i e calcula o minimo
    max = np.max(train_input.iloc[:,i]) # Seleciona todas as linhas da coluna i e calcula o minimo
    minmaxs.append([min,max]) # Cria o vetor com todas os minimos e mximo por linha

# (X - min)/(max - min) - Reescala todos os dados para os tres datasets com base nos respectivos mi

  for i in range(cols):
    train_input.iloc[:,i] = (train_input.iloc[:,i]-minmaxs[i][0])/(minmaxs[i][1]-minmaxs[i][0]) # S
    valid_input.iloc[:,i] = (valid_input.iloc[:,i]-minmaxs[i][0])/(minmaxs[i][1]-minmaxs[i][0])
    test_input.iloc[:,i] = (test_input.iloc[:,i]-minmaxs[i][0])/(minmaxs[i][1]-minmaxs[i][0])

  return train_input, valid_input, test_input, test_target, valid_target
```

Figura 11

4. One class support vector machine (OCSVM)

Como o dataset escolhido é desbalanceado, foi escolhido então para detectar as anomalias ou falhas , o One Class SVM porque esse modelo de aprendizado de máquina busca aprender uma função que capture a região onde a maioria dos dados de treinamento reside. Para fazer isso, ele tenta encontrar um hiperplano na alta dimensionalidade do espaço dos dados, de modo que a maioria das amostras de treinamento fiquem do mesmo lado do hiperplano, enquanto as amostras que estão fora dessa região são consideradas anomalias [13] . Observarmos melhor como o modelo traça esse hiperplano na Figura 12.

Para testar a capacidade do modelo com os dados é preciso escolher três parâmetros fundamentais: kernel, nu e gamma. Esse primeiro parâmetro é responsável por mapear os dados para um espaço de alta dimensionalidade para encontrar o hiperplano de separação, o mais comum e utilizado neste trabalho devido à sua aplicabilidade em dados não lineares foi o RBF (Radial Basis Function) [13]. O segundo parâmetro a ser escolhido é o nu, ele define a fração de outliers tolerados e a fração mínima de suportes vetoriais, valores altos permitem mais outliers e aumentam a complexidade do modelo, enquanto valores baixos resultam em um modelo mais simples e menos sensível a outliers [13]. Por último, o parâmetro gamma controla a influência de cada amostra de treinamento no modelo, valores altos tornam o modelo mais sensível a variações locais, aumentando o risco de overfitting e valores baixos suavizam o modelo, podendo levar a underfitting [13]..

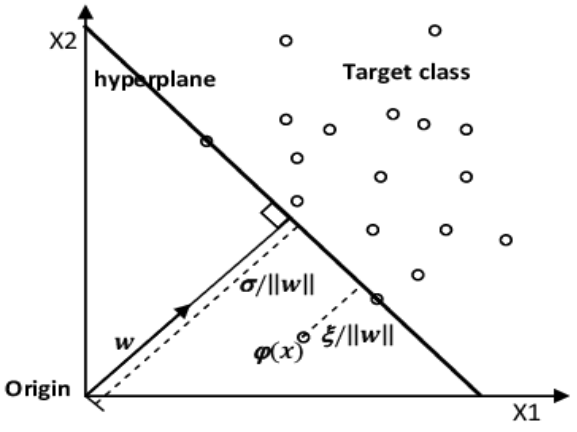

Figura 12

Para utilizar o One Class SVM (OCSVM), primeiro é preciso separar os dados normais das anomalias. Ou seja, separamos os dados onde não aconteceu falha (target=0) dos dados onde aconteceu a falha (target=1). Após isso, utilizando uma busca variando os parâmetros de nu, kernel e

gamma, treinamos o modelo do OCSVM testando cada combinação do vetor dos parâmetros utilizando os dados de treinamento onde não aconteceu a falha. No final do laço, selecionamos o modelo com melhor acurácia com base nos dados de validação contendo os dados onde apresentou e não apresentou falhas.

Válido notar que quando utilizamos o .predict do OCSVM ele retorna 1 para o dado normal e -1 para anomalia, então para utilizar o dados de target da validação para calcular a acurácia é preciso realizar uma pequena conversão onde os dados de validação que são 0 (normal) viram 1 e os dados que são 1 (anomalia) viram -1.

