UNIVERSIDADE FEDERAL DO CEARÁ

TECHNICAL REPORT

Aluno: Shelda de Souza Ramos

1. Introdução

Este relatório aborda a análise e aplicação de modelos de aprendizado de máquina em um conjunto de dados que contém informações coletadas de sensores em várias máquinas, com o objetivo de prever falhas de máquinas com antecedência. O conjunto de dados inclui uma variedade de leituras de sensores, bem como registros de falhas das máquinas.

O conjunto de dados utilizado nesta avaliação é composto por diversas variáveis que representam as condições operacionais e ambientais das máquinas monitoradas. A seguir, a descrição de cada coluna do conjunto de dados:

- Footfall: Número de pessoas ou objetos passando pela máquina.
- TempMode: Modo de temperatura ou configuração da máquina.
- AQ: Índice de qualidade do ar próximo à máquina.
- USS: Dados do sensor ultrassônico, indicando medições de proximidade.
- CS: Leituras de corrente elétrica utilizadas pela máquina.
- VOC: Nível de compostos orgânicos voláteis detectados perto da máquina.
- RP: Posição rotacional ou RPM (rotações por minuto) das peças da máquina.
- IP: Pressão de entrada na máquina.
- Temperature: Temperatura operacional da máquina.
- Fail: Indicador binário de falha da máquina (1 para falha, 0 para nenhuma falha).

Diante das informações citadas, o objetivo se concentra em aplicar vários modelos de aprendizado de máquina aos dados fornecidos para prever falhas nas máquinas com antecedência e antes disso aplicar GridSearchCV para obter as melhores parametrizações. Para isso, serão utilizados diferentes algoritmos de classificação que foram solicitados nas especificações dessa avaliação, incluindo Decision Tree, Random Forest, AdaBoost, Gradient Boosting, SGB e SVM.

2. Observações

Uma observação muito importante para destacar é que o SGB (Stochastic Gradient Boosting) é originalmente projetado para regressão, portanto foi utilizado o algoritmo **HistGradientBoosting** que tem o mesmo objetivo de melhorar a performance em problemas de classificação, porém utilizando técnicas diferentes.

3. Resultados e discussão

Antes de partir para as questões em si, é importante destacar que a implementação foi organizada em scripts modulares, cada um com funções específicas para realizar uma etapa do processo. A seguir é possível entender bem o fluxo de conexão entre os scripts:

1. **questao1.py**:
    - Etapa de GridSearch: Realiza a busca pelos melhores hiperparâmetros para diferentes modelos e salva os melhores estimadores no dicionário **best_estimator**.
    - Usa o **split** do **dataset_split.py** para dividir os dados e o **meu_pipeline** do **pipeline.py** para padronizar os dados antes de treinar os modelos.
2. **questao2.py:**
    - Etapa de seleção do melhor modelo: Faz uso dos modelos treinados em **questao1.py** (**via best_estimator**) e, com a função **Best_Model**, avalia a acurácia e retorna o modelo de melhor desempenho.
    - Usa o **split** para dividir o dataset novamente e **Best_Model** para comparar os classificadores.
3. **calculo_metricas.py:**
    - Métricas: Responsável por calcular e exibir as métricas de desempenho (como matriz de confusão e classification report) e por selecionar o melhor modelo com base na acurácia.

Após o entendimento de como os scripts se relacionam, é fundamental discutir como foi feita a implementação e os resultados obtidos:

1. Questão 1:
   Na questão 1, o que se pede é que seja realizado um GridSearch para encontrar a melhor parametrização de diferentes modelos de classificação: Árvore de Decisão, Random Forest, AdaBoost, Gradient Boosting, Stochastic Gradient Boosting (SGB) e SVM (Support Vector Machine). Lembrando mais uma vez que o SGB foi substituído por HistGradientBoosting. Para essa tarefa foram utilizadas as bibliotecas pandas e sklearn.
   O GridSearch envolve a criação de uma grade de combinações de parâmetros e a avaliação de cada uma delas com o objetivo de encontrar a combinação que resulta no melhor desempenho do modelo.

