Um Estudo Sobre o Uso de Modelos de Machine Learning para Manutenção Preditiva Industrial

Pedro Vinícius Meerholz
Centro Universitário SENAI/SC
pedro_meerholz@estudante.sesisenai.org.br

Willian Daniel de Mattos
Centro Universitário SENAI/SC
willian.mattos@edu.sc.senai.br

Dhyonatan Santos de Freitas
Centro Universitário SENAI/SC
dhyonatan.freitas@edu.sc.senai.br

Eduardo Camilo Inacio
Centro Universitário SENAI/SC
eduardo.inacio@edu.sc.senai.br

ABSTRACT

With Industry 4.0, the integration between physical and digital environments enabled some improvements within several production segments. Among these improvements, advancements on the application of machine learning algorithms to predict current and future states of equipment have been gaining attention, specially, for maintenance purposes. This research work presents a comparative experimental study on machine learning algorithms applied to classification of industrial machinery states. After training and evaluating models based on five different algorithms (i.e., Decision Tree, Naive Bayes, Support Vector Machines, XGBoost and Neural Network), some interesting results were obtained. Considering the accuracy, precision, recall and training time of each model, it was observed that some models performed well, while others may not be as suitable for solving the problem. Such good performing models could be used to schedule interventions on a given industrial equipment, avoiding production stoppages.

KEYWORDS

MACHINE LEARNING, APRENDIZAGEM SUPERVISIONADA, REDES NEURAIS, INDÚSTRIA 4.0

1 INTRODUÇÃO

Com a chegada da Quarta Revolução Industrial, também chamada de Indústria 4.0, os equipamentos físicos utilizados dentro do ambiente industrial passam a trabalhar em conjunto com o meio digital e, dessa forma, essa integração entre máquinas e sistemas, quando utilizada da forma correta, pode proporcionar dados relevantes que podem ser utilizados a favor da indústria de diversas maneiras. Uma das maneiras de se utilizar os dados obtidos com essa integração, é aplicá-los junto ao conceito de manutenção preditiva. A manutenção preditiva utiliza informações obtidas através de dados históricos para prever em qual momento, uma determinada máquina precisará de manutenção. Ao prever quando uma máquina irá precisar de manutenção, alguns benefícios são obtidos, tais como a redução de custos sobre manutenção e a redução do tempo de manutenção [1]. A partir disso, a área de machine learning foi uma das quais obteve destaque quando se trata da Indústria 4.0. De forma geral, o machine learning consiste em utilizar algoritmos computacionais e dados para detectar padrões que irão ajudar em alguma tomada de decisão, nesse caso, se é necessário realizar a manutenção do equipamento ou não.

No entanto, para que o machine learning seja aplicado de forma eficiente, algumas etapas devem ser realizadas, sendo elas a coleta e o tratamento de dados. Quando se fala em obter dados para aplicar técnicas de machine learning, existem algumas possibilidades que podem ser exploradas. Primeiramente, é possível produzir os próprios dados, que no caso da indústria, por exemplo, poderiam ser produzidos através da instalação de sensores [2], o qual no cenário industrial seria a principal forma de adquirir esses dados. No entanto, essa pode não ser uma tarefa tão simples e pode trazer novos custos ao processo, por isso, outra opção seria comprar dados de outras instituições que forneçam dados confiáveis e legítimos para aquele determinado fim.

Depois da obtenção de dados, possivelmente será necessário realizar o tratamento desses dados, pois os mesmos não podem ser aplicados diretamente nos algoritmos de machine learning. Durante a coleta de dados, é possível que alguns dados estejam faltando, estejam errados ou não sejam necessários para aquele contexto [1].

Dessa forma, a motivação para esse trabalho tem como base a aplicação de algoritmos de machine learning no contexto da indústria 4.0, tendo em vista que seu objetivo é apresentar uma comparação sobre o uso de determinados modelos de machine learning para classificação, com a finalidade de classificar máquinas industriais como "com defeito" ou "sem defeito". A partir dessa comparação, é possível identificar dentre os modelos com base em algumas métricas, quais são os mais adequados para realizar predições utilizando dados industriais sintéticos. Além disso, também é demonstrado o processo utilizado para chegar aos resultados, abordando desde a manipulação dos dados até a otimização dos hiperparâmetros dos modelos.

O restante do artigo está organizado da seguinte maneira. Na Seção 2 será apresentada a fundamentação teórica acerca dos modelos de machine learning. Logo após, na Seção 3, é realizada uma breve abordagem sobre o ambiente de execução utilizado e qual linguagem e suas bibliotecas foram utilizadas no processo. Na Seção 4, se encontra a proposta do artigo, onde será descrito o processo de tratamento dos dados, o processo de ajuste dos modelos de classificação e seus processos de treinamento. A Seção 6 contém os resultados experimentais, onde é realizada a comparação entre os modelos, seguido pela Seção 7, que apresenta as conclusões e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

A fundamentação teórica deste trabalho consiste em apresentar os modelos de classificação utilizados, apresentar as métricas adotadas para avaliar os modelos e também fundamentar a normalização de dados.

2.1 Árvore de Decisão

Uma árvore de decisão é uma estrutura composta por nodos, sendo eles o nodo raiz, os nodos internos e os nodos folha. Cada nodo possui uma característica diferente, por exemplo, o nodo raiz é onde a árvore se inicia e dele saem os nodos internos, que posteriormente irão gerar os nodos folhas, que representam o resultado da classificação [3]. Os nodos raiz e internos são responsáveis por realizar uma verificação de uma característica do conjunto de dados, sendo que o resultado dessa verificação pode resultar em um nodo interno ou em um nodo folha [2]. A Figura 1 representa a estrutura de uma árvore de decisão, exemplificando as informações descritas neste parágrafo.

