

# Relatório Técnico: Resgate de Vítimas em Catástrofes

## 1. Introdução

### 1.1 Contextualização

O resgate de vítimas em situações de catástrofes naturais, desastres e grandes acidentes é uma tarefa essencial que requer rapidez e precisão. A adoção de técnicas de Inteligência Artificial (IA) pode aprimorar a eficácia desses resgates, priorizando vítimas em estado crítico e otimizando o tempo de resposta das equipes de socorro. Este relatório apresenta o desenvolvimento e avaliação de um sistema baseado em IA, que utiliza uma Rede Neural MLP (Multi-Layer Perceptron) e uma Árvore de Decisão como classificadores para auxiliar na priorização do salvamento de vítimas.

### 1.2 Objetivos do Relatório

O objetivo deste relatório é detalhar o desenvolvimento de um sistema de priorização de resgate de vítimas em cenários de catástrofes, utilizando classificadores para determinar a gravidade das vítimas. Serão apresentados os métodos, implementação e resultados obtidos com o uso de uma Rede Neural MLP e de uma Árvore de Decisão, ambos configurados para classificar as vítimas em diferentes níveis de gravidade.

## 2. Rede Neural MLP (Multi-Layer Perceptron)

### 2.1 Fundamentação

O Perceptron Multicamadas (MLP) é uma das arquiteturas de redes neurais artificiais mais conhecidas e utilizadas para a tarefa de classificação. Consiste em múltiplas camadas de neurônios, onde cada camada é completamente conectada à camada anterior e à próxima, formando uma rede *feedforward*.

**Funcionamento:** O MLP é composto por:

- **Camada de Entrada:** Recebe o conjunto de características (features) como entrada.
- **Camadas Ocultas:** Processam as entradas aplicando uma função de ativação em cada neurônio, que introduz não linearidades na rede, permitindo que ela capture relações complexas nos dados.
- **Camada de Saída:** Produz a saída final, que pode ser uma classe em um problema de classificação.

**Funções de Ativação:** As funções de ativação são responsáveis por introduzir não linearidade na rede. As mais comuns incluem:

- **ReLU (Rectified Linear Unit):** Retorna zero para valores negativos e o próprio valor para valores positivos, acelerando a convergência do treinamento.
- **Sigmoid (Logistic) :** Função logística que mapeia valores de entrada para o intervalo  $[0, 1]$ , sendo útil para problemas de classificação binária.
- **Tanh:** Mapeia valores de entrada para o intervalo  $[-1, 1]$ , geralmente preferida quando se deseja centragem dos dados.

**Solvers (Otimizadores):**

- **Adam (Adaptive Moment Estimation):** Ele ajusta a taxa de aprendizado para cada parâmetro individual, utilizando médias móveis dos gradientes e seus quadrados. O Adam é eficiente em termos de memória e geralmente funciona bem em problemas de larga escala e com grandes conjuntos de dados.

**Vantagens e Desvantagens do MLP:**

- **Vantagens:** Capaz de modelar relações não lineares complexas, flexível e aplicável a uma ampla gama de problemas.
- **Desvantagens:** Requer grande quantidade de dados para treinamento eficaz, mais difícil de interpretar comparado a métodos como árvores de decisão, e pode levar a *overfitting* sem técnicas adequadas de regularização.

**Validação Cruzada**

A validação cruzada é uma técnica amplamente utilizada para avaliar a performance de modelos de aprendizado de máquina. No método k-fold, o conjunto de dados é dividido em k subconjuntos (folds). O modelo é treinado em k-1 desses subconjuntos e testado no subconjunto restante. Este processo é repetido k vezes, e a média dos resultados é calculada. Essa técnica ajuda a garantir que o modelo generalize bem para dados não vistos.

**5-Folds:** Neste trabalho, utilizou-se a validação cruzada com 5 folds, o que significa que o conjunto de dados foi dividido em 5 partes. Cada parte foi usada como conjunto de teste uma vez, enquanto as outras quatro partes foram usadas para treinar o modelo. A performance final é a média das 5 iterações, proporcionando uma estimativa robusta da generalização do modelo.

## 2.2 Metodologia

### 2.2.1 Modelagem

A modelagem da Rede Neural MLP (Multi-Layer Perceptron) foi desenvolvida conforme os critérios de avaliação estabelecidos, garantindo uma abordagem robusta e eficaz para a classificação das vítimas em diferentes níveis de gravidade. As etapas principais da modelagem incluem a seleção das características, o pré-processamento dos dados e a definição da saída do modelo.

**Características:** As características utilizadas para treinar a Rede Neural MLP foram selecionadas com base na relevância clínica para a avaliação da gravidade das vítimas. As variáveis consideradas incluem:

- **qPA:** qualidade da pressão arterial; resulta da avaliação da relação entre a pressão sistólica e a diastólica;
- **pulso:** pulsação ou Batimento por Minuto (pulso)
- **frequência respiratória:** frequência da respiração por minuto
- **classes de gravidade:** são 4 classes que apresentam o estado de saúde do acidentado. É o que deve ser predito pelo classificador: 1 = crítico, 2 = instável, 3 = potencialmente estável e 4 = estável.

Essas características fornecem uma visão abrangente do estado de saúde das vítimas, permitindo que o modelo capture padrões críticos para a classificação adequada.

**Pré-processamento:** Não houve necessidade de realizar nenhum tipo de pré-processamento, já que os dados testados estavam completos. Não foi implementado nenhum tipo de arbitragem de dados para características incompletas.

**Saída:** A saída da Rede Neural MLP foi configurada para classificar as vítimas em quatro categorias de gravidade:

- Crítico
- Grave
- Moderado
- Leve

### 2.2.2 Parametrização

**Topologia e Configuração das Camadas:** Foram testadas diversas topologias de camadas ocultas para determinar a configuração que melhor se adapta aos dados. As topologias avaliadas incluíram:

- Uma camada oculta com 50, 10, e 20 neurônios
- Duas camadas ocultas com (50, 10), (20,10), (10,10), (30,10), (10,7), (10, 10), (20, 10), (20,5), (15,5) neurônios, respectivamente
- Três camadas ocultas com (50, 10, 5), (20,10,5), (15,10,5) neurônios.
- Quatro camadas ocultas com (50, 10, 5 e 5) neurônios
- Cinco camadas ocultas com (50, 10, 5, 5, 5) neurônios.

- Seis camadas ocultas com (50, 10, 5, 5, 5, 5) neurônios.

#### **Função de Ativação dos Neurônios:**

- ReLU (Rectified Linear Unit)
- Tanh
- Sigmoid (Logistic)

#### **Taxa de Aprendizado e Momento:**

- **Taxa de Aprendizado:** Inicialmente configurada em 0.001, ajustada conforme necessário durante o treinamento para garantir uma convergência estável.
- **Momento:** Dentro da biblioteca utilizada, há dois momentos utilizados no Adam, sendo “*beta\_1*” e “*beta\_2*” tendo os valores de 0.9 e 0.99, respectivamente.

**Solver Utilizado:** Foi utilizado o *solver* Adam (*Adaptive Moment Estimation*) devido às suas vantagens em termos de eficiência e adaptabilidade. Essa escolha se justifica pela sua capacidade de lidar eficazmente com grandes conjuntos de dados e por proporcionar uma convergência mais rápida e estável em comparação com outros *solvers*.

### **2.2.3 Treino e validação**

**Descrição do Dataset:** O dataset utilizado para o treinamento e validação da Rede Neural MLP compreende 4000 registros de vítimas, cada um contendo informações detalhadas sobre sinais vitais e a classe de gravidade associada. São 8 dados apresentados no dataset:

- **Id:** identificação da vítima [0, n]
- **pSist:** pressão diastólica
- **pDiast:** pressão diastólica
- **qPA:** qualidade da pressão arterial; resulta da avaliação da relação entre a pressão sistólica e a diastólica;
- **pulso:** pulsação ou Batimento por Minuto (pulso)
- **frequência respiratória:** frequência da respiração por minuto
- **gravidade:** valor a ser estimado pela RN em função dos sinais (regressor).
- **classes de gravidade:** são 4 classes que apresentam o estado de saúde do acidentado. É o que deve ser predito pelo classificador: 1 = crítico, 2 = instável, 3 = potencialmente estável e 4 = estável.(classificador)

**Procedimento de Treino e Validação:** O processo de treino e validação foi conduzido da seguinte maneira:

1. **Validação Cruzada:** Foi utilizada a validação cruzada com 5 folds para avaliar a performance do modelo durante o treinamento.
2. **Grid Search:** Uma busca em grade (Grid Search) foi implementada para explorar diferentes combinações de hiperparâmetros, identificando a configuração que maximiza a precisão e a capacidade de generalização do modelo.

