Relatório do EP02 (MAC0425)

João Pedro Barioni Agostini
Nathalia Yukimi Uchiyama Tsuno

30 de Junho de 2025

Resumo

Este projeto busca aprofundar os conhecimentos no tópico de Redes Neurais em Inteligência Artificial, fazendo uso do dataset de exames de COVID-19, disponibilizados e coletados em 2020 pelo Hospital das Clínicas da Universidade de São Paulo (HCUSP).

Primeiramente, foi feito um pré-processamento dos dados coletados, já que raramente dados vêm padronizados e prontos para a aplicação na predição por redes neurais. Dentre os principais processamentos, foram removidos exames de baixa adesão ($< 10\%$), dados foram refatorados para garantir que fossem numéricos e dados binários foram transformados em 0 ou 1 (ou 0.5 no caso de não-aplicação). Assim, podemos garantir resultados coerentes e até mais eficientes do que obteríamos com dados muito ruidosos na questão de formatação. Em muitos casos, sem essa etapa crucial, o treinamento da rede sequer é possível.

Finalmente, foi feita a arquitetura de uma rede neural, fazendo uso de bibliotecas como PyTorch e SciKitLearn. Com o modelo treinado, conseguimos, a partir de dados novos similares aos obtidos no processo anterior, prever se um paciente é portador do coronavírus SARS-CoV-2 com uma alta acurácia, algo muito útil na medicina moderna.

Este relatório contém os resultados obtidos com o pré-processamento e o treinamento do modelo, entrando afundo na discussão do design da arquitetura da rede neural e dos resultados obtidos nos conjuntos de teste.

Conteúdo

1 Introdução

Em meio a uma emergência sanitária global vigente entre os anos de 2020 e 2021, causada pelo vírus da COVID-19, a Inteligência Artificial e as Técnicas de Aprendizado de Máquina também tiveram seu desenvolvimento impulsionado. Ferramentas de AI foram altamente utilizadas, visando mitigar e controlar o contágio dessa doença, enquanto fármacos eram pesquisados para impedir o seu avanço.

Uma máxima muito difundida por pesquisadores e historiadores é o poder de impulso que uma guerra pode acarretar em itens tecnológicos. As melhores armas bélicas e estratégias para derrotar e desastabilizar o inimigo iminente. E, tal como num evento dessas proporções, a pandemia provocada pelo SARS-CoV-2 promoveu adaptações e melhorias nos modelos e ferramentas de aprendizagem de máquina.

Nesse projeto, visamos encontrar um modelo robusto, treinando dados relacionados aos exames de detecção de COVID-19, coletados em 2020 pelo Hospital das Clínicas da USP. Usamos técnicas de ciência de dados, seleção de features, extração e substituição de padrões, aplicação de normalização e PCA, além de análises constantes sobre a distribuição de dados, evitando-se a ocorrência de outliers que poderiam distorcer as funções de probabilidade, projetando um modelo com PyTorch para realizar a detecção da doença a partir da bateria de exames do paciente, seu sexo e idade.

Finalmente, nesse relatório apresentaremos as metodologias utilizadas, resultados obtidos com gráficos das funções perda, matriz de confusão e outras métricas e, ao fim, as conclusões obtidas com esses experimentos.

Adicionalmente, gostaríamos de esclarecer que o pré-processamento e a arquitetura foram feitos com Júpiter Notebook e, portanto, estão na extensão `.ipynb` nos diretórios respectivos. Ademais, o conjunto de dados pré-processados `.csv`, para uma melhor acurácia, deverá passar pela splitação, pela normalização e busca do PCA no conjunto de dados na primeira parte da construção das redes neurais.

2 Metodologia

2.1 Pré-Processamento de Dados

O dataset é composto por dois conjuntos de dados, relacionados pela chave ID_PACIENTE.

O conjunto de dados `HC_EXAMES_1.csv` indica cada exame realizado por paciente (indicado por seu ID), com uma subdivisão indicada pelo analito correspondente.

Já o conjunto de dados `HC_PACIENTES_1.csv` indica os dados relacionados ao paciente identificado por um ID, como sexo e ano de nascimento.

2.1.1 União e Processamento de Tabelas

A primeira etapa do pré-processamento realizado foi a criação de atributos derivados no dataset de exames: o merge entre `DE_EXAME` e `DE_ANALITO`, para indicar um exame com uma finalidade diferente. Essa novo atributo foi subdividido em mais três outros atributos derivados, relativos ao resultado obtido, aos valores de referência e à unidade de medida.

Além disso, como solicitado, foram mergeadas as tabelas correspondentes aos datasets, tal que cada registro corresponda a um paciente identificado por seu ID. Os demais atributos correspondem aos exames realizados, tal que para cada combinação `EXAME` e `ANALITO`, tem-se:

1. `EXAME - ANALITO - RESULTADO`

2. `EXAME - ANALITO - UNIDADE DE MEDIDA`

3. `EXAME - ANALITO - VALOR DE REFERÊNCIA`

E preservando o `SEXO` e `ANO DE NASCIMENTO` de cada ID de atendimento.

2.1.2 Processamento e Limpeza de Dados

A segunda etapa do pré-processamento correspondeu à limpeza de atributos pouco relevantes para a avaliação final.

A priori, uma grande máxima da ciência de dados é a redução de dimensionalidade de dados e a mitigação de outliers. Assim, muitos atributos continham poucas ocorrências (isto é, apenas um ou outro paciente realizou determinado exame). Uma solução plausível e recomendada foi estipular um limiar (nesse caso, de 10%), mantendo colunas que possuíssem, ao menos, este valor mínimo de dados válidos.

A posteriori, essencialmente, como produto de uma situação real e que ocorreu em meio a uma grande urgência global, os dados eram todos misturados e escritos das formas mais diversas possíveis.

Muitas colunas de `RESULTADOS`, essencialmente numéricos, continham registros isolados que mesclavam elementos numéricos com outros caracteres diversos (como > 0.9 ou ?0.4). Então, um passo crucial foi analisar os padrões recorrentes dessa ocorrência de mistura. Certamente, isso se mostrou como um empencilho de um processamento rápido utilizando as ferramentas programadas e disponibilizadas pelo framework `PANDAS`.