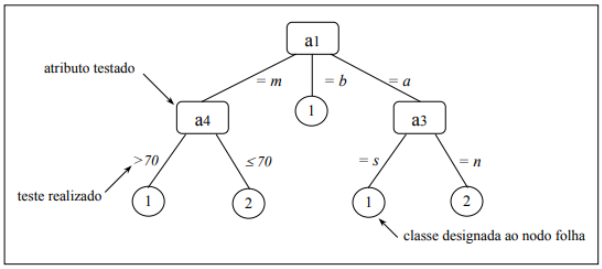

Figura 1: Representação da estrutura de uma árvore de decisão. Fonte: Retirado de [4]

2.2 Classificador Bayesiano Simples

De modo geral, classificadores bayesianos assim como o algoritmo de Naive Bayes, são algoritmos baseados na teoria de Thomas Bayes, usando da probabilidade condicional para realizar a classificação dos registros. Um ponto a destacar, é que os algoritmos de classificação bayesiana simples suportam registros que possuam dados faltantes. Isso porque quando o algoritmo encontra um registro que possui algum dado faltando ele irá ignorá-lo e seguirá com os próximos registros do conjunto de dados, seja na hora de treinar o modelo ou de realizar classificações com ele [3]. Uma outra característica do algoritmo, é que durante o processo de classificação de um registro, o modelo não avalia as características em conjunto, mas sim, de forma independente [2].

2.3 Support Vector Machines

As Support Vector Machines (SVM), são algoritmos de classificação e regressão baseados em estatística. Esse algoritmo tem como objetivo traçar várias retas e dentre delas encontrar a reta que melhor separe duas classes distintas, essa reta também é chamada de hiperplano [3].

Cada hiperplano traçado pelo algoritmo possui um limite de decisão. O limite de decisão é a margem entre o hiperplano e as instâncias mais próximas de cada classe. Essas instâncias são os chamados vetores de suporte. Tendo em vista que o limite de decisão é a margem entre o hiperplano e os vetores de suporte, quando se tem uma margem muito pequena, maior é chance de que os outliers do conjunto de dados interfiram negativamente no desempenho do modelo. Em contrapartida, com uma margem maior, menor será o impacto dos outliers no desempenho do modelo [3, 5].

2.4 XGBoost

O eXtreme Gradient Boosting ou XGBoost, é um algoritmo de Ensemble Learning que utiliza a estrutura do Gradient Boosting. A premissa dos algoritmos de Gradient Boosting é utilizar vários previsores de forma sequencial, onde cada previsor tem o objetivo de corrigir o erro do previsor anterior. Aplicando a premissa dos algoritmos de Gradient Boosting e sabendo que o XGBoost utiliza previsores baseados em árvores de decisão, pode-se entender que o XGBoost faz uso de várias árvores de decisão de forma sequencial para realizar suas tarefas, que podem tanto ser de classificação, quanto de regressão. Vale ressaltar também que, um modelo XGBoost treinado pode ser implantado em sistemas distribuídos, tal como o Hadoop, e, além disso, o algoritmo pode ser acelerado com placas de vídeo compatíveis com CUDA. Por fim, vale mencionar que o algoritmo pode ser utilizado em várias linguagens de programação, tais como Python, Java, R, Julia, C, Ruby e entre outras [5, 6].

2.5 Rede Neural Artificial (ANN)

De forma geral, uma Rede Neural Artificial (ANN) nada mais é do que uma tentativa de emular a estrutura do cérebro humano em um computador, utilizada dentro do campo de Inteligência Artificial [7], que pode ser utilizada para resolver problemas de classificação, reconhecimento de voz [8], categorização, predição[9] e etc.

Nas redes neurais, assim como no cérebro humano, também existem os chamados neurônios, que nas redes neurais, são chamados de neurônios artificiais, os quais podem realizar operações de multiplicação, somatório e funções de ativação. Após realizar uma determinada operação, um neurônio artificial irá retornar uma saída, a qual é o resultado do somatório das entradas multiplicadas pelos seus respectivos pesos, utilizados em uma função de ativação [10]. A Figura 2 apresenta a estrutura de um neurônio artificial.

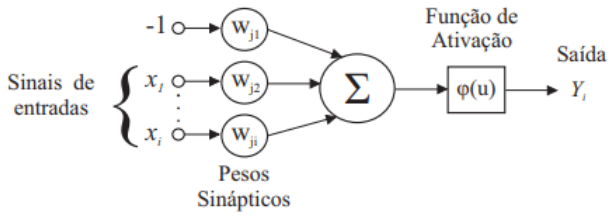

Figura 2: Representação de um neurônio artifical. Fonte: Retirado de [10]

Os neurônios artificiais são organizados em camadas. Existem três tipos de camadas possíveis em uma rede neural: a camada de entrada, a camada oculta e a camada de saída [11]. A camada de entrada, tem como objetivo representar os atributos de entrada [3] e também repassar os valores recebidos para uma camada oculta ou diretamente para a camada de saída [11]. Por representar os atributos de entrada, a quantidade de neurônios nessa camada deve ser igual ao número de variáveis utilizadas. Já a camada de saída irá representar o resultado obtido pelo modelo. Dessa forma, esta camada também precisa de uma determinada quantidade de neurônios, um para cada possível saída [3]. Toda camada será conectada com a camada posterior[10]. Dessa forma, a camada de entrada irá se conectar com a camada de saída ou com alguma camada oculta

(quando existir uma). Já uma camada oculta pode se conectar com outra camada oculta posterior ou com a camada de saída. É possível visualizar essas conexões na Figura 3, que representa a estrutura de uma Rede Neural Artificial composta pela camada de entrada, duas camadas ocultas e a camada de saída.