Uma vez calculado o melhor modelo, utilizamos os dados de teste que contém todas as amostras de falhas e não falhas para realizar a predição. A partir daí, com base no target dos dados de teste, calculamos os valores da matriz de confusão, acurácia, precisão,recall, F1 score. Novamente tomando cuidado com o retorno do .predict do OCSVM e realizando a conversão mencionada anteriormente.

Como resultado da aplicação do OCSVM, foi obtido uma acurácia de 0.983 indicando que o modelo acertou 98.3% das previsões feitas. A precisão de 0.988 mostra que, quando o modelo prevê uma classe positiva, ele está correto 98.8% das vezes. O recall de 0.994 revela que o modelo conseguiu identificar 99.4% das verdadeiras instâncias positivas, o que demonstra sua eficácia em detectar a classe positiva quando ela realmente existe. O F1 Score, que combina precisão e recall, foi de 0.991, indicando um equilíbrio muito bom entre esses dois aspectos. Em termos dos valores da matriz de confusão (verdadeiros negativos, verdadeiros positivos, falsos negativos e falsos positivos), o modelo identificou corretamente 16 verdadeiros negativos e 1950 verdadeiros positivos, enquanto cometeu 11 erros ao não identificar verdadeiros positivos e 23 erros ao classificar erroneamente instâncias negativas como positivas. É válido notar também o desempenho em termos de tempo do modelo para rodar o código em torno de 27s para rodar o código. Apesar do bom desempenho geral, o modelo ainda pode ser melhorado devido a quantidade de erros ao detectar as anomalias.

5. t-DISTRIBUTED STOCHASTIC NEIGHBOR EMBEDDING (t-SNE)

Para melhorar o desempenho do nosso modelo OCSVM podemos aplicar uma técnica de redução de dimensionalidade conhecida como t-distributed Stochastic Neighbor Embedding (t-SNE), ela é usada para visualizar dados complexos e de alta dimensão em um espaço reduzido, que no caso do nosso modelo foi escolhido, após testes e o modelo com melhor acurácia se revelou ao utilizarmos a redução para 3 dimensões, como pode ser observado na Figura 13. A técnica preserva a estrutura local dos dados, o que significa que pontos semelhantes em alta dimensão permanecem próximos na visualização reduzida. Para isso, t-SNE calcula a probabilidade de vizinhança entre pontos usando uma distribuição gaussiana em alta dimensão e uma distribuição t de Student com poucos graus de liberdade em baixa dimensão [14].

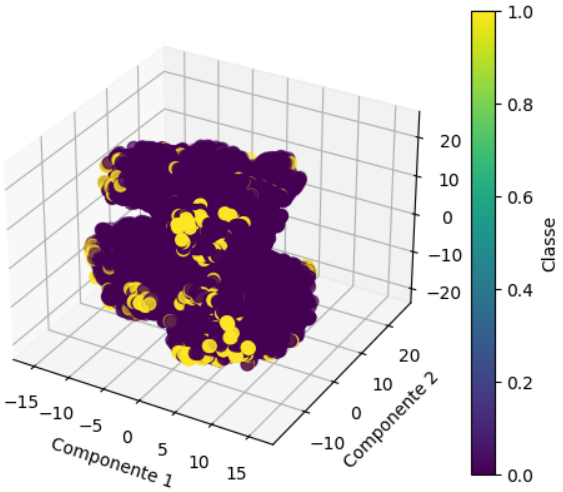

Figura 13

Como resultado da aplicação do OCSVM em conjunto com o PCA, foi obtida uma acurácia de 0.9865, indicando que o modelo acertou 98.65% das previsões feitas. A precisão de 0.9869 mostra que, quando o modelo prevê uma classe positiva, ele está correto 98.69% das vezes. O recall de 0.9995 revela que o modelo conseguiu identificar 99.95% das verdadeiras instâncias positivas, demonstrando sua eficácia em detectar a classe positiva quando ela realmente existe. O F1 Score, que combina precisão e recall, foi de 0.9932, indicando um equilíbrio muito bom entre esses dois aspectos.