   O processo envolve os seguintes passos:

- Definir os hiperparâmetros a serem testados para cada um dos modelos. Exemplo de hiperparâmetros: profundidade máxima, número de estimadores, taxa de aprendizado, etc.
- Configurar e aplicar o GridSearch para cada modelo, onde ele vai testar as combinações de hiperparâmetros em diferentes divisões dos dados de treino.
- Ajustar o modelo de acordo com a melhor combinação encontrada.

- Retornar a melhor parametrização para cada um dos modelos testados.

Ao final, é apresentada a melhor configuração de parâmetros para cada algoritmo. A seguir podemos visualizar os resultados obtidos:

DECISION TREE	RANDOM FOREST	ADABOOST	GRADIENT BOOSTING	HISTGB	SVM
Critério: GINI **Profundidade Máxima:** 5 **Min Samples Split:** 25	**Critério:** GINI **Número De Estimadores:**200 **Profundidade Máxima**: 10 **Min Samples Split:** 20	**Número de Estimadores:** 200 **Taxa De Aprendizado:** 0.1	**Número De Estimadores:** 300 **Profundidade Máxima:** 5 **Taxa De Aprendizado:** 0.01	**Profundidade Máxima:** 5 **Taxa De Aprendizado:** 0.1 **Número De Iterações:** 100	**Número De Iterações:** 100 **C:** 1 **Kernel:** RBF

Tabela 1 - Melhores parâmetros

Breve explicação sobre os parâmetros utilizados no GridSearch:

- max_depth: Profundidade máxima da árvore. Controla o quão "profunda" a árvore pode ser, ou seja, o número máximo de divisões que o modelo pode realizar antes de parar.
- min_samples_split: Número mínimo de amostras necessárias para realizar uma divisão em um nó. Controla o quão grandes os nós precisam ser para serem divididos.
- Criterion (Critério): Função usada para medir a qualidade das divisões (exemplo: gini ou entropy).
- n_estimators (número de estimadores): Número de árvores no conjunto. Um valor maior geralmente melhora a performance, mas aumenta o tempo de treinamento.

- learning_rate: Taxa de aprendizado. Controla o quanto cada estimador contribui para o ajuste final. Valores mais baixos reduzem o impacto de cada estimador, exigindo mais estimadores.
- C: Parâmetro de regularização. Controla o trade-off entre maximizar a margem e minimizar o erro de classificação. Um valor maior dá mais peso aos erros, resultando em margens menores e mais flexíveis, enquanto valores menores aumentam a margem, mas com maior tolerância aos erros.
- Kernel: O tipo de função de kernel usada para transformar os dados antes da classificação. A escolhida foi "rbf", comum em problemas não lineares.

Toda o processo de construção da grade de parâmetros está dentro da própria questão, mas a questão 1 utiliza funções que estão nos scripts dataset_split.py (usado para dividir os dados) e pipeline.py (utilizado para padronizar os dados antes dos outros processos).

2. Questão 2:

Na Questão 2, o objetivo foi criar um script que não só realiza a busca pelos melhores parâmetros dos modelos, como na Questão 1, mas também retorna o melhor modelo entre todos os testados com base no desempenho. Essa implementação exigiu avaliação e escolha de modelos, utilizando a acurácia como principal métrica de comparação.

A Questão 2 faz uso de funções e scripts desenvolvidos anteriormente. A seguir podemos ver a estrutura de como foi feito:

- **Função split():** Importada do script de divisão de dados (dataset_split.py), essa função é responsável por dividir o dataset em treino e teste.
- **Função Best_Model():** Vem de calculo_metricas.py e foi implementada para avaliar e selecionar o melhor modelo com base na acurácia, além de exibir o classification_report e a matriz de confusão de cada modelo.
- **Variável best_estimator**: Importada da Questão 1 para garantir que os melhores estimadores de cada modelo sejam utilizados na avaliação final. Esses estimadores foram obtidos através do GridSearch.