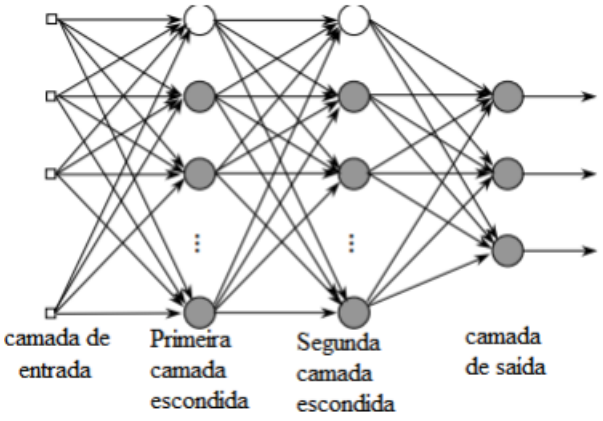

Figura 3: Representação da estrutura de uma Rede Neural Artifical. Fonte: Retirado de [11]

Percebe-se também na Figura 3 que nas conexões entre duas camadas, todos os neurônios da camada anterior se conecatam com todos os neurônios da camada posterior.

2.6 Métricas de Avaliação: Precisão, Revocação e Acurácia

A precisão, a revocação e a acurácia são métricas utilizadas para avaliar o desempenho de um modelo de classificação. Cada uma terá sua relevância dependendo do cenário em que o classificador é aplicado. Por exemplo, a precisão indica a porcentagem de acerto das previsões positivas feitas pelo modelo, ou seja, ela será mais relevante em cenários onde reduzir a quantidade de falsos positivos possuem maior relevância, ou seja, quanto maior a precisão, maior será o número de verdadeiros positivos. Para calcular a precisão de um modelo basta dividir o número de verdadeiros positivos pela soma dos verdadeiros positivos com os falsos positivos, conforme definido na Equação 1.

$$Precisao = \frac{Verdadeiros\ Positivos}{(Verdadeiros\ Positivos + Falsos\ Positivos)} \quad (1)$$

Já a revocação, indica quantos verdadeiros positivos o modelo acertou, por isso, ela é mais utilizada quando minimizar o número de falsos negativos possui maior relevância para o cenário de aplicação do modelo. Para calcular a revocação, basta dividir o número de verdadeiros positivos pela soma dos verdadeiros positivos com os falsos negativos, como descrito na Equação 2.

$$Revocacao = \frac{Verdadeiros\ Positivos}{(Verdadeiros\ Positivos + Falsos\ Negativos)} \quad (2)$$

Por fim, a acurácia indica a exatidão dos acertos obtidos através das previsões realizadas pelos modelos. A acurácia de um modelo é calculada através da Equação 3 [12],

$$Acc = \frac{VN + VP}{VP + FN + VN + FP} \quad (3)$$

onde VP é o equivalente ao número de verdadeiros positivos, VN ao número de verdadeiros negativos, FN ao número de falsos negativos e FP ao número de falsos positivos [12].

2.7 Normalização

Quando se tem um conjunto de dados onde as características possuem grandes diferenças de intervalos, é necessário transformar esses dados colocando-os em valores próximos, para que assim, o modelo não faça a distribuição de pesos entre as características de forma errônea. Uma das formas de transformar os dados é realizar a normalização, que irá deixar todos os valores entre 0 e 1, ou, entre -1 e 1 caso existam valores negativos [13]. A Equação 4 mostra a fórmula matemática utilizada para realizar a normalização dos dados [13].

$$x' = \frac{x - min}{max - min} \quad (4)$$

Para exemplificar a Equação 4, tem-se um caso onde temos uma feature onde o menor valor (representado na equação pelo *min*) é 12.000 e o maior valor (representado na equação pelo max) é 98.000. Sabendo dos valores de min e max, deseja-se normalizar o número 73.000, representando o valor de x. Nessas condições, a Equação 5 representa a estrutura da Equação 4 com os valores já preenchidos [14].

$$x' = \frac{73.000 - 12.000}{98.000 - 12.000} \quad (5)$$

2.8 Validação Cruzada

A validação cruzada é uma técnica utilizada para auxiliar na avaliação do modelo de machine learning. Essa técnica consiste em separar o conjunto de dados de treinamento em K partes, os chamados folds. A partir disso, o modelo será treinado K vezes com $K - 1$ partes e será testado com a parte restante. Ou seja, em um caso onde o conjunto de treinos é separado em 10 partes, o modelo será treinado 10 vezes, tendo em vista que cada treino será realizado com 9 das partes separadas e irá validar com a parte que sobrou. Utilizando a validação cruzada, é possível ter uma noção do desempenho do modelo de machine learning utilizado, pois, com essa técnica, pode-se visualizar o desempenho do modelo durante o treino e teste. Sabendo do desempenho do modelo, é possível determinar se o modelo está se sobreajustando ou não, para que as medidas necessárias possam ser tomadas [5].

2.9 Overfitting

O overfitting é um problema que pode vir a ocorrer em tarefas de aprendizado supervisionado, tal como a classificação. O overffiting ocorre quando o modelo de machine learning possui um ótimo desempenho com os dados de treino mas, em contrapartida, não possui um desempenho adequado com os dados de teste. Dessa forma, pode-se dizer que o modelo com overfitting caracteriza um modelo que memoriza os dados de treinamento e não consegue se ajustar a dados diferentes daqueles os quais foram utilizados em sua etapa de treino [15].