### Critérios de Parada:

- **Tolerância:** Quando não houver melhorias de no mínimo 0.001 no score, durante 10 épocas, o treinamento para.
- **Número Máximo de Épocas:** Definido em 2000 épocas para garantir que o modelo tivesse oportunidades suficientes para convergir, evitando ao mesmo tempo um tempo de treinamento excessivo.

**Lotes e Tamanho:** O treinamento foi realizado utilizando o método de mini-batch com um tamanho de lote (*batch size*) de 40 amostras.

**Escolha do Melhor Modelo:** Através do *GridSearchCV*, onde o mesmo seleciona o teste com maior acurácia.

## 2.3 Resultados

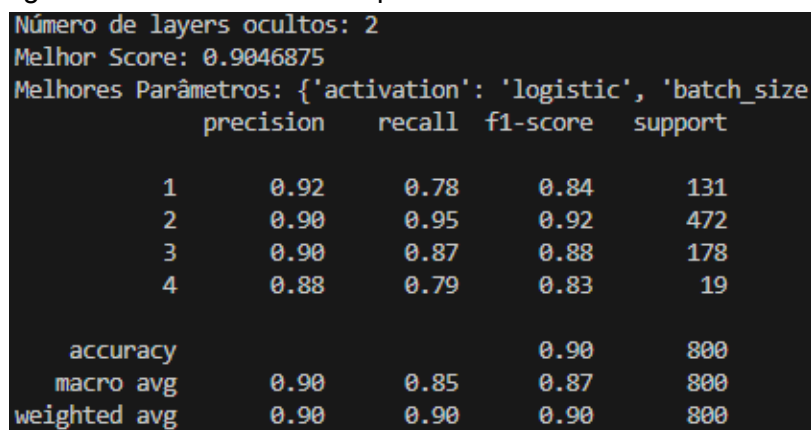
Os testes são realizados todos de uma vez, então apresenta-se apenas o melhor resultado de todas as possibilidades. Foram realizados pelo programa 54 candidatos, com 270 fits. Esse teste foi feito 3 vezes, devido à randomização dos treinamentos.

### 2.3.1 Primeiro Treino

Dentre os 54 candidatos foi escolhido o melhor com os seguintes parâmetros:

- Ativação: Sigmoid (Logistic)
- Camadas ocultas: 30, 10
- Solver: Adam
- batch-size: 40

Segue Figura 1 com os resultados apresentados:



```
Número de layers ocultos: 2
Melhor Score: 0.9046875
Melhores Parâmetros: {'activation': 'logistic', 'batch_size': 40, 'camadas_ocultas': [30, 10], 'solver': 'adam'}

      precision    recall  f1-score   support

     1         0.92         0.78         0.84         131
     2         0.90         0.95         0.92         472
     3         0.90         0.87         0.88         178
     4         0.88         0.79         0.83          19

 accuracy         0.90         0.90         0.90         800
 macro avg         0.90         0.85         0.87         800
 weighted avg         0.90         0.90         0.90         800
```

Figura 1: Resultados do primeiro treino

Na Figura 2 e Figura 3, temos nossa matriz de confusão e curva de perda:

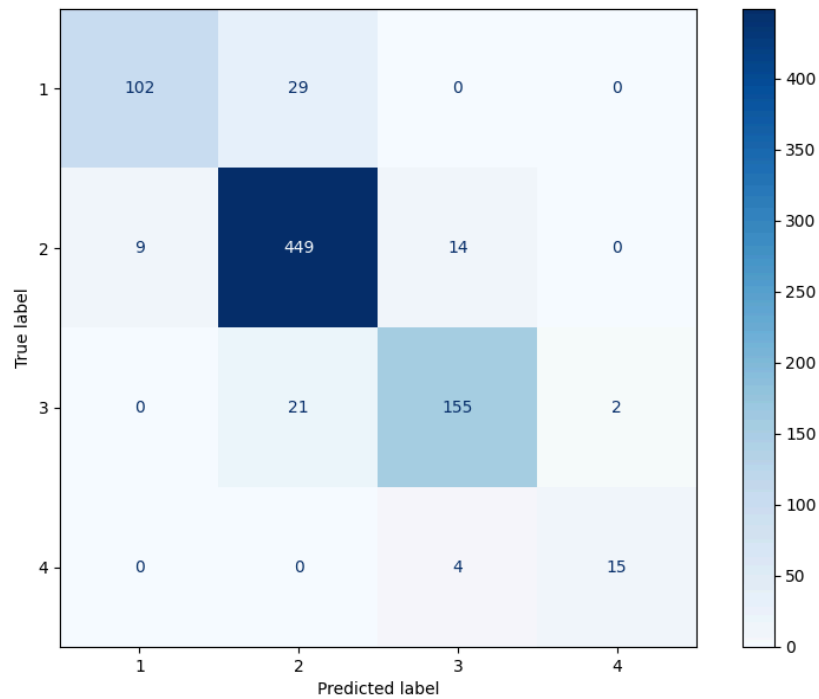


Figura 2: Matriz de Confusão treino 1

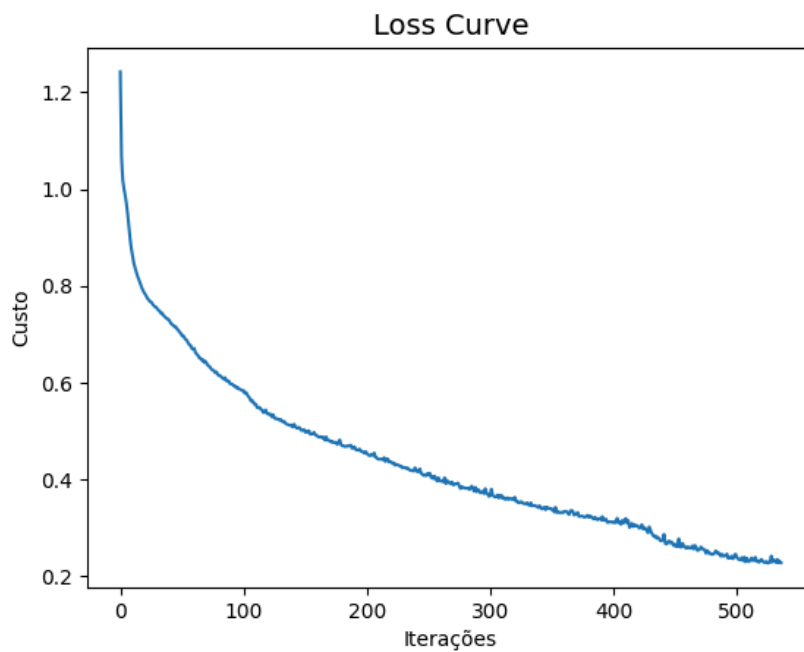


Figura 3: Curva de perda treino 1

### 2.3.1.1 Teste Cego

Segue os resultados do teste cego no dataset de 800 vítimas, Figura 4:

```

Previsões do tipo 1: 97
Previsões do tipo 2: 482
Previsões do tipo 3: 206
Previsões do tipo 4: 15
Acuracia: 0.90
Taxa de Erro: 0.10
Erro Quadrático Médio: 0.10
Matriz de Confusão:
[[ 89  30   0   0]
 [   8 429  18   0]
 [   0  23 186   2]
 [   0   0   2  13]]

```

Figura 4: Resultados do teste cego.

Observa-se que a acurácia obtida no teste cego, está congruente com o valor dos treinos, assim sendo validada.