Esta questão pode ser considerada a principal na hora da execução, pois ela retorna os resultados da questão 1 e os resultados da própria questão 2. A seguir serão discutidos os resultados do relatório de classificação de cada um dos modelos e qual foi o melhor modelo escolhido.

- Resultados para Árvore de decisão (Decision Tree):

O Decision Tree apresentou um bom desempenho, com acurácia de 92%. Tanto a precisão quanto o recall são equilibrados entre as classes, o que significa que

o modelo lida bem tanto com a identificação de falsos positivos quanto de falsos negativos, como podemos observar na tabela 2.

RELATÓRIO DE CLASSIFICAÇÃO		
CLASSES	0	1
Precision	0.94	0.90
Recall	0.92	0.92
F1 – Score	0.93	0.91
Acurácia	0.92	

Tabela 2 - Relatório para Decision Tree

Na tabela 3, podemos observar a matriz de confusão: O algorítmo classificou corretamente 402 instâncias da classe 0 e 295 da classe 1, com apenas 34 falsos positivos e 24 falsos negativos.

MATRIZ DE CONFUSÃO
[402 34]
[24 295]

Tabela 3 - Decision Tree

- Resultados para Random Forest:
  O Random Forest melhora ligeiramente em relação ao Decision Tree, com uma acurácia de 93%. Isso era esperado, já que o Random Forest é uma combinação de várias árvores de decisão, o que tende a fornecer resultados mais robustos e estáveis, especialmente em datasets complexos.

RELATÓRIO DE CLASSIFICAÇÃO		
CLASSES	0	1
Precision	0.94	0.92
Recall	0.94	0.92
F1 – Score	0.94	0.92
Acurácia	0.93	

Tabela 4 - Relatório para Random Forest

Podemos observar a partir da matriz de confusão que o algoritmo classificou corretamente 411 instâncias da classe 0 e 294 da classe 1.

MATRIZ DE CONFUSÃO
[411 25]
[25 294]

Tabela 5 - Random Forest

- Resultados para AdaBoost:

O AdaBoost alcançou uma acurácia de 92%, e apresenta um bom equilíbrio entre precisão e recall. AdaBoost é eficaz em melhorar o desempenho de classificadores fracos, mas neste caso o Random Forest superarou o AdaBoost.

RELATÓRIO DE CLASSIFICAÇÃO		
CLASSES	0	1
Precision	0.94	0.89
Recall	0.92	0.92
F1 – Score	0.93	0.90
Acurácia	0.92	

Tabela 6 - Relatório para AdaBoost

Sobre a matriz de confusão podemos inferir que foram classificadas corretamente 400 instâncias da classe 0 e 292 da classe 1.

MATRIZ DE CONFUSÃO
[400 36]
[27 292]

Tabela 7 - AdaBoost

- Resultados para Gradient Boosting:
  O Gradient Boosting se destaca com uma acurácia de 97%, uma melhoria significativa em relação aos modelos anteriores. Ele combina árvores de decisão de forma sequencial, ajustando iterativamente os erros cometidos por árvores anteriores, o que explica sua excelente performance.

RELATÓRIO DE CLASSIFICAÇÃO		
CLASSES	0	1
Precision	0.97	0.98
Recall	0.99	0.96
F1 – Score	0.98	0.97
Acurácia	0.97	

Tabela 8 - Relatório para Gradient Boosting

Em relação a matriz de confusão podemos observar que o Gradient Boosting classificou corretamente 430 instâncias da classe 0 e 305 da classe 1, com apenas 14 falsos negativos e 6 falsos positivos.