3 AMBIENTE DE EXECUÇÃO E BIBLIOTECAS

Para realizar o treinamento dos modelos, foi utilizado o ambiente do Google Collaboratory, onde o notebook criado com a plataforma foi disponibilizado na plataforma GitHub [16]. Em relação às tecnologias utilizadas, a linguagem de programação escolhida para o desenvolvimento foi a linguagem Python (versão 3.7.13), junto de suas bibliotecas Pandas (versão 1.3.5), scikit-learn (versão 1.0.2) e XGBoost (versão 0.90). A biblioteca Pandas, foi utilizada para realizar todas as manipulações com os dados e gerar análises descritivas, enquanto o scikit-learn foi utilizado para que fosse possível aplicar os modelos de Árvore de Decisão, SVM, Naive Bayes e o Multilayer Perceptron (representa uma ANN). Outra biblioteca utilizada para aplicar modelos de machine learning, foi a XGBoost, utilizada especificamente para o modelo de mesmo nome. Por fim, para gerar os gráficos presentes no trabalho, foi utilizada a biblioteca matplotlib, na sua versão 3.2.2.

4 MANIPULAÇÃO DO CONJUNTO DE DADOS

O conjunto de dados utilizado para aplicar os modelos de classificação foi obtido do site UCI machine learning Repository [17] e consiste de informações artificiais sobre máquinas industriais. Esse conjunto de dados possui um total de 10.000 registros e 10 colunas, que possuem dados categóricos, contínuos e ordinais. A Seção 4.1 tem como objetivo realizar um resumo sobre as colunas do conjunto de dados.

4.1 Detalhamento das colunas do conjunto de dados

A coluna *UDI* tem como objetivo definir um ID para os registros presentes no conjunto de dados. Na sequência, se encontra a coluna *Product ID*, que consiste em um valor serial composto por letras e números. A coluna *Type*, define a qualidade da máquina, podendo ser L para baixa, M para média e H para alta; após, na coluna *Air Temperature*, pode-se visualizar a temperatura em Kelvin da temperatura do ar no momento em que a máquina apresentou falha. Na sequência, a coluna *Process Temperature* indica a temperatura em Kelvin do processo no momento em que a máquina apresentou falha. Já na coluna *Rotational Speed*, se tem uma medida em RPM (rotações por minuto) que representa a velocidade de rotação no momento em que a máquina apresentou falha. A coluna *Tool wear* representa o desgaste da máquina no momento em que a mesma apresentou falha. A penúltima coluna, nomeada de Target, é umas das colunas que podem ser usadas como classes/rótulos. Seus valores podem ser 0 para não falha ou 1 para falha. Para finalizar, a coluna *Failure Type*, que assim como a coluna Target é uma das colunas que podem ser usadas como classes/rótulos. Seus valores podem ser: No Failure, Heat Dissipation Failure, Power Failure, Overstrain Failure, Tool Wear Failure, Random Failures. Cada um desses valores representa um tipo de falha diferente.

Para complementar a descrição acima, a Tabela 1 apresenta uma análise descritiva dos dados das colunas *Air Temperature*, *Process Temperature*, *Rotational Speed*, *Torque* e *Tool Wear*, com a finalidade de detalhar um pouco mais os dados que foram utilizados.

Tabela 1: Análise Descritiva das colunas *Air Temperature*, *Process Temperature*, *Rotational Speed*, *Torque* e *Tool Wear*

Métrica	*Air Temp.*	*Process Temp.*	*Rotat. Speed*	*Torque*	*Tool Wear*
Registros	9937.00	9937.00	9937.00	9937.00	9937.00
Média	300.00	310.00	1538.72	39.99	107.43
Mínimo	295.30	305.70	1168.00	3.80	0.00
25%	298.30	308.80	1423.00	33.20	53.00
50%	300.10	310.10	1503.00	40.10	107.00
75%	301.50	311.10	1612.00	46.80	162.00
Máximo	304.50	313.80	2886.00	76.60	251.00
Desvio Padrão	2.00	1.48	179.24	9.96	63.36

4.2 Pré-processamento dos Dados

Para preparar os dados para serem utilizados pelos algoritmos de classificação, foi necessário realizar algumas manipulações neles. Primeiramente, na coluna Failure Type, os registros que pertencem às classes Tool Wear Failure e Random Failures foram removidas pois elas não possuíam dados suficientes para que a máquina pudesse observar seus padrões corretamente, podendo implicar na redução do desempenho do modelo. No decorrer do artigo, não serão estas as classes utilizadas como variáveis alvo, mas sim, as variáveis presentes na coluna Target, que foram detalhadas na Seção 4.1.

Com isso, tendo removido alguns dados e as colunas consideradas irrelevantes para este estudo, foi o momento de realizar transformações nos dados da coluna Type. Originalmente, essa coluna possuía dados categóricos no formato de texto que não podiam ser utilizados nos modelos. A coluna foi dividida em três colunas, uma para cada valor presente na coluna Type, sendo que essas colunas podem conter dois valores, um ou zero, assim como mostra a Figura 4.

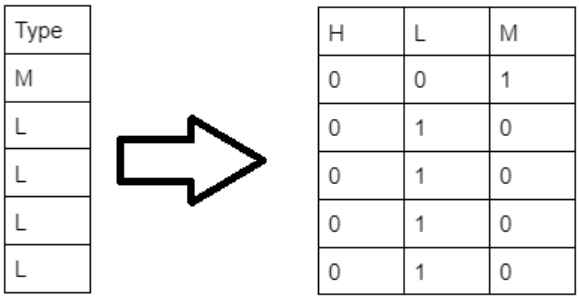

Figura 4: Exemplificação da divisão da coluna Type

Após manipular os dados da coluna Type, o conjunto de dados foi dividido entre as features e as variáveis alvo. Por fim, foi necessário colocar os dados dentro de um mesmo intervalo de valores e para isso foi realizada a normalização dos dados, colocando os valores em um intervalo entre 0 e 1. A distribuição dos dados normalizados pode ser observada na Figura 5. É possível também comparar a Tabela 1 com a Tabela 2 para notar a diferença dos dados antes e depois de passarem pelo processo de normalização.