### 2.3.2 Segundo Treino

Dentre os 54 candidatos foi escolhido o melhor com os seguintes parâmetros:

- Ativação: Sigmoid (Logistic)
- Camadas ocultas: 30, 10
- Solver: Adam
- batch-size: 40

Segue Figura 5 com os resultados apresentados:

```

Número de layers ocultos: 2
Accuracy: 0.8875
Melhor Score: 0.89875
Melhores Parâmetros: {'activation': 'logistic', 'batch_size': 40}

```

	precision	recall	f1-score	support
1	0.85	0.79	0.82	117
2	0.87	0.95	0.91	449
3	0.96	0.81	0.87	216
4	0.94	0.83	0.88	18
accuracy			0.89	800
macro avg	0.90	0.85	0.87	800
weighted avg	0.89	0.89	0.89	800

Figura 5: Resultados segundo treino

Na Figura 6 e Figura 7, temos nossa matriz de confusão e curva de perda:

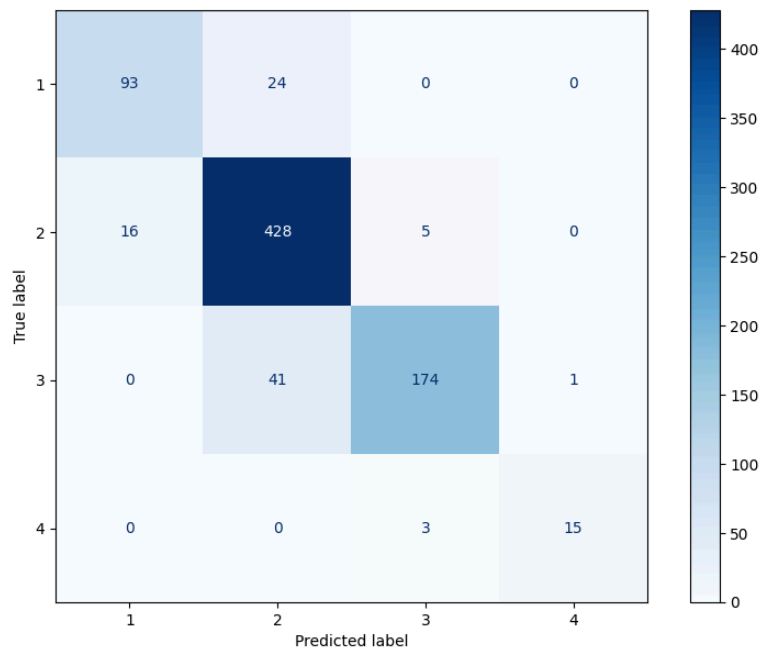


Figura 6: Matriz de confusão treino 2

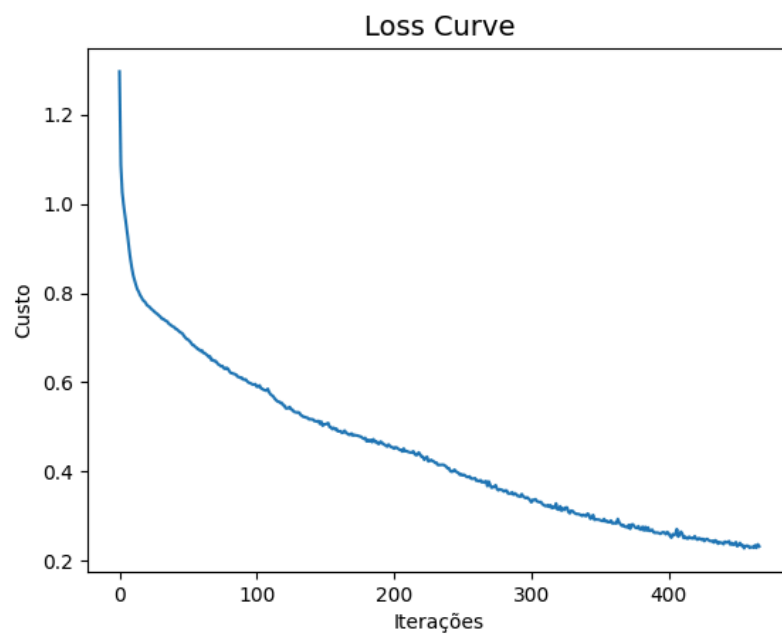


Figura 7: Curva de perda treino 2

### 2.3.2.1 Teste Cego

Segue os resultados do teste cego no dataset de 800 vítimas, Figura 8:



```

Previsões do tipo 1: 107
Previsões do tipo 2: 491
Previsões do tipo 3: 191
Previsões do tipo 4: 11
Acuracia: 0.90
Taxa de Erro: 0.10
Erro Quadrático Médio: 0.10
Matriz de Confusão:
[[ 92  27   0   0]
 [ 15 434   6   0]
 [   0  30 181   0]
 [   0   0   4  11]]

```

Figura 8: Resultados do teste cego.

Observa-se que a acurácia obtida no teste cego, está um pouco acima dos valores dos treinos, assim sendo validada.

### 2.3.3 Terceiro Treino

Dentre os 54 candidatos foi escolhido o melhor com os seguintes parâmetros:

- Ativação: Tanh
- Camadas ocultas: 50, 10
- Solver: Adam
- batch-size: 40

Segue Figura 9 com os resultados apresentados:

```

Número de layers ocultos: 2
Accuracy: 0.8788
Melhor Score: 0.8834375000000001
Melhores Parâmetros: {'activation': 'tanh', 'batch_size': 40}

```

	precision	recall	f1-score	support
1	0.86	0.77	0.81	118
2	0.89	0.92	0.91	467
3	0.88	0.85	0.86	202
4	0.67	0.92	0.77	13
accuracy			0.88	800
macro avg	0.82	0.86	0.84	800
weighted avg	0.88	0.88	0.88	800

Figura 9: Resultados do terceiro treino.

Na Figura 10 e Figura 11, temos nossa matriz de confusão e curva de perda:

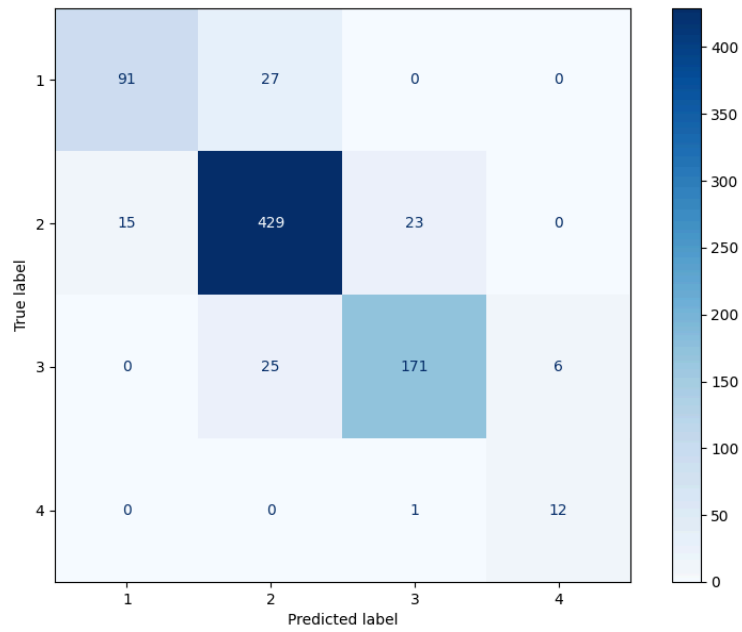


Figura 10: Matriz de Confusão treino 3.

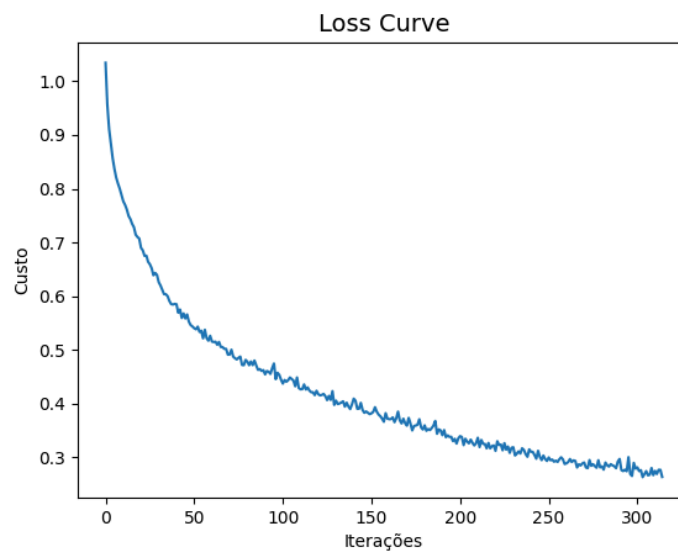


Figura 11: Curva de perda treino 3.

