

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | |
|  | | **Conhecimento e Raciocínio** | | | | |  | |
|  |  | | | | | | |  |
|  | | | |  |  | | | |
|  | | | | **João Choupina – 2020151878**  **Ricardo Tavares - 2021144652** |  | | | |
|  | | | | R&D Nester - ISEC - Instituto Superior de Engenharia de Coimbra09/05/2024  —  Redes Neuronais  —  Licenciatura em Engenharia Informática |  | | | |
|  | | |  | | |  | | |

Índice

[Introdução 3](#_Toc166409232)

[Decisões tomadas 4](#_Toc166409233)

[Aplicação 6](#_Toc166409234)

[Analise de Resultados 8](#_Toc166409235)

[Ficheiro Train: 8](#_Toc166409236)

[Ficheiro Start: 9](#_Toc166409237)

[Conclusão 11](#_Toc166409238)

# Introdução

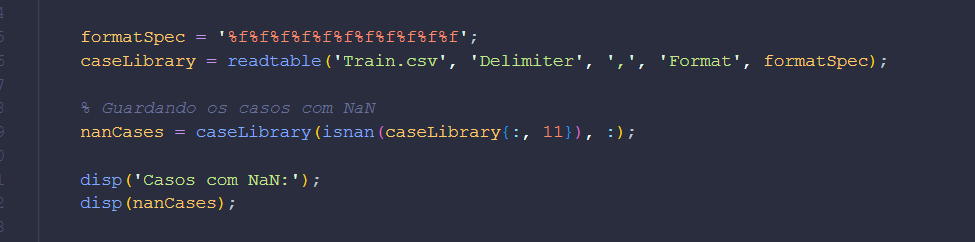
Este relatório descreve o desenvolvimento e análise de um trabalho prático desenvolvido no âmbito da unidade curricular de Conhecimento e Raciocínio, que explora os conceitos de raciocínio baseado em casos e redes neurais. Utilizamos a plataforma MATLAB, especificamente a toolbox Deep Learning, para implementar redes neurais.

O objetivo principal deste trabalho é aplicar os conhecimentos adquiridos nas aulas para explorar e aprofundar os conceitos de raciocínio baseado em casos e redes neurais do tipo Feedforward. Optamos por selecionar o conjunto de dados "stroke" com o intuito de realizar essa aplicação. Adicionalmente desenvolvemos uma interface gráfica básica onde podemos treinar e verificar as redes neuronais.

# Decisões tomadas

**Deteção e Isolamento de Casos com Valores Ausentes (NaN)**

O algoritmo começa por identificar as linhas na tabela principal (*caseLibrary*) que contêm valores ausentes ("NaN") na coluna "stroke", a variável alvo que indica a presença ou ausência de AVC. Essas linhas com dados ausentes são agrupadas em tabelas separadas chamadas *nanCases*. As restantes linhas, sem valores ausentes em "stroke", são mantidas na tabela original *caseLibrary*.



*Fig1. Identificadores casos NaN*

**Casos Semelhantes para Imputação de Valores Ausentes**

Para cada linha na tabela *nanCases*, o algoritmo tenta encontrar linhas semelhantes na tabela *caseLibrary*.

A similaridade é calculada usando uma função especializada de nome *retrieve*. Essa função considera nove atributos dos pacientes, cada um com um peso específico:

* + Género (peso 1)
  + Idade (peso 5)
  + Hipertensão (peso 5)
  + Doença cardíaca (peso 5)
  + Já foi casado (peso 2)
  + Tipo de residência (peso 1)
  + Nível médio de glicose (peso 4)
  + BMI (peso 4)
  + Situação de tabagismo (peso 5)

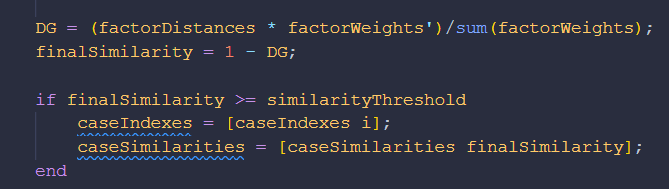
Para cada atributo, a distância entre os valores do paciente na linha *nanCases* e os valores dos pacientes na tabela caseLibrary é calculada.

* + Para atributos numéricos (como idade e nível de glicose), a função *linear\_distance* é utilizada para calcular a diferença absoluta entre os valores.
  + Para o atributo de situação de tabagismo (categórico), a função *smoking\_status\_*distance é aplicada.

A dissimilaridade final é calculada a partir da soma ponderada das distâncias dos atributos e normalizada para um intervalo entre 0 e 1, onde 0 indica total similaridade e 1 indica total diferença.



*Fig2. Cálculo distância de fatores*



*Fig3. Cálculo final similaridades*

**Ajuste do Limite de Similaridade e Imputação do Valor Ausente**

Se nenhuma linha similar for encontrada com base no limite de dissimilaridade inicial (*similarityThreshold*), o algoritmo não para. Ele diminui gradualmente esse limite até encontrar uma linha na tabela *caseLibrary* que seja suficientemente similar à linha em questão na tabela *nanCases*.