Com uso do `RegEx`, buscou-se tratar esses padrões (uso de caracteres como $>$, $<$, $=$, ?, -, *), eliminando-os, mas, mantendo os valores numéricos. A busca foi rápida e facilitada e permitiu transformar colunas numéricas inteiras.

Outro ponto recorrente nesse pré-processamento foi o tratamento de colunas de natureza categórica. Muitas delas eram binárias. Então, bastou-se aplicar um *encoding*, transformando-as em numéricas binárias $\{0, 1\}$, para adaptar à entrada do modelo. Por outro lado, haviam atributos categóricas com mais de duas classes. Nesse caso, como não eram relevantes para a classificação, foram eliminadas (verifique que um $label - encoding$ seria uma aplicação estranha, já que não havia uma hierarquia estabelecida entre os dados e um $onehot - encoding$ poderia estender a dimensionalidade dos dados, para valores que não fossem tão interessantes).

Assim, também com ajuda do `RegEx`, alguns atributos foram binarizados para $\{0, 0.5, 1\}$, fazendo um mapeamento de forma como {`True, Positivo, Detectado, Reativo, Presente, Sim` $\rightarrow 1$;
`False, Negativo, Não Detectado, Não Reativo, Ausente, Não` $\rightarrow 0$}. E, conforme especificado, para valores como `Inconclusivo`, o mapeamento foi dado para 0.5.

Ademais, para tratar *missing values*, foram substituídos os NaN por 0, para colunas não-binárias, conforme especificado. Uma outra estratégia seria o uso do `KNN-Inputer` ou Árvores de Decisão, para tentar manter a distribuição de probabilidade dos dados presentes e reais.

Como os valores de resultados são o foco do pré-processamento, os atributos relacionados à `UNIDADE` ou aos `VALORES DE REFERÊNCIA` foram eliminados do $dataframe$, para reduzir a dimensionalidade e não atrapalhar o modelo a generalizar os dados.

Com os dados referentes ao `SEXO`, foram binarizados para $\{0, 1\}$ e ao `ANO DE NASCIMENTO`, para *missing values*, foi-se estipulado o ano de `1970`.

Finalmente, para efeitos de treinamento do modelo e classificação, foram mantidos atributos relacionados à detecção de COVID-19 como target. Em especial, ao exame `COVID-19 IgG`. Como sugerido e instruído, primeiramente, são usados os resultados desse exame. Caso não haja um exame relacionado à COVID-19

que o registro realizou, então, ele é tirado do $dataframe$. Caso contrário, o target é dado como 0.

2.1.3 Preparação e Melhorias Para Adaptação no Modelo

A terceira etapa do pré-processamento de dados foi buscar estratégias que visassem a melhoria da generalização de dados pelo modelo a ser aprendido.

Foi aplicado a normalização de dados para tentar mitigar a distorção de dados no domínio do problema (para os dados de treinamento e o vetor de classificação).

Para reduzir a dimensionalidade do plano, foi aplicado o algoritmo de PCA, que selecionou um conjunto de vetores que retivesse 95% da variância dos dados. O resultado foi um novo conjunto de dados com 9 componentes principais. Isso significa que, nesse conjunto mais limitado, estão contidos 95% da informação distintiva presente original.

2.2 Arquitetura da Rede Neural

A Arquitetura da Rede Neural, construída com o PyTorch.

Foi constituído por duas *hidden layers*, cada uma contendo 256 perceptrons (poucas camadas, mas, densas). A função de ativação foi a $LeakyReLU$, que melhor funciona no Backpropagation, em razão da sua derivada.

$$\text{LeakyReLU(x)} = \begin{cases} \alpha x, & \text{se } x \geq 0 \\ x, & \text{Caso contrário} \end{cases}$$

Onde $\alpha = 0.1$.

A camada de saída possui a função `Logit` como função de ativação, já que sua saída está entre 0 e 1. Simulando uma probabilidade.

A otimização é dado pelo algoritmo do `Gradiente Descendente Estocástico`, com taxa de aprendizagem $\alpha = 0.001$ (não tão grande para convergir rapidamente, nem tão pequeno para agilizar o processo da função de perda). E a função a ser otimizada é a `binary cross-entropy loss` com função sigmoide:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}[y_i log(\hat{y}_i) + (1 - y_i)(1 - log(\hat{y}_i))].$$

Durante a fase de treinamento, cada dado sofre o processo de $foward$, sendo multiplicado pelos pesos atuais mais os viéses, adicionados a uma função não linear/ de ativação (no nosso caso, a $LeakyReLU$). Isso passa por entre as camadas ocultas, sofrendo mais transformações lineares e sai no output, que possui uma função `logit` como ativadora. Aqui, a `Cross-Entropy` mede os erros entre os valores preditos e reais e, com a função de otimização `SGD`, o processo de *backpropagation* vai atualizando os pesos e viéses das camadas de saída e ocultas.

A entrada é adaptável à quantidade de atributos a serem considerados do conjunto de dados a ser inputado. Por exemplo, realizando todo o pré-processamento, obtivemos 10 colunas (que eram as componentes principais). Já a saída corresponde à distribuição de probabilidade do predito (nesse caso, binário). E nossos batches tem tamanho de 32 amostras.

2.2.1 Pequeno Manual de Uso da Arquitetura da Rede Neural

A rede é um objeto da classe `CovidPredictor`. E também precisamos de um objeto que faça a adaptação do dataframe para o modelo em PyTorch, `CovidDataset`, que relaciona cada entrada x para um target y.

Então, é necessário instanciar um objeto do `CovidDataset` para o conjunto de treinamento e para o de validação. E, após instanciar a rede, como o input corresponde a um registro por vez, é necessário inputar os dados de `CovidDataset` com auxílio de um laço iterativo pelo valores do objeto.