### 2.3.3.1 Teste Cego

Segue os resultados do teste cego no dataset de 800 vítimas, Figura 12:

```
Previsões do tipo 1: 106
Previsões do tipo 2: 461
Previsões do tipo 3: 215
Previsões do tipo 4: 18
Acuracia: 0.88
Taxa de Erro: 0.12
Erro Quadrático Médio: 0.12
Matriz de Confusão:
[[ 91  28   0   0]
 [ 15 413  27   0]
 [   0  20 186   5]
 [   0   0   2  13]]
```

Figura 12: Resultados do teste cego.

Observa-se que a acurácia obtida no teste cego está congruente com os valores dos treinos, assim sendo validada.

## 2.4 Análise dos resultados

Observando os resultados, infelizmente, eles são ruins, pois possuem valores de acurácia muito abaixo do esperado. Para esse tipo de situação, esperávamos obter algum resultado acima de 95% de aproveitamento, assim sendo possível considerar sua implementação.

Contudo, mais testes extensivos podem ser realizados, alterando outros hiperparâmetros, modificando o *batch-size*, a tolerância de parada, e até mesmo variando ainda mais as camadas ocultas utilizadas. Observa-se que dentre os 3 testes, sempre obteve-se o melhor resultado com apenas 2 camadas. Podendo assim realizar mais testes com apenas duas camadas.

Encontrar a solução ótima desses treinamentos é realmente difícil e requer muitos mais testes, ou pelo menos para encontrar uma solução com 95% de aproveitamento.

## 3. Árvore de Decisão

### 3.1 Fundamentação e modelagem

Árvore de decisão é um modelo de aprendizado supervisionado usado para classificação e regressão. A árvore é composta por nós de decisão, ramos e folhas, onde cada nó representa uma característica (atributo), cada ramo representa uma regra de decisão, e cada folha representa uma saída (classe ou valor). No caso, o modelo de múltiplas classes foi utilizado.

A árvore tem a estrutura definida na forma de nós de decisão, ramos e folhas, onde os nós representam uma decisão baseada em um atributo (feature) dos dados, os ramos representam as possíveis saídas (ou decisões) baseadas nos atributos e as folhas representam a decisão final ou a previsão do modelo, a classe (conforme escolhido).

Para a criação do modelo treinado foram utilizadas três características, listadas no dataset de treino sendo elas: qPa, Pulso e frequência respiratória, além do resultado esperado (classe alvo) denominada aqui e nos códigos de  $y$ . A saída gerada é o melhor modelo treinado com base na entropia e outros dados referentes a qualidade desse modelo que serão melhor explicados no capítulo de comparações entre técnicas adotadas.

### 3.2 Parametrizações utilizadas

Para a divisão dos nós da árvore, foi adotada a entropia. A entropia mede a impureza dos dados em cada nó da árvore, e a divisão é feita de forma a minimizar a entropia após a divisão. No entanto, não foi adotada nenhuma técnica de poda ou similar.

Outra técnica adotada foi a de profundidade máxima, a profundidade da árvore refere-se ao número de níveis de nós que a árvore pode ter, começando pela raiz e indo até as folhas. Limitar a profundidade da árvore ajuda a evitar o *overfitting*, que ocorre quando a árvore se ajusta excessivamente aos dados de treinamento e captura padrões que não generalizam bem para novos dados. A profundidade máxima da árvore foi configurada através do parâmetro *max\_depth* na função *GridSearchCV*, de uma biblioteca específica para criação de árvores de decisão. Foram testados diferentes valores para esse parâmetro para encontrar o melhor ajuste para o modelo. O modelo avalia essas opções durante a busca em grade e seleciona o valor que resulta no melhor desempenho, de acordo com a métrica escolhida (neste caso, `f1_macro`).

O parâmetro mínimo de amostras por folha define o número mínimo de amostras que um nó folha deve ter. Esse parâmetro controla o tamanho dos nós folha da árvore. Definir um valor para *min\_samples\_leaf* ajuda a evitar que a árvore se torne muito complexa, criando nós folha com muito poucas amostras, o que pode levar ao *overfitting*. Esses parâmetros foram configurados e ajustados usando *GridSearchCV*, que realiza uma busca exaustiva sobre as combinações possíveis desses parâmetros para encontrar a configuração que otimiza o desempenho do modelo.

### 3.2 Treino e validação

O treinamento e a validação do modelo foram realizados utilizando a abordagem de validação cruzada com a função *GridSearchCV*. Primeiramente, os dados foram divididos em conjuntos de treinamento e teste usando a função *train\_test\_split*, com o parâmetro *test\_size* definido como 0.25. Isso significa que 25% dos dados foram reservados para teste e 75% para treinamento. A divisão dos dados foi feita de forma aleatória, mas reproduzível, utilizando a semente de aleatoriedade *random\_state=42*.

Os parâmetros da árvore de decisão foram definidos em um dicionário chamado *parameters*, incluindo:

- Critério de Divisão (*criterion*): ['entropy']
- Profundidade Máxima (*max\_depth*): [4, 6, 80, 100, 200]
- Mínimo de Amostras por Folha (*min\_samples\_leaf*): [2, 3, 5, 10]

O *GridSearchCV* foi configurado para realizar uma busca exaustiva sobre todas as combinações possíveis desses parâmetros. Utilizou validação cruzada com 30 dobras

(cv=30) e a métrica de avaliação foi definida como *f1\_macro*, que fornece uma visão equilibrada do desempenho do modelo em diferentes classes.

Durante o processo, o *GridSearchCV* ajustou o modelo (fit) aos dados de treinamento (*X\_train*, *y\_train*), treinando o modelo em todas as combinações de parâmetros especificados e avaliando cada uma usando validação cruzada. Após a execução da busca, o *GridSearchCV* selecionou os melhores parâmetros que resultaram na melhor pontuação de *f1\_macro*.

O melhor modelo foi acessado através do atributo *best\_estimator\_* do *GridSearchCV*, que contém a configuração de parâmetros que obteve o melhor desempenho. Esse modelo foi então utilizado para fazer previsões tanto nos dados de treinamento quanto nos dados de teste. As métricas de desempenho, como acurácia e matriz de confusão, foram calculadas para avaliar a qualidade do modelo.

### 3.3 Resultados

O método *GridSearchCV* realizou 600 ajustes de modelo no total, para cada uma das 20 combinações de hiperparâmetros, ele fez 30 treinos usando diferentes divisões dos dados de treinamento (30 *folds* de validação cruzada).

#### 3.3.1 Primeiro Teste

A melhor combinação foi uma árvore com profundidade de 80. Na Figura XX a seguir, estão ilustradas as pontuações obtidas para cada combinação de parâmetros, bem como o percentual de acerto de previsão nos dados de treinamento (99,17%) e nos dados de teste (95,20%) alcançados pela árvore considerada a melhor.

```
* Melhor classificador *
DecisionTreeClassifier(criterion='entropy', max_depth=80, min_samples_leaf=2,
                      random_state=42)
Pontuações para cada combinação de parâmetros:
Parâmetros: {'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 2} - Pontuação média: 0.5637
Parâmetros: {'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 3} - Pontuação média: 0.5637
Parâmetros: {'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 5} - Pontuação média: 0.5637
Parâmetros: {'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 10} - Pontuação média: 0.5637
Parâmetros: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf': 2} - Pontuação média: 0.6566
Parâmetros: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf': 3} - Pontuação média: 0.6558
Parâmetros: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf': 5} - Pontuação média: 0.6566
Parâmetros: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf': 10} - Pontuação média: 0.6566
Parâmetros: {'criterion': 'entropy', 'max_depth': 80, 'min_samples_leaf': 2} - Pontuação média: 0.8687
Parâmetros: {'criterion': 'entropy', 'max_depth': 80, 'min_samples_leaf': 3} - Pontuação média: 0.8677
Parâmetros: {'criterion': 'entropy', 'max_depth': 80, 'min_samples_leaf': 5} - Pontuação média: 0.8608
Parâmetros: {'criterion': 'entropy', 'max_depth': 80, 'min_samples_leaf': 10} - Pontuação média: 0.8520
Parâmetros: {'criterion': 'entropy', 'max_depth': 100, 'min_samples_leaf': 2} - Pontuação média: 0.8687
Parâmetros: {'criterion': 'entropy', 'max_depth': 100, 'min_samples_leaf': 3} - Pontuação média: 0.8677
Parâmetros: {'criterion': 'entropy', 'max_depth': 100, 'min_samples_leaf': 5} - Pontuação média: 0.8608
Parâmetros: {'criterion': 'entropy', 'max_depth': 100, 'min_samples_leaf': 10} - Pontuação média: 0.8520
Parâmetros: {'criterion': 'entropy', 'max_depth': 200, 'min_samples_leaf': 2} - Pontuação média: 0.8687
Parâmetros: {'criterion': 'entropy', 'max_depth': 200, 'min_samples_leaf': 3} - Pontuação média: 0.8677
Parâmetros: {'criterion': 'entropy', 'max_depth': 200, 'min_samples_leaf': 5} - Pontuação média: 0.8608
Parâmetros: {'criterion': 'entropy', 'max_depth': 200, 'min_samples_leaf': 10} - Pontuação média: 0.8520
Acurácia com dados de treino: 99.17%
Acurácia com dados de teste: 95.20%
```

Figura 13: Pontuação média com base em entropia do melhor caso de cada combinação (Árvore-1).