MATRIZ DE CONFUSÃO
[430 6]
[14 305]

Tabela 9 - Gradient Boosting

- Resultados para Hist Gradient Boosting:
  O HistGradientBoosting é o modelo com melhor desempenho, alcançando uma acurácia impressionante de 99%. Ele praticamente não comete erros, com 1 falso negativo e 2 falsos positivos. Este resultado mostra que o HistGradientBoosting foi capaz de capturar as nuances do dataset de forma altamente eficaz, sendo o modelo mais confiável para este problema específico, porém, como qualquer modelo de boosting, ele pode sofrer overfitting, que é quando um algoritmo se ajusta muito de perto aos seus dados de treinamento, especialmente em datasets desequilibrados. O fato de ele atingir uma acurácia tão alta sugere que pode estar memorizando o conjunto de treinamento, em vez de aprender padrões generalizáveis.

RELATÓRIO DE CLASSIFICAÇÃO		
CLASSES	0	1
Precision	1	0.99
Recall	1	1
F1 – Score	1	1
Acurácia	0.99	

Tabela 10 - Relatório para HistGB

Sobre a matriz de confusão: classificou corretamente 434 instâncias da classe 0 e 318 da classe 1, com apenas 3 erros no total.

MATRIZ DE CONFUSÃO
[434 2]
[1 318]

Tabela 11 - HistGB

- Resultados para Suport Vector Machine (SVM):
  O SVC apresenta uma acurácia sólida de 94%. Ele se destaca em datasets onde a separação das classes não é linear, utilizando kernels para aumentar sua capacidade de generalização. Neste caso, ele tem um desempenho semelhante ao Random Forest, mas inferior ao Gradient Boosting e ao HistGB.

RELATÓRIO DE CLASSIFICAÇÃO		
CLASSES	0	1
Precision	0.95	0.92
Recall	0.94	0.94
F1 – Score	0.95	0.93
Acurácia	0.94	

Tabela 12 - Relatório para SVM

A matriz de confusão nos mostra que o algoritmo classificou corretamente 411 instâncias da classe 0 e 299 da classe 1.

MATRIZ DE CONFUSÃO
[411 25]
[20 299]

Tabela 13 – SVM

Conclusão Geral

- HistGradientBoosting foi o modelo com o melhor desempenho, quase perfeito na classificação.
- Gradient Boosting também mostrou resultados muito fortes, com uma acurácia de 97%.
- Random Forest e SVC se destacaram com acurácias de 93% e 94%, sendo boas opções com alta capacidade de generalização.
- Decision Tree e AdaBoost foram menos eficazes, mas ainda apresentam boas performances, especialmente para cenários que demandam menor complexidade computacional.

4. Conclusões

Os resultados alcançados nas questões 1 e 2 demonstram uma análise eficiente dos diferentes algoritmos de classificação aplicados ao dataset. Na questão 1, o **GridSearchCV** foi utilizado para identificar as melhores parametrizações para cada modelo, fornecendo insights valiosos sobre como ajustar os hiperparâmetros de forma eficaz. Cada algoritmo — **Decision Tree, Random Forest, AdaBoost, Gradient Boosting, SVC, e HistGradientBoosting** — mostrou desempenho competitivo, com variações notáveis em suas acurácias e capacidades de ajuste.

Na questão 2, foi implementado um método para selecionar o melhor modelo com base na acurácia, considerando os resultados de todos os classificadores. O Gradient Boosting e o HistGradientBoosting destacaram-se com as melhores performances, sendo que o HistGradientBoosting apresentou uma acurácia quase perfeita, sugerindo a possibilidade de overfitting, já que os dados não foram balanceados por conta que o intuito dessa avaliação era basicamente construir funções que retornavam os melhores parâmetros e o melhor modelo, portanto não foi considerado utilizar balanceamento, e o outro motivo é pelo fato de que a distribuição de classes não apresentava um grande desbalanceamento.

Em resumo, os experimentos mostram que, com ajustes corretos de hiperparâmetros, os modelos de Boosting (modelos que são treinados de forma sequencial) tendem a ser mais poderosos para esse tipo de tarefa. Contudo, é necessário um cuidado adicional para evitar overfitting.

5. Próximos passos

Para os próximos passos, seria interessante verificar a performance em um conjunto de **validação** ou em **cross-validation** para garantir que o modelo não está ajustado demais aos dados de treino.