A saída do modelo corresponde às distribuições preditas que precisam ser avaliadas sobre os valores reais pela `loss` e realizar o *backpropagation* para atualizar os pesos.

2.3 Descrição de Resultados

Finalmente, para os experimentos, foram realizadas mais outras técnicas de ciência de dados e aprendizagem de máquina.

Vale também notar que, em razão de haver uma proporção maior de casos negativos, houve um peso para tentar balancear as classes na classificação. Nesse caso, os parâmetros utilizados foram 87.5% para negativos e 12.5%, para positivos.

Os dados foram repartidos em `Treinamento`, `Validação` e `Teste`. O uso do `train_test_split` da biblioteca do `Sktlearn` foi importante para splitar os dados de teste na proporção 20% e 80%. Mantivemos uma semente associada para garantir reprodutibilidade dos experimentos.

O treinamento do modelo foi realizado, utilizando-se $k - folds$, para $k = 5$. Para cada k, o treino foi feito em *batches* de 32 amostras, para um treinamento com menos memória e mais rápido.

Note que, como $k = 5$, a função `kf.split` realiza a validação cruzada de forma estratificada, em que 80% dos dados de treinamento foram utilizados para treinamento e 20%, para validação no modelo, automaticamente. Obedecendo, assim, o especificado no enunciado.

O número de épocas foi 100, um número razoavelmente grande, e adequado para a magnitude do conjunto resultante do pré-processamento realizado.

Para cada iteração do $k - folds$, o modelo é treinado com os dados de treinamento e validado posteriormente.

Finalmente, as métricas utilizadas foram, sobretudo, a `Acurácia`. Contudo, também foram apresentados a `Revocação`, a `Precisão` e a `F1`.

3 Resultados

Rodando os experimentos, foram obtidos esses resultados:

3.1 Fold 1

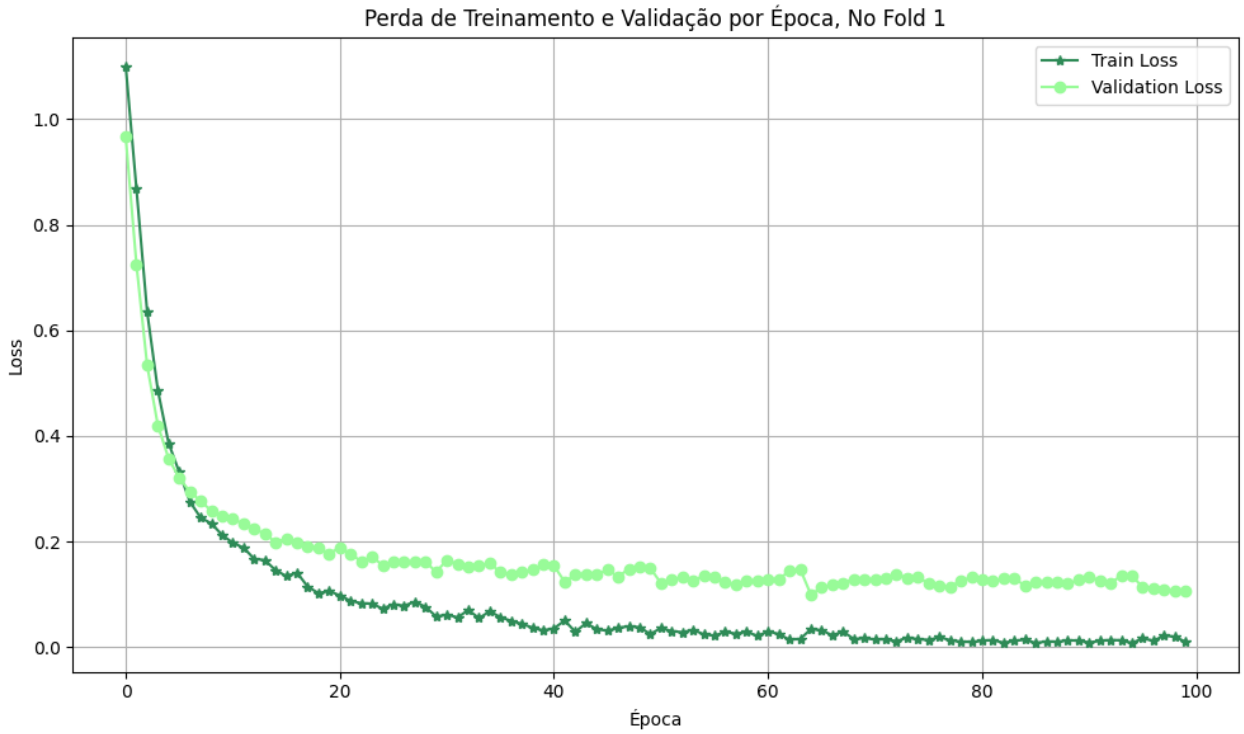

Figura 1: Evolução da Função de Perda

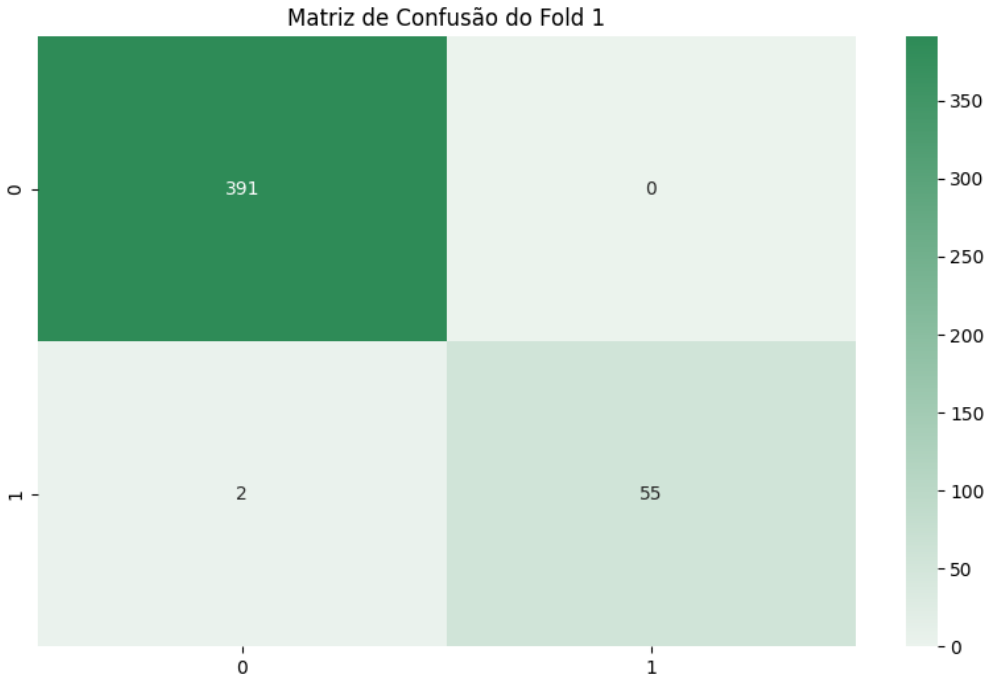

Figura 2: Matriz de Confusão

Métricas

1. Acurácia : 0.9353
2. Revocação : 0.9649
3. Precisão : 1.0000
4. F1-Score : 0.9821

3.2 Fold 2

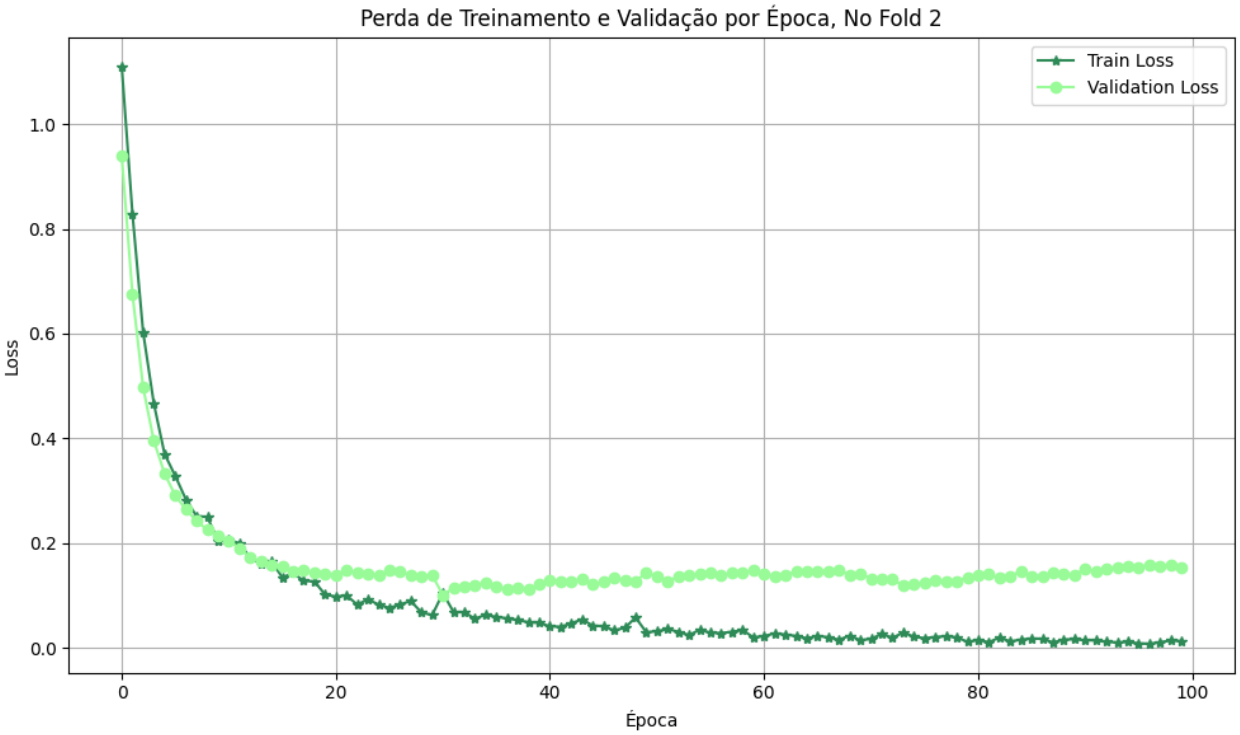

Figura 3: Evolução da Função de Perda

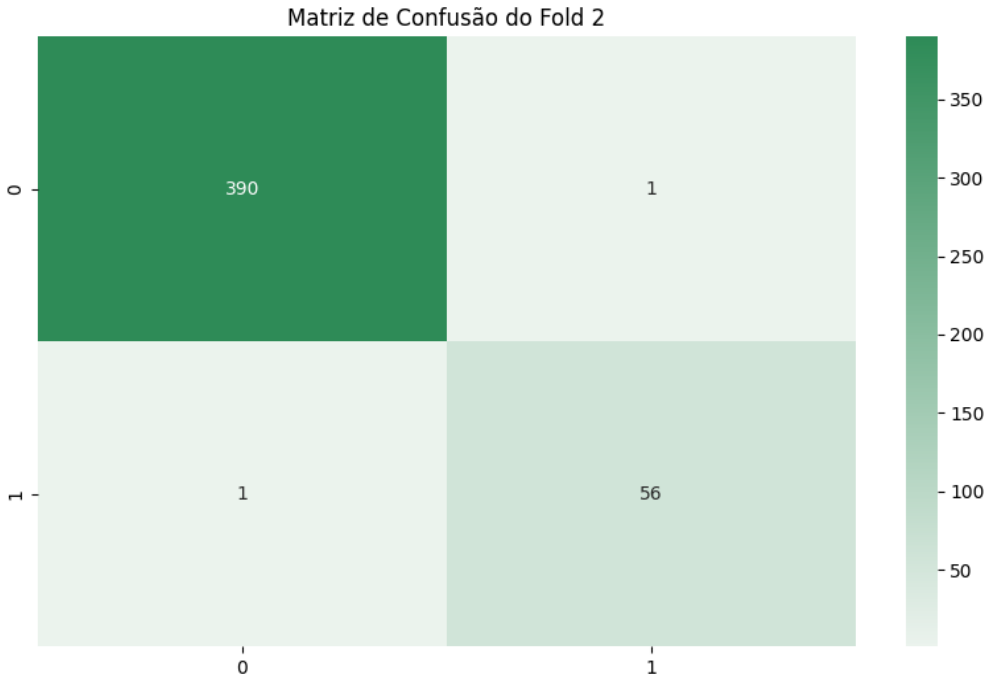

Figura 4: Matriz de Confusão

`Métricas`

1. `Acurácia` : 0.9955
2. `Revocação` : 0.9825
3. `Precisão` : 0.9825
4. `F1-Score` : 0.9825