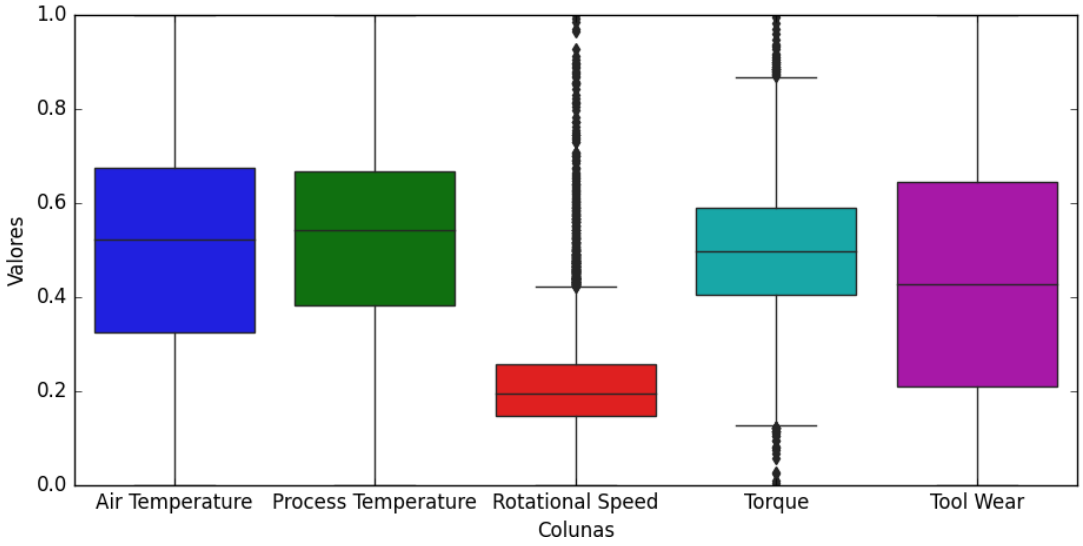

Figura 5: Distribuição dos dados após o processo de normalização

Tabela 2: Análise Descritiva do dataset após o processo de normalização

Métrica	*Air Temp.*	*Process Temp.*	*Rotat. Speed*	*Torque*	*Tool Wear*
Registros	9937.00	9937.00	9937.00	9937.00	9937.00
Média	0.51	0.53	0.21	0.49	0.42
Mínimo	0.00	0.00	0.00	0.00	0.00
25%	0.32	0.38	0.14	0.40	0.21
50%	0.52	0.54	0.19	0.49	0.42
75%	0.67	0.66	0.25	0.59	0.64
Máximo	1.00	1.00	1.00	1.00	1.00
Desvio Padrão	0.21	0.18	0.10	0.13	0.25

4.3 Separação de Subconjuntos de Treino, Teste e Validação

Com os dados pré-processados, o conjunto de dados foi separado em um conjunto de treino e um conjunto de testes. O conjunto de dados teve uma distribuição de 80% dos dados para o conjunto de treino, 10% para o conjunto de testes e os outros 10% para o conjunto de validação, sendo que os rótulos/classes foram divididos proporcionalmente entre os conjuntos de forma randomizada.

5 AJUSTE DE HIPERPARÂMETRO DOS MODELOS

Com os conjuntos de dados de treino, teste e validação separados, os dados estavam devidamente prontos para serem utilizados pelos algoritmos de classificação. Então, era chegado o momento de ajustar os modelos antes de realizar o treinamento de cada um deles.

Para que os hiperparâmetros dos modelos pudessem ser ajustados, foram utilizados os métodos Grid Search [18] e Randomized Search [19] para avaliar quais os melhores valores para os hiperparâmetros de cada modelo e após isso foi aplicada a técnica da validação cruzada junto ao conjunto de dados de validação, com o intuito de verificar se os modelos estavam tendo um bom desempenho com os hiperparâmetros ajustados e se não possuíam overfitting. Foi notado que ao utilizar os hiperparâmetros encontrados pelos métodos Grid Search e Randomized Search, os modelos estavam com overfitting, então alguns dos hiperparâmetros encontrados foram ajustados manualmente com o intuito de remover ou reduzir o overfitting dos modelos.

Para dar início a descrição dos hiperparâmetros e seus valores, vale mencionar que o modelo Naive Bayes não possui nenhum hiperparâmetro para ser ajustado. Em relação ao modelo de árvore de decisão, foi ajustado o hiperparâmetro `max_depth`. Esse hiperparâmetro controla a profundidade da árvore e nesse caso ele foi ajustado com o número 2. Já em relação ao SVM, foram ajustados os hiperparâmetros `C`, `kernel` e `gamma`. Na Tabela 3, podem ser observados quais os valores foram atriuídos para cada um dos hiperparâmetro mencionados.

Tabela 3: Valores dos hiperparâmetros ajustados (SVM)

Hiperparâmetro	Valor Ajustado
C	100
`kernel`	rbf
`gamma`	`scale`

O hiperparâmetro `C` serve como um parâmetro de regularização para a SVM, afetando diretamente o tamanho do limite de decisão do modelo. Com isso, quanto maior for o valor de `C` maior será a regularização (realizada sempre que o modelo classifica um registro de forma incorreta) da SVM e menor será o seu limite de decisão. Consequentemente, quanto menor o valor de C, menor será a regularização do modelo e maior será o seu limite de decisão. O

hiperparâmetro de kernel foi definido como rbf (Radial Basis Function) pois os dados utilizados não são linearmente separáveis. E por fim, o hiperparâmetro gamma foi definido como scale, que utiliza de uma fórmula matemática para definir qual será o coeficiente do kernel da SVM.

O modelo XGBoost, teve 6 hiperparâmetros ajustados, sendo eles: `booster`, `eta`, `eval_metric`, `gamma`, `max_depth` e `objective`. A Tabela 4 apresenta os valores ajustados para cada hiperparâmetro do XGBoost.