Em termos dos valores da matriz de confusão, o modelo identificou corretamente 13 verdadeiros negativos e 1960 verdadeiros positivos, enquanto

cometeu 1 erro ao não identificar um verdadeiro positivo e 26 erros ao classificar erroneamente instâncias negativas como positivas. Em termos de tempo de execução, o modelo levou cerca de 13 segundos para rodar o código, o que representa um desempenho significativamente melhor em comparação com o tempo de 5 minutos observado anteriormente.

6. Principal Component Analysis (PCA)

Outra técnica de redução de dimensionalidade que pode ser aplicada , além do o t-SNE, ao modelo OCSVM é a Análise de Componentes Principais (PCA) que transforma um conjunto de dados com múltiplas variáveis correlacionadas em um conjunto menor de variáveis não correlacionadas, chamadas componentes principais. O PCA centraliza os dados subtraindo suas médias, calcula a matriz de covariância para avaliar a variabilidade e a correlação entre as variáveis, e, em seguida, determina os autovalores e autovetores dessa matriz. Os autovalores indicam a quantidade de variância explicada por cada componente principal, enquanto os autovetores definem a direção desses componentes. Os dados são projetados nas direções dos componentes principais que explicam a maior parte da variância, reduzindo assim a dimensionalidade dos dados [15]. Após testes com a redução de dimensionalidade de 4 até 2 dimensões, observamos que mantendo a dimensão em 2D tínhamos os melhores resultados de acurácia, tal redução pode ser observada na Figura 14.

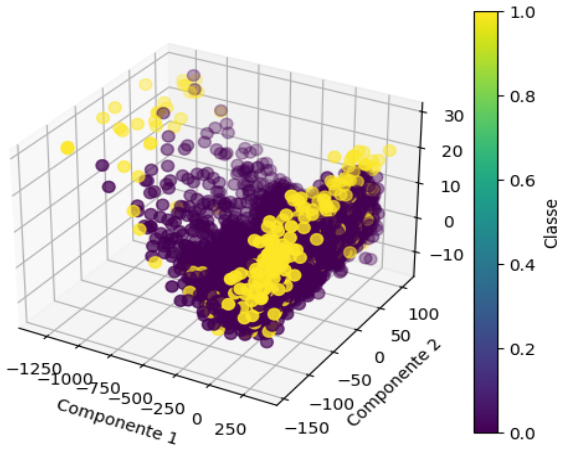

Figura 14

Como resultado da aplicação do OCSVM em conjunto com o PCA, foi obtida uma acurácia de 0.9865, indicando que o modelo acertou 98.65% das previsões feitas. A precisão de 0.9869 mostra que, quando o modelo prevê uma classe positiva, ele está correto 98.69% das vezes. O recall de 0.9995 revela que o modelo conseguiu identificar 99.95% das verdadeiras instâncias positivas, demonstrando sua eficácia em detectar a classe positiva quando ela realmente existe. O F1 Score, que combina precisão e recall, foi de 0.9932, indicando um equilíbrio muito bom entre esses dois aspectos.

Em termos dos valores da matriz de confusão, o modelo identificou corretamente 13 verdadeiros negativos e 1960 verdadeiros positivos, enquanto cometeu 1 erro ao não identificar um verdadeiro positivo e 26 erros ao classificar erroneamente instâncias negativas como positivas. Em relação ao tempo de execução, o modelo levou cerca de 13 segundos para rodar o código.

IV. Resultados

Analisando a tabela 1, temos que a Acurácia, Precisão, Recall e F1 Score dos três métodos são muito parecidos, não apresentando valores com muita variação entre si, não nos informando muito sobre qual melhor modelo escolher. Porém, se analisarmos em relação ao tempo de execução temos que o tSNE-OCSVM apresenta um tempo muito maior em relação aos demais mesmo apresentando métricas semelhantes, se apresentando como uma solução muito complexa para um problema mais simples que sua solução.

Ao observar a Tabela 2 temos que o modelo PCA-OCSVM apresenta as melhores características para detectar uma falha\anomalia devido ao seu desempenho em identificar o maior número de verdadeiros positivos (1960) e o menor número de falsos negativos (1), significando que quase todas os dados positivos, onde não ocorre falha, são detectados. Embora tenha um número ligeiramente maior de falsos positivos (26) em comparação com o modelo com apenas o OCSVM (23), essa diferença é pequena e compensada pelo excelente desempenho na detecção dos verdadeiros positivos e o menor número de falsos negativos. Portanto, entre os três modelos apresentados, o PCA-OCSVM se apresenta como a melhor alternativa.