3.3 Fold 3

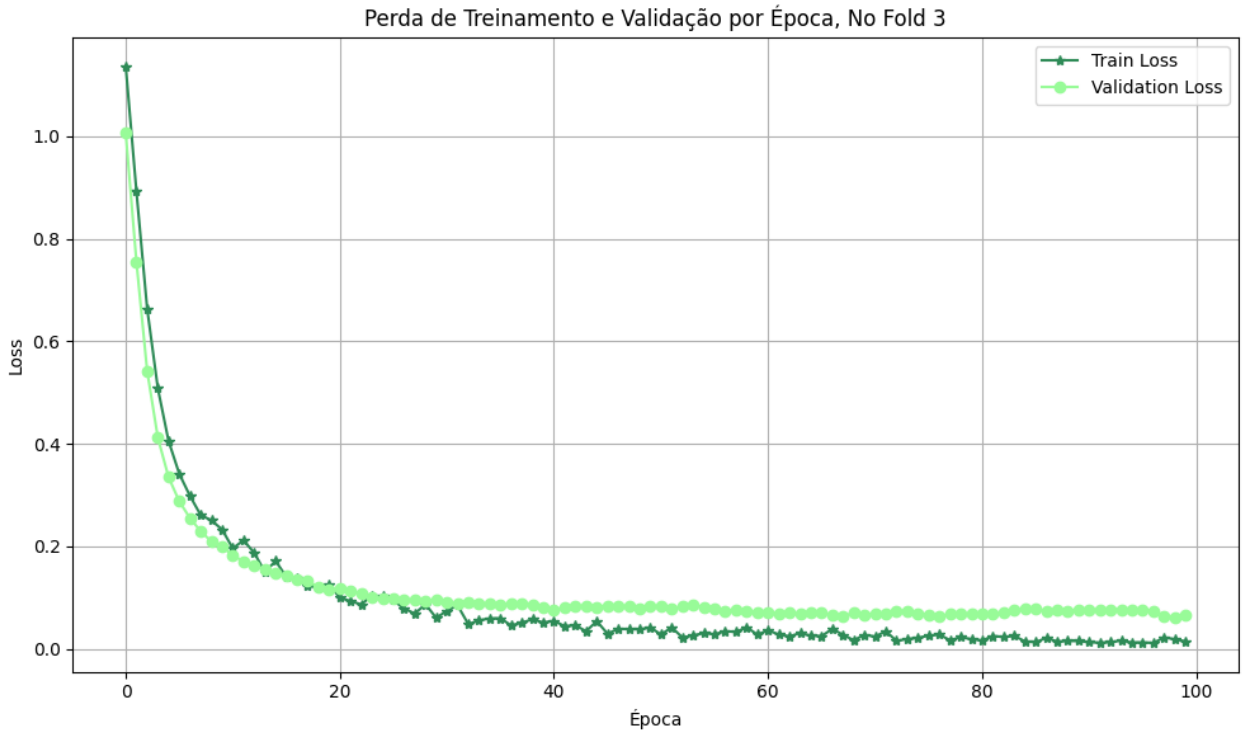

Figura 5: Evolução da Função de Perda

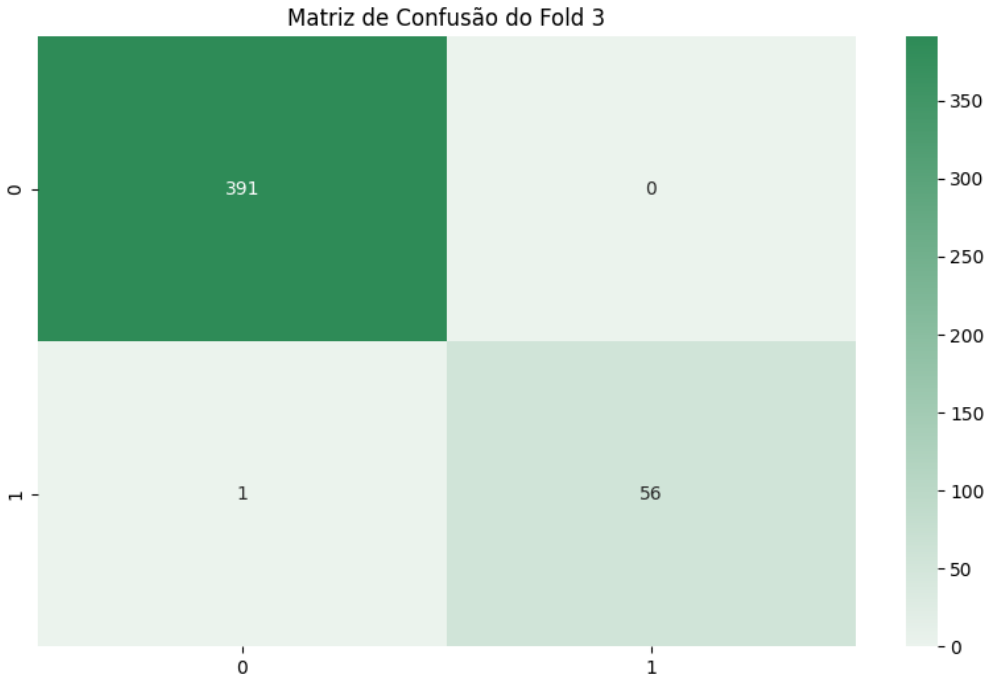

Figura 6: Matriz de Confusão

Métricas

1. Acurácia : 0.9978
2. Revocação : 0.9825
3. Precisão : 1.0000
4. F1-Score : 0.9912