Tabela 4: Valores dos hiperparâmetros ajustados (XGBoost)

Hiperparâmetro	Valor Ajustado
`booster`	`dart`
`eta`	0.15
`eval_metric`	`error`
`gamma`	6
`max_depth`	4
`objective`	`binary:hinge`

O hiperparâmetro `booster` define qual será o tipo de modelo utilizado (baseado em árvore ou regressão). Ao ajustá-lo como dart, o XGBoost irá utilizar modelos baseados em árvore. Para ajustar a taxa de aprendizagem do modelo, foi utilizado o hiperparâmetro `eta`. Já o hiperparâmetro `eval_metric`, define a métrica utilizada para avaliar o modelo ao utilizar os dados de validação e nesse caso, como o problema abordado é um problema de classificação, o seu valor foi definido como error, assim como mostrado na Tabela 4. `Gamma` irá afetar o tamanho das árvores de decisão utilizadas pelo XGBoost, restringindo o quão grande a árvore poderá ficar diante das divisões de seus nodos. Outro hiperparâmetro que afeta o tamanho das árvores do modelo é o `max_depth`, que determina qual a profundidade máxima do modelo. E por último, foi ajustado o hiperparâmetro `objective`, que é o responsável por definir a função de perda do modelo.

E por fim, os hiperparâmetros da Rede Neural Artificial (ANN). Para este modelo, os seguintes cinco hiperparâmetros foram ajustados: `validation_fraction`, `solver`, `random_state`, `max_iter`, `hidden_layer_sizes` e `activation`. A Tabela 5 mostra os valores ajustados para cada hiperparâmetro mencionado.

Tabela 5: Valores dos hiperparâmetros ajustados (ANN)

Hiperparâmetro	Valor Ajustado
`validation_fraction`	0.2
`solver`	adam
`random_state`	0
`hidden_layer_sizes`	(17,)
`activation`	tanh

O hiperparâmetro `validation_fraction` determina a quantidade de dados que serão utilizados para a etapa de validação do modelo, que nesse caso será de 20%. Já o hiperparâmetro `solver` irá definir qual será o otimizador utilizado na ANN. Para definir qual será o valor inicial dos pesos e do bias do modelo, é utilizado o hiperparâmetro `random_state`. Enquanto isso, o hiperparâmetro `hidden_layer_sizes` irá definir a quantidade de neurônios que a camada oculta do modelo terá. E por fim, o hiperparâmetro `activation` irá definir qual será a função de ativação utilizada nas camadas ocultas da ANN.

6 RESULTADOS EXPERIMENTAIS

Nesta seção, são apresentados os resultados obtidos através do treinamento dos modelos de machine learning considerados, trazendo uma comparação entre eles. Dessa forma, a comparação dos modelos foi feita com base na acurácia, na precisão, na revocação e no tempo de treinamento de cada um.

6.1 Avaliação dos Modelos

Após ajustar os modelos e treiná-los, eles foram utilizados para realizar predições com base nos dados de teste e a partir da predição realizada por eles, foi possível avaliar o desempenho de ambos os modelos com dados diferentes daqueles nos quais foram treinados. É possível observar o desempenho dos modelos pelas métricas de acurácia, revocação, precisão e tempo de treinamento nas Figuras 6, 7, 8 e 9, respectivamente.

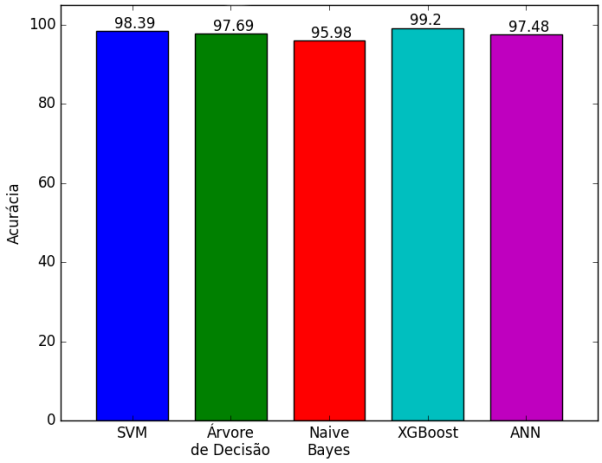

Figura 6: Comparativo das acurácias obtidas entre os modelos

Ao visualizar a Figura 6 e analisar a acurácia isoladamente, é possível observar que o XGBoost foi superior dentre os cinco modelos, chegando aos 99.2%. Pouco abaixo desse ponto, se encontra a SVM, que atingiu os 98.39% de acurácia, seguida pela árvore de decisão, que alcançou os 97.69%, ficando 0.6% e 1.51% abaixo dos resultados da SVM e do XGBoost respectivamente. Quase igualando a árvore de decisão se encontra a ANN, que alcançou os 97.49% de acurácia e ficando apenas 0.21% atrás do modelo baseado em árvore. Ainda assim, a ANN ficou à frente do modelo Naive Bayes, que atingiu somente 95.98% de acurácia. No entanto, não é correto afirmar que o modelo é melhor somente porque sua acurácia foi superior e, por esse motivo, apresentamos as métricas de precisão (Figura 8) e revocação (Figura 7).

