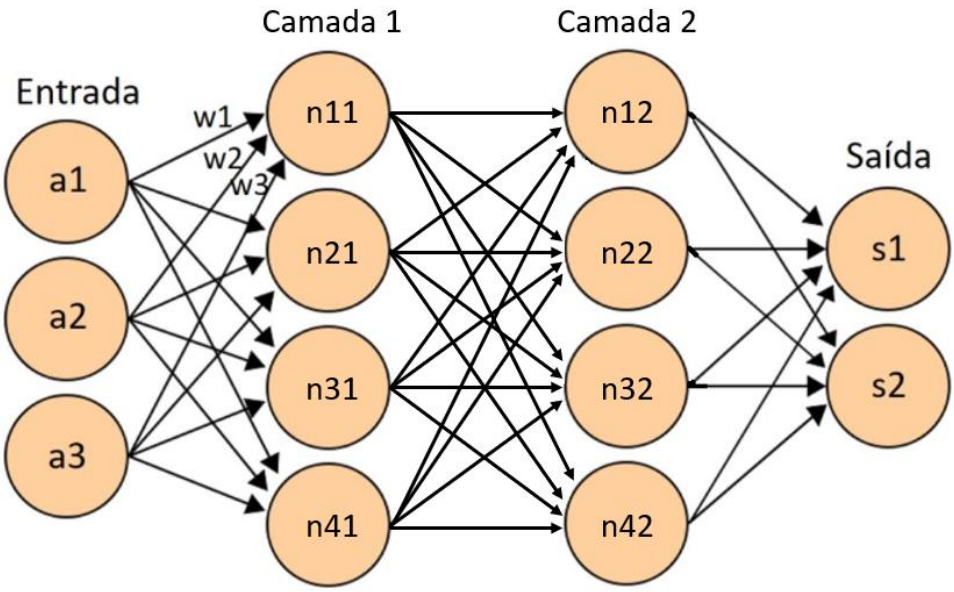
Ano Letivo: 2023/2024

Conhecimento e Raciocínio

Relatório Trabalho Prático



Trabalho realizado por:

Martim Alexandre Vieira Antunes

nº: 2022141890 **Curso:** LEI

Pedro Lino Neves Faneca

nº: 2022134142 **Curso:** LEI

Índice

1.Introdução

Este trabalho foi realizado no âmbito da unidade curricular de Conhecimento e Raciocínio do Instituto Superior de Engenharia durante o ano letivo de 2023/2024.

O objetivo deste trabalho consiste em explorar e aprofundar os conceitos de raciocínio baseado em casos (CBR) e redes neuronais (RNs). Utilizou-se a ferramenta MATLAB, especialmente a toolbox Deep Learning.

2.Escolha do dataset

Nós escolhemos o dataset 2,“**Stroke**”, devido à sua importância significativa na área da saúde pública. Os acidentes vasculares cerebrais (AVCs), representam uma das principais causas de mortalidade e incapacidade em todo o mundo. Este conjunto de dados oferece uma variedade de informações relevantes, como idade, género, histórico médico e estilo de vida, que são cruciais para compreender e prever os fatores de risco associados aos AVCs. Além disso, a classificação numérica presente neste conjunto de dados, onde o objetivo é determinar se um indivíduo teve ou não um AVC, proporciona uma abordagem clara e mais acessível que os outros datasets disponíveis. A sua análise pode oferecer perceções valiosas para a prevenção precoce, diagnóstico e tratamento eficaz dos AVCs.

Este dataset é constituído por 3 ficheiros. O ficheiro Start é constituído por 10 linhas (10 casos), o ficheiro Train é constituído por 620 linhas e o Test é constituído por 10 linhas também.

O dataset “Stroke” tem 11 atributos (números e texto), entre eles um id, em que mais à frente irá ser retirado para o estudo das redes pois não afeta os resultados, o género (masculino ou feminino),idade, hipertensão (sim ou não), problema de coração (sim ou não), relação conjugal (sim ou não), tipo de residência (urbana ou rural), índice de glucose, índice de massa corporal, condição de fumador e por último o nosso target (sim ou não), ou seja, se a pessoa é provável a ter AVC ou não.

3.Preparação do dataset

Na primeira parte da preparação do ficheiro **Train**, foi necessário converter todos os atributos em valores numéricos ou booleanos. Isso envolveu analisar os tipos de dados dos atributos do conjunto de dados do ficheiro START e o ficheiro disponibilizado sobre as informações de cada dataset e efetuar as alterações necessárias para garantir que todos os atributos estivessem num formato adequado para análise posterior. Este processo foi realizado utilizando um script em Matlab para realizar as conversões necessárias, conforme descrito nas instruções do projeto. Esta etapa é crucial para assegurar que os dados estejam prontos para serem utilizados na análise e modelagem subsequentes.