3.4 Fold 4

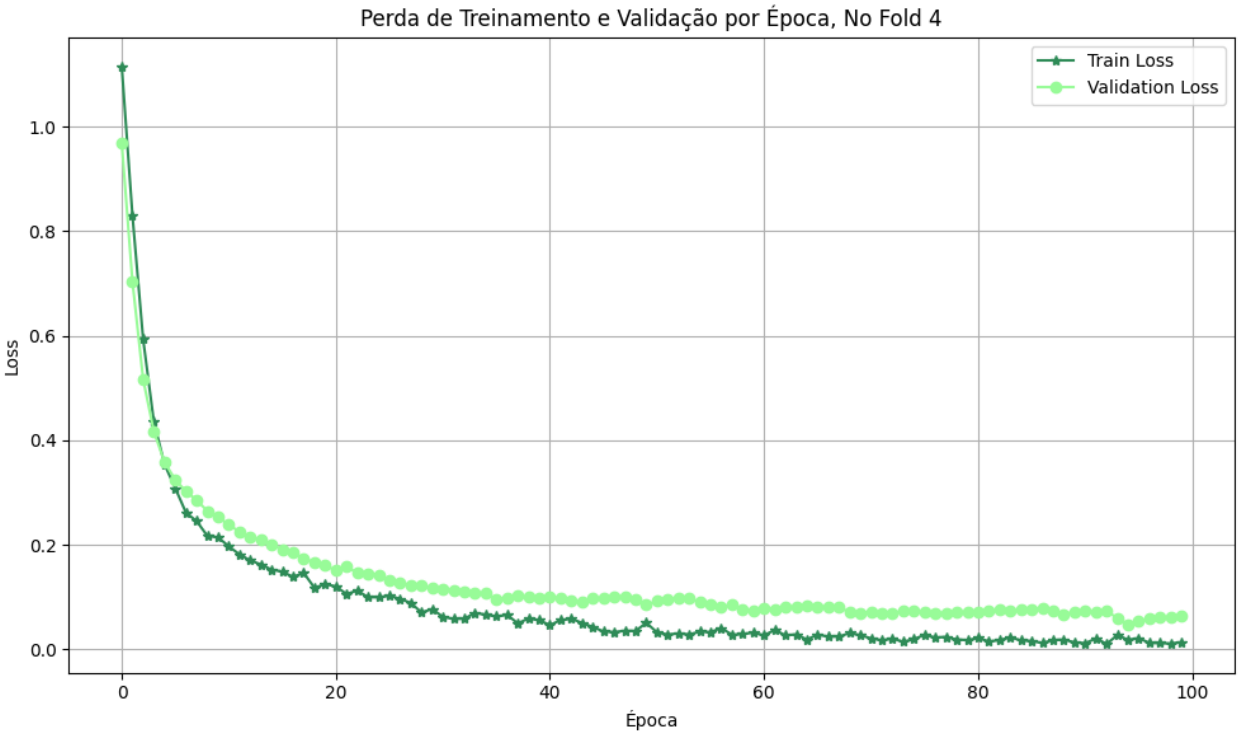

Figura 7: Evolução da Função de Perda

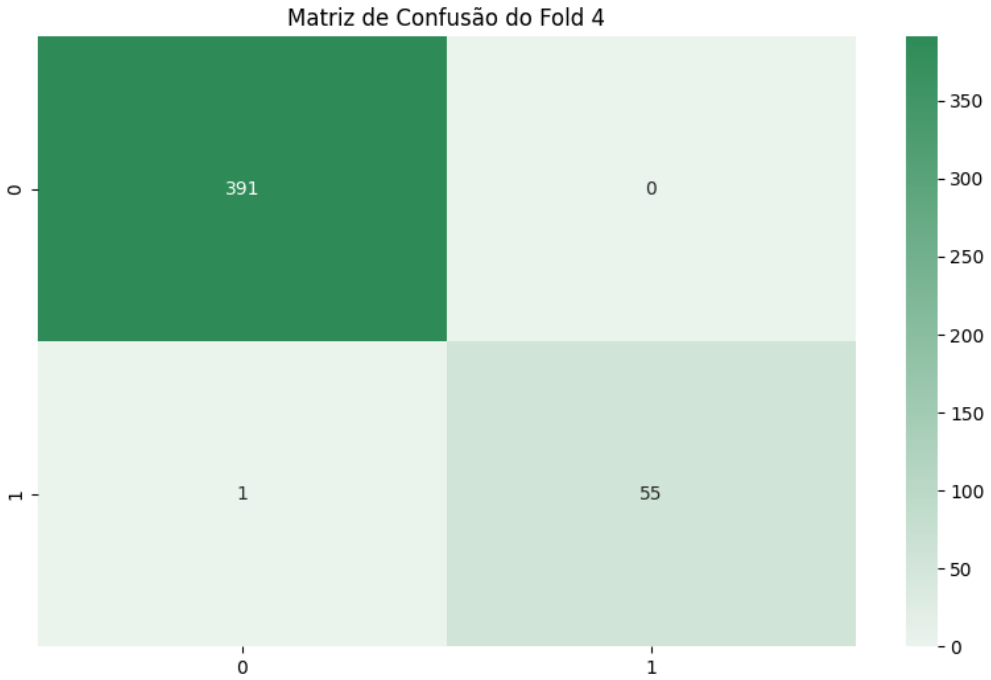

Figura 8: Matriz de Confusão

`Métricas`

1. `Acurácia` : 0.9978
2. `Revocação` : 0.9821
3. `Precisão` : 1.0000
4. `F1-Score` : 0.9910