Uma vez que essa linha é encontrada, o valor ausente de "stroke" na linha *nanCases* é preenchido usando o modo (valor mais frequente) do atributo "stroke" nas linhas similares. Essa estratégia garante que o valor imputado seja o mais plausível possível, considerando os pacientes mais semelhantes.

A linha preenchida de *nanCases* é então adicionada de volta à tabela principal *caseLibrary*, reunindo todos os dados com valores completos.

**Observações Importantes**

O processo de busca por casos similares e imputação de valores ausentes é iterativo, ou seja, é realizado para cada linha em *nanCases* até que todos os valores ausentes sejam preenchidos.



*Fig4. Iteração sobre casos NaN*

O limite de dissimilaridade (*similarityThreshold*) é um parâmetro crucial que influencia a qualidade dos valores imputados. Um valor muito alto pode levar à imputação de valores incorretos, enquanto um valor muito baixo pode resultar em números excessivos de casos sem similaridade suficiente. A escolha do valor ideal pode depender das características dos dados e do objetivo final da análise.

# Aplicação

Uma imagem com texto, captura de ecrã, ecrã, software

Descrição gerada automaticamente

Figura 1 - Interface gráfica

Como previamente referido, o trabalho inclui também uma interface gráfica, que permite treinar uma rede neuronal podendo escolher alguns dos seus parâmetros de entrada, tais como numero de camadas, uma ou duas, o numero de neurónios por camada, épocas de treinamento, funções de treino, divisão e ativação e também os valores de train, val e test.

Além disto, está também preparada para validar todos os dados inseridos de modo a mostrar mensagens de erro ao utilizador e impedir comportamentos inesperados.

Uma imagem com texto, captura de ecrã, ecrã, software

Descrição gerada automaticamente

Figura 2 - Interface Gráfica - exemplo de uma mensagem de erro

Como podemos observar neste exemplo onde o número de épocas de treinamento não foi especificado e assim que colocamos a rede a treinar nos aparece este pop-up de erro de forma a melhorar a interação do utilizador com o programa.

# Analise de Resultados

## Ficheiro Train:

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

Figura 3 - Estudo número de cadamas escondidas

Com base nesses resultados, podemos concluir que o número e dimensão das camadas escondidas influenciam significativamente o desempenho da rede neural. Em geral, um aumento no número de camadas e neurónios pode melhorar o desempenho da rede até certo ponto, como visto nas Conf1 e Conf2. No entanto, adicionar camadas em excesso, como na Conf4, pode levar a uma degradação no desempenho devido a possíveis problemas de overfitting.

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

Figura 4 - Estudo função de treino

Com base nesses resultados, podemos concluir que a função de treino influencia significativamente o desempenho da rede neural. Algumas funções de treino, como trainbr, podem resultar em melhorias significativas no desempenho em comparação com outras, como traingd. A escolha da função de treino adequada é crucial para obter o melhor desempenho da rede neural em termos de precisão global e precisão no teste.

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

Figura 5 - Estudo função de ativação

Com base nessas observações, podemos concluir que as funções de ativação têm um impacto significativo no desempenho da rede neural. Algumas combinações de funções de ativação, como logsig e purelin na Conf1, podem resultar em desempenho superior, enquanto outras, como tansig e logsig na Conf2, podem levar a resultados muito inferiores. A escolha adequada das funções de ativação é essencial para obter o melhor desempenho da rede neural.

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

Figura 6 - Estudo train, val e test

Com base nessas observações, podemos concluir que a divisão dos exemplos pelos conjuntos influencia significativamente o desempenho da rede neural. Uma distribuição equilibrada pode resultar em melhores resultados, como observado nas Conf3 e Conf6, enquanto distribuições desequilibradas podem levar a um desempenho inferior, como nas Conf1 e Conf2.

## Ficheiro Start:

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

Figura 7 - Estudo função de treino

Com base nessas observações, podemos concluir que a função de treino tem um impacto significativo no desempenho da rede neural. Algumas funções de treino, como traingd e trainbfg, podem levar a um desempenho perfeito do modelo em conjunto com um tempo de execução relativamente baixo. Outras funções, como trainbr, podem resultar em uma precisão total menor e um tempo de execução mais rápido. A escolha adequada da função de treino depende das características do problema e dos requisitos de desempenho.

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

Figura 8 - Estudo função de ativação

Com base nessas observações, podemos concluir que as funções de ativação têm um impacto significativo no desempenho da rede neural. Algumas combinações de funções de ativação, como logsig e purelin na Conf1, podem resultar em desempenho superior, enquanto outras, como tansig e logsig na Conf2, podem levar a resultados muito inferiores. A escolha adequada das funções de ativação é essencial para obter o melhor desempenho da rede neural.

# Conclusão

Com base nos resultados deste trabalho, conclui-se que vários fatores influenciam o desempenho das redes neurais em tarefas de classificação. Desde o número de camadas escondidas até a escolha das funções de ativação e de treino, cada especto desempenha um papel crucial. A distribuição equilibrada dos exemplos entre os conjuntos de treino, validação e teste também é essencial. Essas descobertas fornecem *insights* valiosos sobre como otimizar o desempenho das redes neurais em aplicações de classificação, destacando a importância da seleção criteriosa de parâmetros e configurações.