O *classification\_report*, na figura XX fornece uma visão detalhada do desempenho do modelo de classificação. Ele inclui métricas como precisão, recall e F1-score para cada

classe no conjunto de dados de teste. A precisão mede a proporção de previsões corretas para cada classe, o recall avalia a capacidade do modelo de identificar todas as instâncias de uma classe, e o F1-score é a média harmônica entre precisão e recall, oferecendo uma métrica balanceada do desempenho geral.

	precision	recall	f1-score	support
Classe 0	0.98	0.96	0.97	176
Classe 1	0.95	0.97	0.96	568
Classe 2	0.93	0.91	0.92	243
Classe 3	0.91	0.77	0.83	13
accuracy			0.95	1000
macro avg	0.94	0.90	0.92	1000
weighted avg	0.95	0.95	0.95	1000

Figura 14: Métrica de desempenho do teste 1 (Árvore).

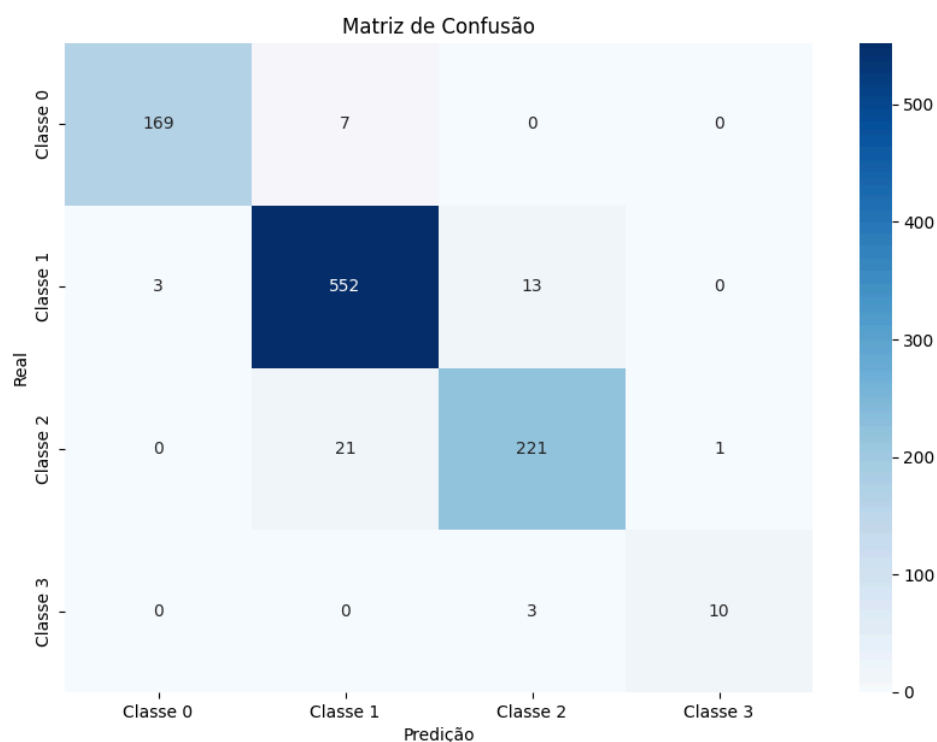


Figura 15: Matriz de Confusão do teste 1 (Árvore)

### 3.3.2 Segundo Teste

Os valores apresentados nas figuras xx, xx mostram que a árvore definida como a melhor têm as mesmas características que a do primeiro teste mas a taxa de acerto da previsão sobre os dados de teste diminuiu.

```
* Melhor classificador *
DecisionTreeClassifier(criterion='entropy', max_depth=80, min_samples_leaf=2,
                      random_state=42)
Pontuações para cada combinação de parâmetros:
Parâmetros: {'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 2} - Pontuação média: 0.5618
Parâmetros: {'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 3} - Pontuação média: 0.5618
Parâmetros: {'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 5} - Pontuação média: 0.5618
Parâmetros: {'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 10} - Pontuação média: 0.5624
Parâmetros: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf': 2} - Pontuação média: 0.7720
Parâmetros: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf': 3} - Pontuação média: 0.7709
Parâmetros: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf': 5} - Pontuação média: 0.7700
Parâmetros: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf': 10} - Pontuação média: 0.7678
Parâmetros: {'criterion': 'entropy', 'max_depth': 80, 'min_samples_leaf': 2} - Pontuação média: 0.8883
Parâmetros: {'criterion': 'entropy', 'max_depth': 80, 'min_samples_leaf': 3} - Pontuação média: 0.8790
Parâmetros: {'criterion': 'entropy', 'max_depth': 80, 'min_samples_leaf': 5} - Pontuação média: 0.8842
Parâmetros: {'criterion': 'entropy', 'max_depth': 80, 'min_samples_leaf': 10} - Pontuação média: 0.8427
Parâmetros: {'criterion': 'entropy', 'max_depth': 100, 'min_samples_leaf': 2} - Pontuação média: 0.8883
Parâmetros: {'criterion': 'entropy', 'max_depth': 100, 'min_samples_leaf': 3} - Pontuação média: 0.8790
Parâmetros: {'criterion': 'entropy', 'max_depth': 100, 'min_samples_leaf': 5} - Pontuação média: 0.8842
Parâmetros: {'criterion': 'entropy', 'max_depth': 100, 'min_samples_leaf': 10} - Pontuação média: 0.8427
Parâmetros: {'criterion': 'entropy', 'max_depth': 200, 'min_samples_leaf': 2} - Pontuação média: 0.8883
Parâmetros: {'criterion': 'entropy', 'max_depth': 200, 'min_samples_leaf': 3} - Pontuação média: 0.8790
Parâmetros: {'criterion': 'entropy', 'max_depth': 200, 'min_samples_leaf': 5} - Pontuação média: 0.8842
Parâmetros: {'criterion': 'entropy', 'max_depth': 200, 'min_samples_leaf': 10} - Pontuação média: 0.8427
Acurácia com dados de treino: 99.23%
Acurácia com dados de teste: 94.10%
```

Figura 16: Pontuação média com base em entropia do melhor caso de cada combinação (Árvore-2).

	precision	recall	f1-score	support
Classe 0	0.97	0.97	0.97	179
Classe 1	0.94	0.96	0.95	563
Classe 2	0.92	0.88	0.90	246
Classe 3	0.75	0.75	0.75	12
accuracy			0.94	1000
macro avg	0.90	0.89	0.89	1000
weighted avg	0.94	0.94	0.94	1000

Figura 17: Métrica de desempenho teste 2 (Árvore).