Nesse caso, como o objetivo é classificar as máquinas como "com defeito" e "sem defeito", a métrica que pode ser considerada mais apropriada é a revocação, pois caso ocorra um falso negativo a

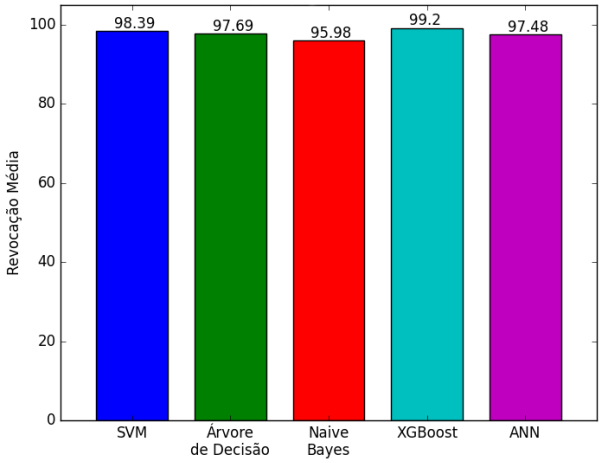

Figura 7: Comparativo das revocações obtidas entre os modelos

máquina será classificada como "sem falha" quando na verdade deveria ser classificada como "com falha". Então, ao olhar para a métrica de revocação na Figura 7, percebe-se que o modelo XGBoost foi cerca de 0.8% melhor em relação a SVM, sendo que o XGBoost alcançou os 99.2% e a SVM 98.39% de revocação. Ao olhar para os modelos que ficaram um pouco abaixo, na casa dos 97%, encontram-se os modelos de árvore de decisão e a rede neural. Ao comparar os dois, é visível que a diferença foi quase nula, sendo de apenas 0.21%, onde a árvore de decisão alcançou os 97.69% e a rede neural, 97.48% de revocação. No entanto, ainda assim, a rede neural foi 1.5% mais efetiva que o modelo de Naive Bayes, que ficou na faixa dos 95.98% nesta métrica.

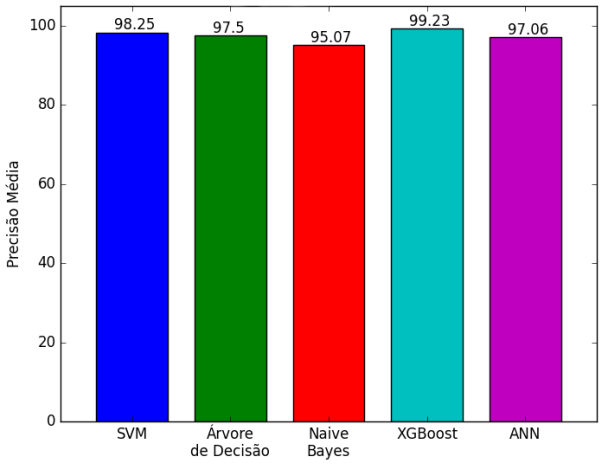

Figura 8: Comparativo das precisões obtidas entre os modelos

Agora, considerando a precisão, percebe-se, na Figura 8, que ao XGBoost foi cerca de 1% mais efetivo em relação a SVM, sendo que o XGBoost alcançou os 99.23% e a SVM alcançou 98.25% de revocação. Ao olhar para os modelos que estão dentro da casa dos 97%, encontram-se os modelos de árvore de decisão e a rede neural. Ao comparar os dois, é possível observar que a árvore de decisão foi cerca de 0.45% mais efetiva que a rede neural, que alcançou os 97.06%. Ainda assim, a ANN foi cerca de 2% melhor que o modelo de Naive Bayes, que alcançou somente 95.07% de revocação, ficando consideravelmente atrás dos demais modelos.

Além das métricas acima, também foi possível registrar o tempo de treinamento de cada modelo. A Figura 9 mostra em uma escala de 0 a 3.5 em segundos o tempo de treinamento médio dos modelos.

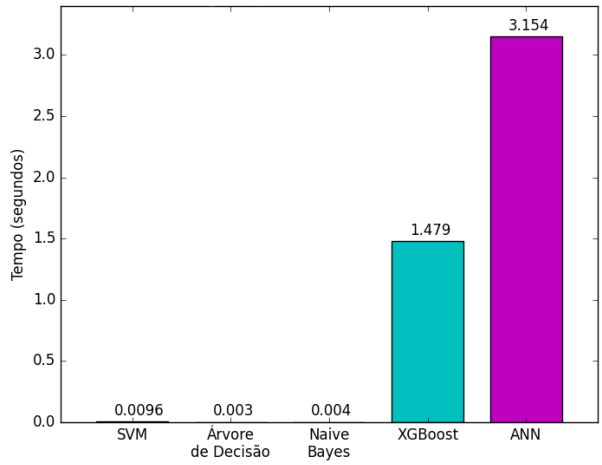

Figura 9: Comparativo do tempo de treinamento dos modelos

Como é possível observar na Figura 9, a árvore de decisão e o algoritmo de Naive Bayes foram os que registraram menor tempo de treinamento, tendo ambos levado apenas 0.003 e 0.004 segundos, respectivamente, para serem treinados. A SVM também teve um tempo de treinamento bem curto e bem próximo dos modelos de árvore de decisão e Naive Bayes, levando apenas 0.096 segundos para ser treinada. Agora, ao olhar para o tempo de treinamento do modelo XGBoost e da ANN, é possível observar que estes demoraram consideravelmente mais quando comparados com os modelos anteriores, ainda que o tempo de ambos tenha sido curto. O XGBoost levou cerca de 1.4 segundos para ser treinado. Já a ANN, levou quase o triplo disso, ficando na faixa dos 3 segundos de treinamento.

Para finalizar esta subseção, a Tabela 6 centraliza todos os dados apresentados nesta seção.