3.5 Fold 5

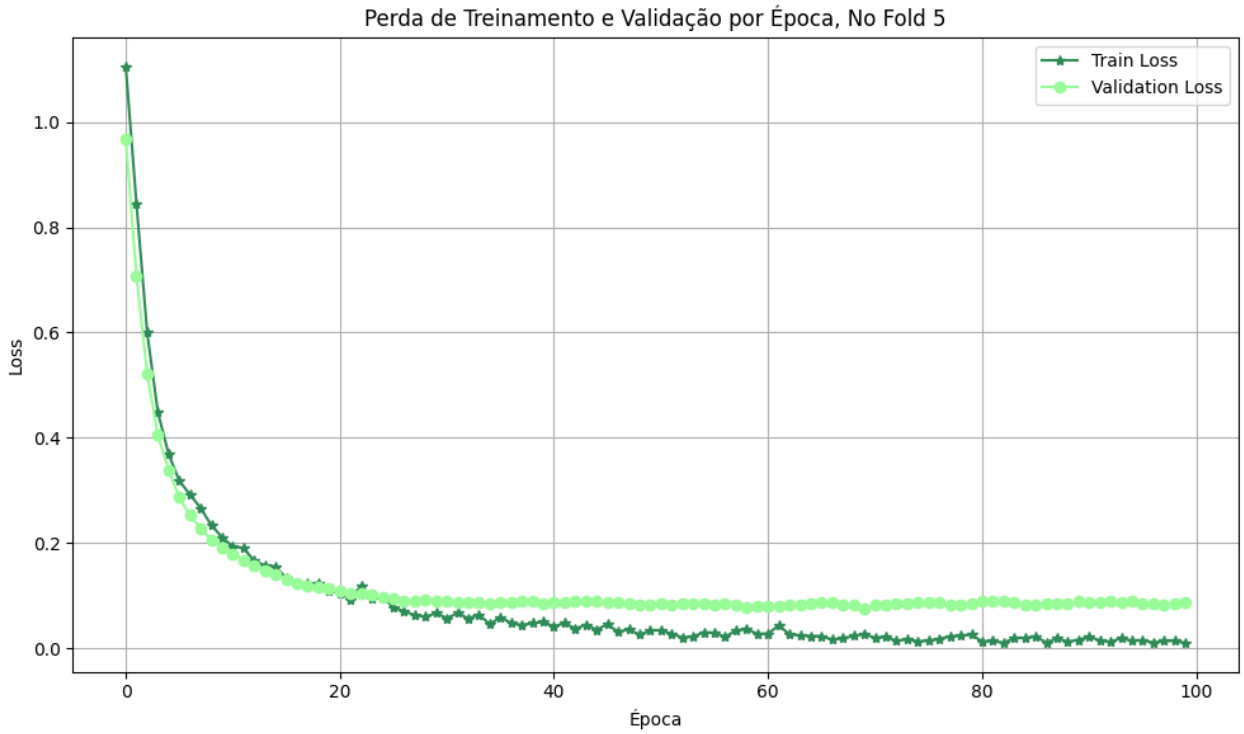

Figura 9: Evolução da Função de Perda

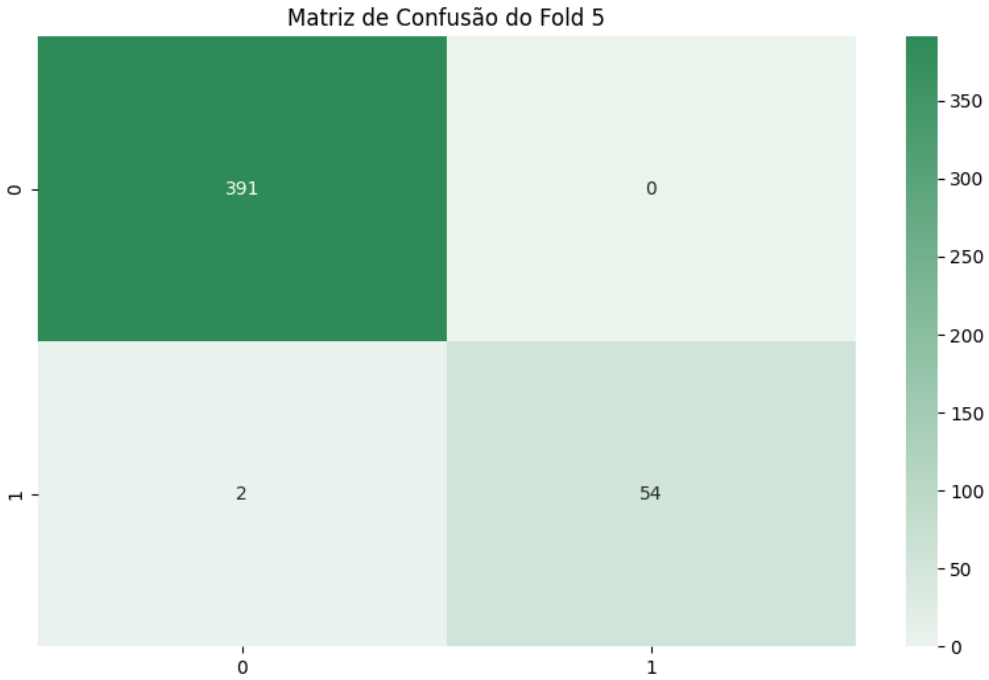

Figura 10: Matriz de Confusão

Métricas

1. Acurácia : 0.9955
2. Revocação : 0.9643
3. Precisão : 1.0000
4. F1-Score : 0.9818

3.6 Médias, Desvios-Padrão, Melhor e Pior Valor de Cada Métrica

Acurácia

1. Média : 0.9964
2. Desvio-Padrão : 0.001
3. Melhor Valor : 0.9978
4. Pior Valor : 0.9955

Revocação

1. Média : 0.9753
2. Desvio-Padrão : 0.0087
3. Melhor Valor : 0.9825
4. Pior Valor : 0.9643

Precisão

1. Média : 0.9965
2. Desvio-Padrão : 0.007
3. Melhor Valor : 1.0
4. Pior Valor : 0.9825

Medida-F

1. Média : 0.9857
2. Desvio-Padrão : 0.0043
3. Melhor Valor : 0.9912
4. Pior Valor : 0.9818

3.7 Conjunto de Testes

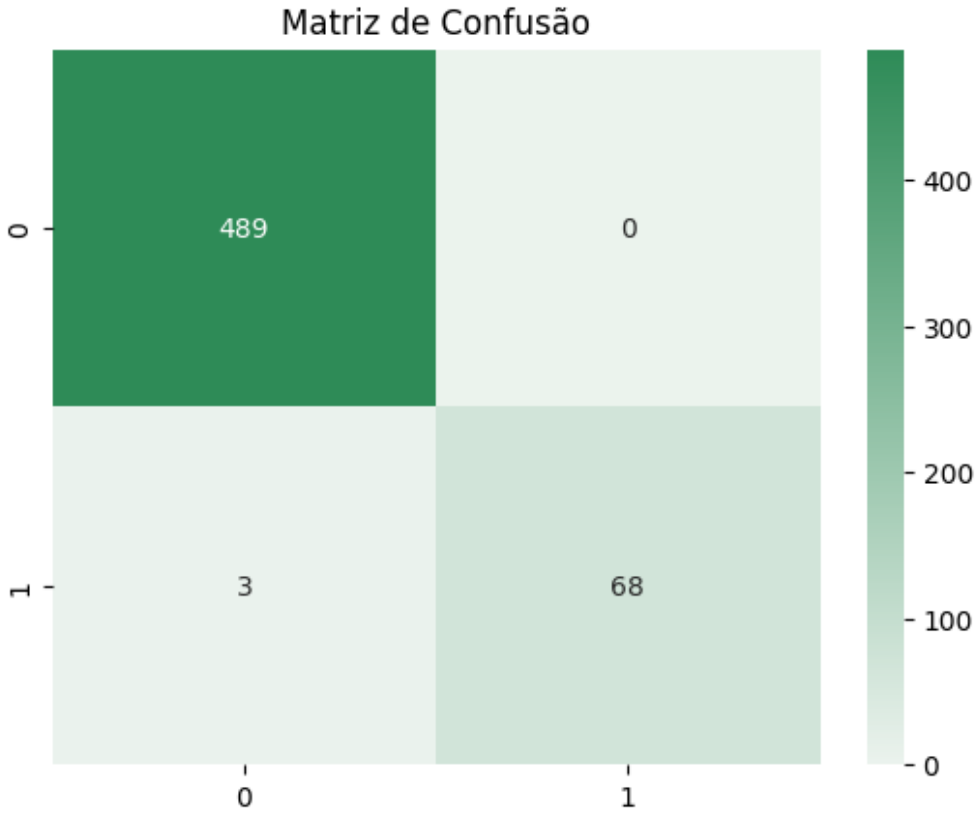

Figura 11: Matriz de Confusão

`Métricas`

1. `Acurácia` : 0.9980
2. `Revocação` : 0.9577
3. `Precisão` : 1.0000
4. `F1-Score` : 0.9818

4 Discussão

Finalmente, analisando os resultados finais, foi possível obter os seguintes resultados: a acurácia teve baixa variância e ficou centrada no valor de 99.64%. Isso significa que o modelo tem uma taxa de acerto muito alta e boa. Isso se comprova quando as métricas do conjunto de testes são observadas. O modelo tem uma classificação média boa.

A revocação ou cobertura também foi muito boa: baixíssima variância e ficou centrada no valor de 97.52%. Esse valor demonstra que o modelo classifica corretamente quando uma pessoa está, de fato, saudável. Portanto, uma pessoa doente não é diagnosticada, erroneamente, por saudável e não recebe o atendimento correto adequado. Isso é comprovado quando verificadas as métricas do conjunto de testes.