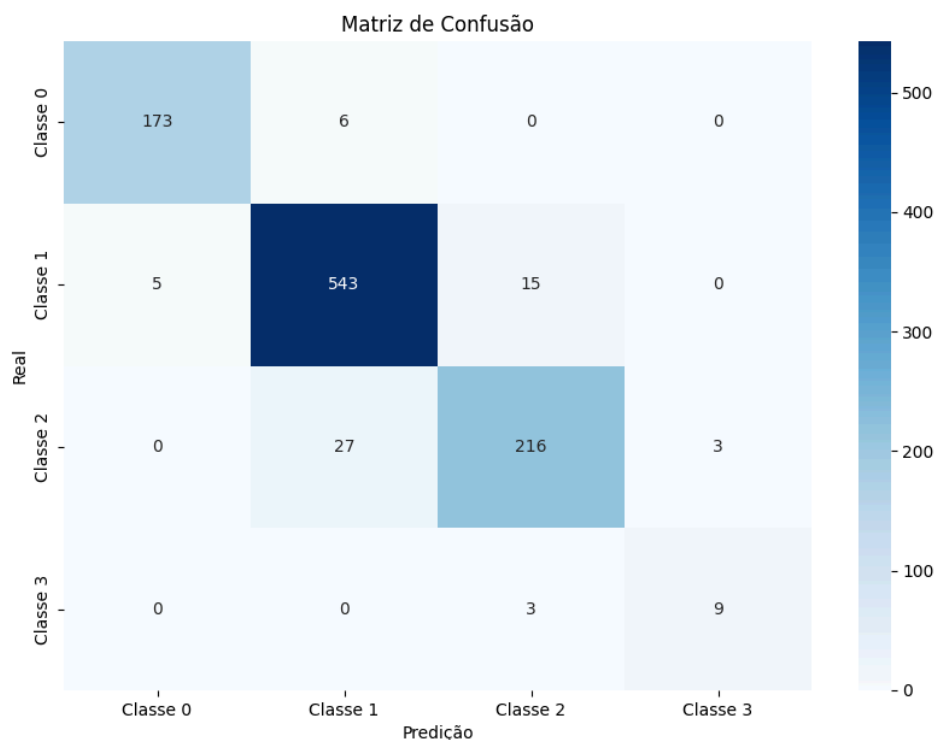


Figura 18: Matriz de Confusão do teste 2 (Árvore)

### 3.3.3 Terceiro Teste

Novamente, na terceira criação, os valores apresentados nas figuras XX e XX indicam que a árvore classificada como a melhor possui as mesmas características que a do primeiro teste. No entanto, a taxa de acerto da previsão nos dados de teste diminuiu.

```
* Melhor classificador *
DecisionTreeClassifier(criterion='entropy', max_depth=80, min_samples_leaf=2,
                      random_state=42)
Pontuações para cada combinação de parâmetros:
Parâmetros: {'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 2} - Pontuação média: 0.5649
Parâmetros: {'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 3} - Pontuação média: 0.5649
Parâmetros: {'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 5} - Pontuação média: 0.5649
Parâmetros: {'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 10} - Pontuação média: 0.5623
Parâmetros: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf': 2} - Pontuação média: 0.7976
Parâmetros: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf': 3} - Pontuação média: 0.7983
Parâmetros: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf': 5} - Pontuação média: 0.7956
Parâmetros: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf': 10} - Pontuação média: 0.7876
Parâmetros: {'criterion': 'entropy', 'max_depth': 80, 'min_samples_leaf': 2} - Pontuação média: 0.8701
Parâmetros: {'criterion': 'entropy', 'max_depth': 80, 'min_samples_leaf': 3} - Pontuação média: 0.8700
Parâmetros: {'criterion': 'entropy', 'max_depth': 80, 'min_samples_leaf': 5} - Pontuação média: 0.8465
Parâmetros: {'criterion': 'entropy', 'max_depth': 80, 'min_samples_leaf': 10} - Pontuação média: 0.8236
Parâmetros: {'criterion': 'entropy', 'max_depth': 100, 'min_samples_leaf': 2} - Pontuação média: 0.8701
Parâmetros: {'criterion': 'entropy', 'max_depth': 100, 'min_samples_leaf': 3} - Pontuação média: 0.8700
Parâmetros: {'criterion': 'entropy', 'max_depth': 100, 'min_samples_leaf': 5} - Pontuação média: 0.8465
Parâmetros: {'criterion': 'entropy', 'max_depth': 100, 'min_samples_leaf': 10} - Pontuação média: 0.8236
Parâmetros: {'criterion': 'entropy', 'max_depth': 200, 'min_samples_leaf': 2} - Pontuação média: 0.8701
Parâmetros: {'criterion': 'entropy', 'max_depth': 200, 'min_samples_leaf': 3} - Pontuação média: 0.8700
Parâmetros: {'criterion': 'entropy', 'max_depth': 200, 'min_samples_leaf': 5} - Pontuação média: 0.8465
Parâmetros: {'criterion': 'entropy', 'max_depth': 200, 'min_samples_leaf': 10} - Pontuação média: 0.8236
Acurácia com dados de treino: 99.07%
Acurácia com dados de teste: 93.80%
```

Figura 19: Pontuação média com base em entropia do melhor caso de cada combinação (Árvore-2).



	precision	recall	f1-score	support
Classe 0	0.98	0.98	0.98	161
Classe 1	0.95	0.96	0.96	568
Classe 2	0.90	0.88	0.89	251
Classe 3	0.80	0.60	0.69	20
accuracy			0.94	1000
macro avg	0.91	0.86	0.88	1000
weighted avg	0.94	0.94	0.94	1000

Figura 20: Métrica de desempenho do teste 3 (Árvore).

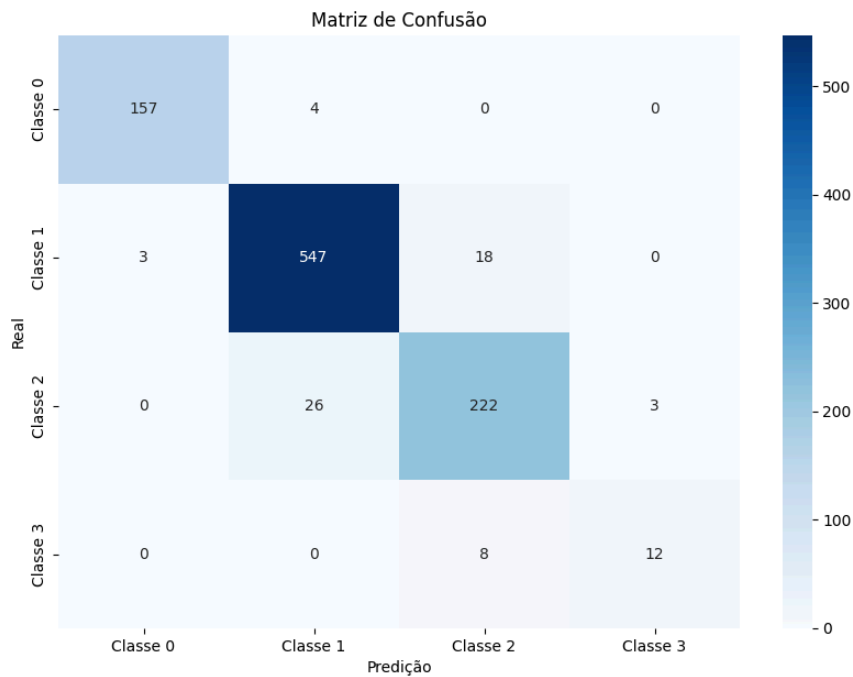


Figura 21: Matriz de Confusão do teste 3 (Árvore)

### 3.3.4 Teste cego

O teste cego foi realizado utilizando um dataset de 800 vítimas, onde uma árvore de decisão foi aplicada para determinar a classe de gravidade para cada vítima. Após a geração das previsões, a acurácia foi calculada para verificar quantas vítimas foram corretamente classificadas de acordo com as previsões do modelo, resultando em uma acurácia de 94%, conforme ilustrado na figura 22. Também foram calculadas a quantidade correta de cada classe, a quantidade prevista e as taxas de erro.

```

valores 'corretos'
y
2    455
3    211
1    119
4     15
Name: count, dtype: int64
-----
Previsões do tipo 1: 120
Previsões do tipo 2: 460
Previsões do tipo 3: 206
Previsões do tipo 4: 14
Acurácia: 0.94
Taxa de Erro: 0.06
Erro Quadrático Médio: 0.07

```

Figura 22: Acurácia e quantidades obtidas.

As figuras 23 e 24 representam, respectivamente, as métricas de desempenho e a matriz de confusão geradas pelo teste cego.

	precision	recall	f1-score	support
Classe 0	0.95	0.96	0.95	119
Classe 1	0.94	0.95	0.95	455
Classe 2	0.91	0.89	0.90	211
Classe 3	0.93	0.87	0.90	15
accuracy			0.94	800
macro avg	0.93	0.92	0.92	800
weighted avg	0.93	0.94	0.93	800

Figura 23: métricas de desempenho geradas pelo teste cego.