6.2 Overfitting

Durante o processo de ajuste dos modelos e também durante o treinamento foi notado que os modelos apresentaram overfitting. Começando pela árvore de decisão, o resultado médio com os dados de treino foi de 98,11%, enquanto com os dados de validação foi de 97,98%. Já com o modelo bayesiano de Naive Bayes, o resultado médio com os dados de treinamento foi de 98.24%, enquanto com os dados de validação, a acurácia média foi de 97.28%. Com valores próximos a da árvore de decisão, a SVM alcançou um resultado

Tabela 6: Resumo dos valores obtidos por métrica

Métrica	Acurácia	Precisão	Revocação	Tempo de Treino
Árvore de Decisão	97.69%	97.5%	97.69%	0.0030s
SVM	98.39%	98.25%	98.39%	0.0096s
Naive Bayes	95.98%	95.07%	95.98%	0.0040s
XGBoost	99.20%	99.23%	99.20%	1.4790s
ANN	97.48%	97.06%	97.48%	3.1540s

médio de 98.11% e 97.88% no treino e na validação, respectivamente. Em relação ao XGBoost, ele apresentou uma diferença de cerca de 1% entre os resultados. E por fim, tratando dos dados de treino e teste da ANN, este modelo foi oque obteve a menor diferença entre treino e validação, sendo apenas de 0.02%. O detalhamento destes resultados é apresentado na Tabela 7.

Tabela 7: Valores médios de treino e validação

Modelo	Resultado Médio com Dados de Treino	Resultado Médio com Dados de Validação
Árvore de Decisão	98.11%	97.98%
SVM	98.11%	97.88%
Naive Bayes	98.24%	97.28%
XGBoost	99.90%	98.90%
Rede Neural (ANN)	97.40%	97.38%

7 CONCLUSÕES E TRABALHOS FUTUROS

Portanto, levando em consideração o objetivo do trabalho e tendo como base os resultados experimentais apresentados e discutidos neste artigo, pode-se concluir que, de maneira geral, os modelos utilizados tiveram um bom desempenho, ressaltando os modelos XGBoost e SVM que se mostraram os mais adequados para a solução do problema. No entanto, também foi possível observar que o algoritmo de Naive Bayes não obteve um desempenho tão satisfatório em relação aos outros modelos, podendo não ser o modelo mais adequado para a resolução do problema quando comparado com o desempenho dos outros modelos utilizados.

Outro ponto a ser considerado é que modelos baseados em árvore se mostraram bons modelos para esta tarefa, tendo em vista que o modelo XGBoost teve o melhor desemepenho dentre todos e o modelo de árvore de decisão que ainda que não tão assertivo, se mostrou qualificado para a solução do problema de modo geral. Ainda existem, no entanto, alguns pontos que podem ser aprimorados ou alterados na busca de melhorar os resultados obtidos neste trabalho.

No decorrer do projeto, notaram-se algumas possibilidades de continuidades e melhorias, tais como: melhorar o desempenho dos modelos buscando melhorar os resultados apresentados; considerar outros modelos de machine learning para realizar a classificação; realizar classificação multiclasse possibilitada pelo conjunto de dados utilizado, buscando classificar qual o tipo de falha; aplicar no conjunto de dados a técnica da padronização ao invés da normalização; buscar maneiras de remover o overfitting dos modelos.

AGRADECIMENTOS

Este trabalho foi realizado no contexto do grupo de pesquisa *Artificial Intelligence for Industrial Innovation* (AI³) e contou com apoio da Fundação de Amparo à Pesquisa e Inovação do Estado de Santa Catarina (FAPESC), por meio de projeto financiado pelo Edital 27/2021 (TO 2021TR001829).

REFERÊNCIAS

[1] Thyago P Carvalho, Fabrízzio AAMN Soares, Roberto Vita, Roberto da P Francisco, João P Basto, and Symone GS Alcalá. A systematic literature review of machine learning methods applied to predictive maintenance. *Computers & Industrial Engineering*, 137:106024, 2019.

[2] Fernando Amaral. *Introdução à ciência de dados: mineração de dados e big data.* Alta Books Editora, 2016.

[3] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introdução ao datamining: mineração de dados.* Ciência Moderna, 2009.

[4] Simone C Garcia. O uso de árvores de decisão na descoberta de conhecimento na área da saúde. 2003.

[5] Aurélien Géron. *Mãos à Obra: Aprendizado de Máquina com Scikit-Learn & TensorFlow.* Alta Books, 2019.

[6] XGBoost. Xgboost documentation, 2021. URL xgboost.readthedocs.io/en/stable/index.html.

[7] Leonardo Noriega. Multilayer perceptron tutorial. *School of Computing. Staffordshire University*, 2005.

[8] Hassan Ramchoun, Youssef Ghanou, Mohamed Ettaouil, and Mohammed Amine Janati Idrissi. Multilayer perceptron: Architecture optimization and training. 2016.

[9] Anil K Jain, Jianchang Mao, and K Moidin Mohiuddin. Artificial neural networks: A tutorial. *Computer*, 29(3):31–44, 1996.

[10] Felipe da Cunha Carpes. Rastreamento de máxima potência fotovoltaica através de redes neurais artificiais perceptron multicamadas. 2017.

[11] Leandro Fleck, Maria Hermínia Ferreira Tavares, Eduardo Eyng, Andrieli Cristina Helmann, and MA de M Andrade. Redes neurais artificiais: Princípios básicos. *Revista Eletrônica Científica Inovação e Tecnologia*, 1(13):47–57, 2016.

[12] Jéssica H Rodrigues and Marcio Eisencraft. Classificação morfológica de galáxias a partir do seu espectro usando machine learning.

[13] Peshawa Jamal Muhammad Ali, Rezhna Hassan Faraj, Erbil Koya, Peshawa J Muhammad Ali, and Rezhna H Faraj. Data normalization and standardization: a technical report. *Mach Learn Tech Rep*, 1(1):1–6, 2014.

[14] Jiawei Han, Jian Pei, and Hanghang Tong. *Data mining: concepts and techniques.* Morgan kaufmann, 2022.

[15] Xue Ying. An overview of overfitting and its solutions. In *Journal of physics: Conference series*, volume 1168, page 022022. IOP Publishing, 2019.

[16] Pedro Vinícius Meerholz. Repositório do trabalho no github, 2022. URL https://github.com/PedroMeerholz/artigo-cotb-2023.

[17] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

[18] Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020.

[19] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.