Modelo	Tempo	Acurácia	Precisão	Recall	F1 Score
tSNE-OCSVM	5 min	98,30%	98,39%	99,90%	99,14%
PCA-OCSVM	8 s	98,65%	98,69%	99,95%	99,32%
OCSVM	27 s	98,30%	98,83%	99,44%	99,14%

Tabela 1

Modelo	Verdadeiro Negativo	Verdadeiro Positivo	Falso Negativo	Falso Positivo
tSNE-OCSVM	7	1959	2	32
PCA-OCSVM	13	1960	1	26
OCSVM	16	1950	11	23

Tabela 2

V. Referências

1. SENSIO. Indústria 4.0: definição, tecnologias e impactos positivos e negativos. Disponível em: https://www.sensio.com.br/blog/industria-4-0-definicao-tecnologias-e-impactos-positivos-e-negativos. Acesso em: 19 jul. 2024.

2. LEEWAYHERTZ. AI in Predictive Maintenance. Disponível em: https://www.leewayhertz.com/ai-in-predictive-maintenance/. Acesso em: 19 jul. 2024.

3. GRABNGOINO. One-Class SVM for Anomaly Detection. Medium. Disponível em: https://medium.com/grabngoinfo/one-class-svm-for-anomaly-detection-6c97fdd6d8af. Acesso em: 19 jul. 2024.

4. SPRINGER. One-Class Support Vector Machine for Anomaly Detection. Disponível em: https://link.springer.com/chapter/10.1007/978-3-030-30490-4_54. Acesso em: 20 jul. 2024.

5. SCIENCEDIRECT. Anomaly Detection Using Machine Learning. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S0950705121001416. Acesso em: 20 jul. 2024.

6. SCIENCEDIRECT. Predictive Maintenance and Anomaly Detection. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S0951832023006312. Acesso em: 20 jul. 2024.

7. SCIENCEDIRECT. Machine Learning for Predictive Maintenance. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S0957417422004705. Acesso em: 20 jul. 2024.

8. SCIENCEDIRECT. PCA for Anomaly Detection. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S0950705114000379. Acesso em: 20 jul. 2024.

9. ARCHIVE. AI4I 2020 Predictive Maintenance Dataset. Disponível em: https://archive.ics.uci.edu/dataset/601/ai4i+2020+predictive+maintenance+dataset. Acesso em: 21 jul. 2024.

10. USP. Mineração de Dados. Disponível em: https://edisciplinas.usp.br/pluginfile.php/4052836/mod_resource/content/4/mineracaodadosbiologicos-parte3.pdf. Acesso em: 21 jul. 2024.

11. SUNIGA, Abner. Conjuntos de Treino, Teste e Validação em Machine Learning. Medium. Disponível em: https://medium.com/@abnersuniga7/conjuntos-de-treino-teste-e-valida%C3%A7%C3%A3o-em-machine-learning-fast-ai-5da612dcb0ed. Acesso em: 21 jul. 2024.

12. HASHTAG TREINAMENTOS.Padronização e Normalização em Ciência de Dados. Disponível em: https://www.hashtagtreinamentos.com/padronizacao-e-normalizacao-em-ciencia-de-dados. Acesso em: 21 jul. 2024.

13. GEEKSFORGEEKS. Understanding One-Class Support Vector Machines. Disponível em: https://www.geeksforgeeks.org/understanding-one-class-support-vector-machines/. Acesso em: 21 jul. 2024.

14. DATACAMP. Introduction to t-SNE. Disponível em: https://www.datacamp.com/pt/tutorial/introduction-t-sne. Acesso em: 21 jul. 2024.

15. CIÊNCIA E NEGÓCIOS. Análise de Componentes Principais (PCA ou ACP). Disponível em: https://cienciaenegocios.com/analise-de-componentes-principais-pca-ou-acp/. Acesso em: 21 jul. 2024.