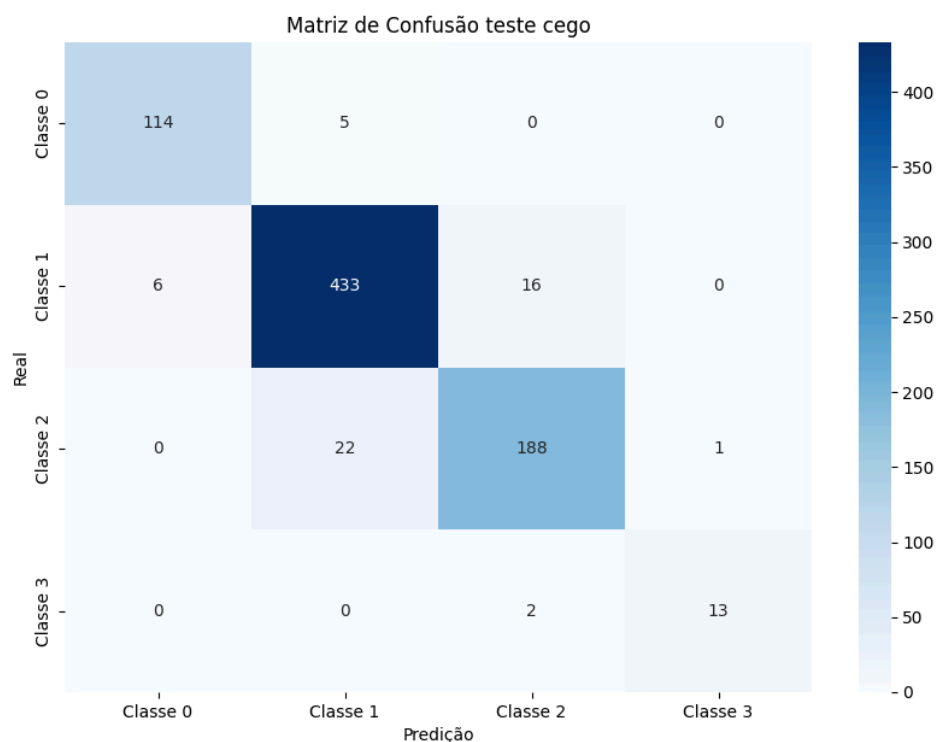


Figura 24: Matriz de confusão gerada pelo teste cego.

## 4. Algoritmo Genético

### 4.1 Fundamentação e modelagem

As pesquisas sobre modelos computacionais inteligentes têm se caracterizado por buscar inspiração na natureza, onde existem inúmeros exemplos de processos que, de certa forma, podem se enquadrar como “inteligentes”. Por exemplo, a Computação Evolucionária (CE) encara a teoria da evolução Darwiniana como um processo adaptativo de otimização, sugerindo um modelo em que populações de estruturas computacionais evoluem de modo a melhorar, em média, o desempenho geral da população frente a um problema. Esta premissa guia uma das técnicas mais difundidas e estudadas da CE, os Algoritmos Genéticos (AGs).

Uma grande parte dos problemas científicos decorre de problemas de busca e otimização. De forma global, a evolução natural implementa mecanismos adaptativos de otimização que, embora não sejam uma forma de busca aleatória, com certeza utilizam aleatoriedade. Os AGs são métodos probabilísticos de busca e otimização, apesar de não serem aleatórios, buscam regiões do espaço onde é provável que os pontos sejam ótimos.

#### 4.1.1 Características

- AGs operam numa população de indivíduos
- AGs operam num espaço de soluções codificadas
- AGs necessitam somente de uma função objetivo para cada membro da população

#### 4.1.3 Representação Cromossômica

Cada solução para um determinado problema corresponde a um cromossomo, e cada elemento da solução é equivalente a um gene. O valor que cada gene assume corresponde ao Alelo. Neste problema de resgate, um cromossomo representa uma sequência de salvamento das vítimas (genes) pelos agentes socorristas.

#### 4.1.4 Fluxo Básico

Tendo definido a representação cromossômica, gera-se um conjunto de soluções possíveis, chamadas soluções-candidatas. Um conjunto de soluções codificadas de acordo com a representação corresponde à população de indivíduos. Cada iteração do AG é denominada geração.

#### 4.1.5 Modelagem

Nesta implementação, foi utilizada a biblioteca para Python 'DEAP', que possui métodos que auxiliam na adaptação do problema do resgate para uma abordagem utilizando algoritmo genético.

Um cromossomo “i” inicial é criado com as vítimas encontradas pelo explorador “i”. A partir desse indivíduo, são criados outros n cromossomos, onde n representa o tamanho da população. O código `toolbox.register("individual", tools.initIterate, creator.Individual, lambda: random.sample(clusterVitimas,`

`len(clusterVitimass))`” registra uma função para criar indivíduos (cromossomos) na sua população inicial. Essa função `random.sample` seleciona uma amostra aleatória de elementos da lista `clusterVitimass` sem repetição. Ela embaralha a lista de vítimas, gerando uma nova sequência aleatória a cada vez que é chamada.

A partir do momento que a população inicial é criada, a primeira geração do problema itera sobre o algoritmo e cada cromossomo é avaliado pela função objetivo, nesse caso, `avaliar_cromossomo`. A ideia consiste em minimizar o valor de “fitness” que é calculado com base no grau de gravidade de cada vítima e o tamanho do percurso entre elas. Ou seja, o “fitness” deve ser menor para aqueles cromossomos onde as vítimas com valor crítico de gravidade estão nos primeiros genes e o percurso entre elas é, de certa forma, mínimo.

## 4.2 Parametrizações utilizadas

### 4.2.1 Operador Crossover

O fenômeno de “crossover” é a troca de fragmentos entre pares de cromossomos, trata-se de um processo aleatório que ocorre com probabilidade fixa (taxa de crossover). A função `tools.cxPartiallyMatched` do “DEAP” é um método de crossover utilizado em algoritmos genéticos para realizar a troca de segmentos entre dois indivíduos (pais) enquanto preserva a posição relativa dos elementos não trocados, ou seja, garante que não haja repetição de genes no cromossomo, uma vez que cada vítima para ser salva é única. Esse operador é especialmente útil em qualquer problema onde a permutação dos genes (ordem) importa. Para esse problema foi aplicado uma taxa de crossover de 0,8

### 4.2.2 Operador Mutação

O processo de mutação é equivalente à busca aleatória. Basicamente, seleciona-se uma posição num cromossomo e muda-se o valor do gene correspondente para um outro alelo possível. O processo é controlado por um parâmetro (taxa de mutação) que indica a probabilidade de um gene sofrer mutação. No problema do resgate de vítimas, o método `“mutate”` do “DEAP” implementa uma Order-Based Mutation, ou seja, troca um gene da posição “i” pelo gene da posição “j”. O parâmetro `“indpd”` representa a taxa de mutação, cujo valor é 0,1.

### 4.2.3 Método de Seleção

Nesse problema, foi adotado o método de seleção por Torneio. A função `“selTournament”` implementa um método de seleção por torneio fornecida pela biblioteca “DEAP”. Consiste basicamente em selecionar um grupo de “k” indivíduos aleatórios da população, o montante “k” é definido pelo parâmetro `“tournsize”`, cujo valor é 3. O processo de seleção não leva em conta o valor do “fitness”. Após a seleção dos “k” cromossomos, escolhe-se aquele que possui o melhor “fitness”.

### 4.2.4 Função Fitness e penalização

A função de fitness é um componente essencial em algoritmos genéticos, usado para avaliar a qualidade ou aptidão de uma solução (cromossomo) para um determinado problema. Em termos simples, a função de fitness atribui um valor numérico a cada cromossomo, indicando quão "boa" é a solução que ele representa em relação ao objetivo do problema. No contexto do problema de resgate de vítimas, a função de fitness é utilizada para avaliar a sequência de salvamento de vítimas. Cada cromossomo é uma sequência ordenada de vítimas, onde cada vítima possui uma posição (x, y) e um valor de gravidade, que pode variar de 1 a 4, sendo 1 a gravidade mais alta (crítica) e 4 a gravidade mais baixa (estável). O objetivo do algoritmo genético é encontrar uma sequência (cromossomo) que minimize o tempo total de resgate, priorizando as vítimas mais graves e mais próximas da posição inicial do resgatador. Para isso, a função avalia cada cromossomo com base nos seguintes critérios:

Distância Total Percorrida:

- A função calcula a distância total que o resgatador precisa percorrer para salvar todas as vítimas na ordem especificada pelo cromossomo. Quanto menor essa distância, melhor (menor valor fitness).