A precisão, igualmente, teve resultados muito surpreendentes. A métrica teve baixa variância, entornando 99.64%. Isso demonstra que nosso modelo classifica corretamente quando uma pessoa está, de fato, doente. Portanto, uma pessoa saudável não é dignosticada, erronamente, por doente e não ocupa o lugar de alguém que realmente precisa. Isso é comprovado quando verificadas as métricas do conjunto de testes.

Consequentemente, isso valorizou a $F1 - Score$, já que se trata de uma ponderação da revocação e da precisão. De todas as formas, essa métrica teve um desempenho médio e sem muita variância no valor de 99.57%.

Adicionalmente, podemos ver a evolução da função de perda agindo ao longo das épocas. Em média, a *loss* se manteve em 0.3, indicando que ela convergiu bem e para um valor baixo. Ou seja, o modelo está aprendendo bem e fazendo previsões próximas ao correto.

Como mantivemos um conjunto de controle de testes separados e o valor foi próximo para as métricas do conjunto de validação, pudemos perceber que o modelo não possui $overfitting$, nem $underfitting$. Portanto, generaliza bem para os demais casos.

5 Bibliografia

JAMES, Gareth; WITTEN, Daniela; HASTIE, Trevor; TIBSHIRANI, Robert. *An Introduction to Statistical Learning: with Applications in Python*. New York: Springer, 2021.

BARBER, David. *Bayesian Reasoning and Machine Learning*. Cambridge: Cambridge University Press, 2012.

JOHNSON, Richard A.; WICHERN, Dean W. *Applied Multivariate Statistical Analysis*. 6th ed. Upper Saddle River: Pearson Prentice Hall, 2007.

MURPHY, Kevin P. *Machine Learning: A Probabilistic Perspective*. Cambridge: MIT Press, 2012.

RASCHKA, Sebastian; LIU, Yuxi; MIRJALILI, Vahid. *Machine Learning with PyTorch and Scikit-Learn*. Birmingham: Packt, 2022.

RUSSELL, Stuart; NORVIG, Peter. *Artificial Intelligence: A Modern Approach*. 4th ed. Hoboken: Pearson, 2021.