Penalidade:

- A função também aplica uma penalidade para desincentivar sequências onde vítimas menos graves (com maior valor de gravidade) aparecem antes de vítimas mais graves. Isso é feito para garantir que o algoritmo priorize corretamente o salvamento das vítimas mais urgentes.

### 4.3 Validação e Testes

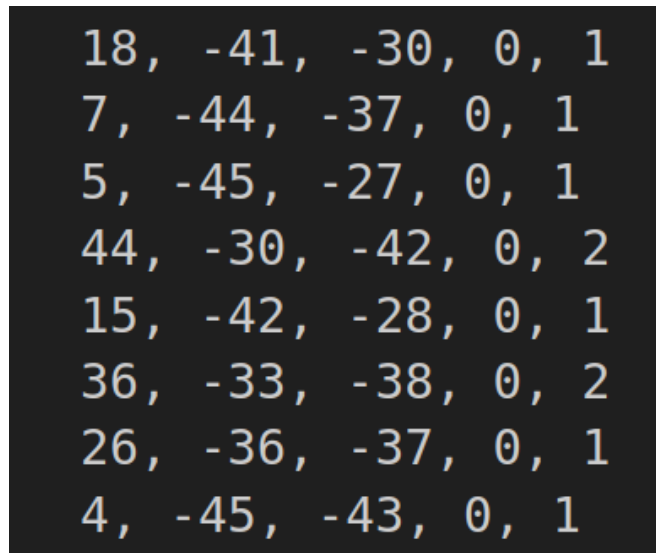
Depois de validar com cenários simples (132 vítimas), testamos o algoritmo em cenários mais complexos (408 vítimas) para observar como ele se comporta. Para cada teste, modificamos o tamanho da população, o número de gerações e os parâmetros do algoritmo genético. Nesse problema, concluímos que um bom valor para o tamanho da população seria 30 e para o número de gerações, 40.

Executamos o algoritmo várias vezes com os mesmos parâmetros e verificamos a consistência das soluções encontradas. Se o algoritmo frequentemente converge para a mesma solução ou para soluções similares, isso sugere que ele está encontrando soluções próximas da ótima.

Monitoramos a evolução da função fitness ao longo das gerações. Se o valor da fitness estiver melhorando consistentemente, isso indica que o algoritmo está convergindo para uma solução melhor.

### 4.3 Resultados

Observando os resultados obtidos para o cenário de 408v / 94x94, tivemos 6,88% de taxa de salvamento das vítimas de 407 encontradas. Abaixo, está uma parte da sequência de vítimas retornada pelo algoritmo genético para o rescatador 1.



```
18, -41, -30, 0, 1
7, -44, -37, 0, 1
5, -45, -27, 0, 1
44, -30, -42, 0, 2
15, -42, -28, 0, 1
36, -33, -38, 0, 2
26, -36, -37, 0, 1
4, -45, -43, 0, 1
```

Figura 25: Parte da sequência de salvamento para o socorrista 1.

## 5. Implementação e análises

A implementação do código fez uso de bibliotecas prontas para a criação de modelos de árvore de decisão e redes neurais. Especificamente, utilizamos o Scikit-learn para implementar a árvore de decisão e o classificador de rede neural (*MLPClassifier*), aproveitando suas funcionalidades para treinamento, validação e ajuste de hiperparâmetros dos modelos. Essas bibliotecas facilitaram a aplicação de algoritmos de machine learning complexos, permitindo o foco na otimização e análise dos resultados.

A implementação do código também utilizou uma biblioteca pronta para a criação de algoritmos genéticos. Foi empregado o pacote *DEAP* para desenvolver e otimizar o algoritmo genético, aproveitando suas ferramentas para definir a função de aptidão, realizar operações de crossover e mutação, e selecionar os melhores indivíduos ao longo das gerações. O uso dessa biblioteca simplificou o desenvolvimento do algoritmo genético, permitindo concentrar os esforços na adaptação do modelo.

A escolha de utilizar a árvore de decisão foi fundamentada nos resultados dos testes realizados, que demonstraram que este modelo gerou uma maior acurácia em comparação com a rede neural. Além disso, a árvore de decisão apresentou um tempo de treinamento e construção significativamente mais rápido, tornando-se uma opção mais eficiente tanto em termos de desempenho quanto de processamento.

## 6. Conclusão

A árvore de decisão implementada demonstrou um desempenho satisfatório nos testes realizados, atingindo os objetivos estabelecidos de precisão e eficácia na classificação dos sinais vitais. O modelo obteve uma acurácia superior a 90%, evidenciando sua capacidade de classificar corretamente as diferentes classes de sinais vitais. Esse desempenho é notável, especialmente considerando que os dados utilizados para treinamento e teste foram criados com um conjunto mais amplo de características para definição das classes, das quais duas foram posteriormente excluídas. A análise dos resultados sugere que a árvore de decisão está bem ajustada aos dados, com uma diferença mínima entre o desempenho em dados de treinamento e validação, indicando baixa probabilidade de sobre adaptação e boa capacidade de generalização para novos dados.

Em contraste, a rede neural Perceptron revelou-se menos eficaz nos testes. A acurácia do modelo Perceptron foi inferior à da árvore de decisão, sugerindo que o modelo teve dificuldades em capturar as complexidades dos dados. Além disso, a diminuição das características, mencionada anteriormente, pode ter contribuído para o desempenho insatisfatório. Outros fatores, como a arquitetura simples da rede, a necessidade de ajustes adicionais nos hiperparâmetros e a possível insuficiência de dados para um treinamento adequado, também podem ter influenciado negativamente nos resultados.

Para aprimorar a solução atual, o código pode ser otimizado e tornado mais eficiente, utilizando melhor as bibliotecas de árvores de decisão, redes neurais e algoritmos genéticos. No futuro, seria interessante explorar redes neurais mais avançadas e outras técnicas para aumentar a eficácia. Além disso, otimizar os hiperparâmetros dos algoritmos genéticos e integrar métodos mais sofisticados para a busca em profundidade pode trazer melhorias significativas. A validação cruzada e o treinamento com *datasets* mais robustos também são essenciais para garantir a robustez e a generalização dos modelos.

Em relação à definição de classes, a solução deve ser considerada neutra, pois não se baseia em identificadores pessoais ou características que não sejam exclusivamente relacionadas a aferições médicas sobre o paciente. No contexto de resgate, o socorrista tem o dever de atender a todas as vítimas identificadas, e não agir dessa forma seria considerado incorreto. A definição de prioridades, que segue uma ordem do mais grave para o menos grave, está adequada ao objetivo final de alcançar e salvar todas as vítimas ainda com vida.

No entanto, a neutralidade pode ser comprometida se houver *datasets* defeituosos usados para criar e validar os modelos de definição de classes. Isso poderia levar a uma classificação incorreta das vítimas, potencialmente resultando em erros no processo de triagem e diminuindo a chance de salvar uma vítima grave que tenha sido erroneamente classificada como estável.

## 7. Referências

1. **Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E.** (2011). *Scikit-learn: Machine Learning in Python*. *Journal of Machine Learning Research*, 12, 2825-2830.
2. **Professor Tacla , UTFPR/Curitiba.** (2024) Slides apresentados em aulas, e fornecidos no moodle. Matéria de Sistemas Inteligentes.
3. **TANOMARU, J.** Motivação, fundamentos e aplicações de algoritmos genéticos. Anais do II Congresso Brasileiro de Redes Neurais, 1995.
4. **SCIKIT-LEARN DEVELOPERS.** *Supervised learning - Neural network models (supervised)*. Disponível em: [https://scikit-learn.org/stable/modules/neural\\_networks\\_supervised.html](https://scikit-learn.org/stable/modules/neural_networks_supervised.html). Acesso em: 29 ago. 2024.
5. **SCIKIT-LEARN DEVELOPERS.** *Decision trees*. Disponível em: <https://scikit-learn.org/stable/modules/tree.html>. Acesso em: 29 ago. 2024.
6. **FUCHS, M.** *NN: Multi-layer perceptron classifier – MLPClassifier*. Disponível em: <https://michael-fuchs-python.netlify.app/2021/02/03/nn-multi-layer-perceptron-classifier-mlpclassifier/>. Acesso em: 29 ago. 2024.