3.1.Script Matlab

```
dados_train = readtable('Train.csv');

% Mapeamento dos valores para os atributos
atributo_smoking = containers.Map({'never smoked','formerly smoked','smokes','Unknown'},{0,1,2,3});
atributo_genero = containers.Map({'Male','Female'},{0,1});
atributo_residencia = containers.Map({'Rural','Urban'},{0,1});
atributo_casamento = containers.Map({'No','Yes'},{0,1});

% Converter valores para os atributos 'Smoking_status', 'Gender' e 'Residence_type' e 'ever_married'
dados_train.smoking_status = cellfun(@(x) atributo_smoking(x),dados_train.smoking_status);
dados_train.gender = cellfun(@(x) atributo_genero(x),dados_train.gender);
dados_train.Residence_type = cellfun(@(x) atributo_residencia(x),dados_train.Residence_type);
dados_train.ever_married = cellfun(@(x) atributo_casamento(x),dados_train.ever_married);


% Salvar o novo dados_trainset
writetable(dados_train, 'train_processado.csv');
```

Este script em MATLAB prepara os dados do conjunto (**dados_train**) para serem utilizados. Aqui está um resumo do que o script faz:

1. Define mapas de valores para os atributos categóricos (“smoking_status”, “gender”, “Residence_type” e “ever_married”) utilizando objetos “**containers.Map”**. Cada mapa associa os valores originais dos atributos a valores numéricos específicos.

2. Aplica os mapas aos dados no conjunto para converter os valores dos atributos das suas representações de string para valores numéricos. Isto é feito utilizando a função “**cellfun”** em conjunto com funções anónimas que utilizam os mapas definidos anteriormente.

3. Após a execução do script, os dados no conjunto estão prontos para serem utilizados em algoritmos de aprendizagem automática, uma vez que todos os atributos foram convertidos em valores numéricos.

3.2.Sistema de raciocínio baseado em casos (CBR)

Foi nos pedido para implementar um sistema de raciocínio baseado em casos para preencher os atributos com valores em falta (NA), mas apenas a fase de Retrieve para encontrar o caso mais semelhante àquele onde se pretende o preenchimento e usar os valores desse caso para preencher os valores em falta.

Na fase de **Retrieve**, são usadas funções de similaridade para calcular a proximidade entre o caso incompleto e os casos da base de casos existentes. As funções de similaridade são essenciais para determinar quão semelhantes dois casos são em termos dos atributos relevantes. Existem duas abordagens principais para calcular a similaridade: local e global.

Como já tínhamos os dados todos numéricos, usámos como função de similaridade local a função linear, "**calculate_linear_distance",** esta calcula a distância linear entre os valores de dois atributos, simplesmente calcula a diferença absoluta entre os valores dos atributos.

```
function [res] = calculate_linear_distance(val1, val2)

    res = abs(val1 - val2);
end
```

No exemplo do atributo "**age**", a função calcula a distância linear entre o valor do atributo "age" no caso atual da biblioteca de casos e o valor correspondente no novo caso. Isso é feito dividindo o valor do atributo "age" no caso atual pelo valor máximo observado para o atributo "age" em toda a biblioteca de casos, e fazendo o mesmo para o valor do atributo "age" no novo caso. Em seguida, a diferença absoluta entre esses valores normalizados é calculada.

```
 if isfield(new_case,'age')
distances(1,2) = calculate_linear_distance(case_library{i,'age'} / max_values('age'), ...
                        new_case.age / max_values('age'));
 end
```

Repetimos este processo para cada um dos nove atributos que queremos estudar, calculando a distância linear entre o valor do atributo atual na biblioteca de casos e o valor correspondente no novo caso. No entanto, devemos excluir os atributos "id" e "stroke", uma vez que o "id" é apenas um identificador único para cada caso e "stroke" é o nosso atributo alvo, ou seja, aquele que queremos prever. Assim, concentramo-nos nos outros nove atributos para determinar a semelhança entre casos.

Após calcularmos as distâncias ponderadas para cada atributo, utilizamos a fórmula apresentada para calcular a similaridade global entre o novo caso e cada caso na biblioteca de casos. A fórmula começa por multiplicar as distâncias pelo vetor de pesos "**weighting_factors"**. Em seguida, somamos os resultados dessas multiplicações e dividimos pelo somatório de todos os pesos. Este processo produz uma medida global de similaridade, representada pela variável "**DG"**.

Após calcular o valor de **DG**, subtraímos este valor de 1 para obter a similaridade final entre o novo caso e cada caso na biblioteca de casos. Essa similaridade final é atribuída à variável "**final_similarity"**.

Essencialmente, este processo de cálculo de similaridade global leva em consideração as distâncias ponderadas entre os atributos e os pesos atribuídos a cada um desses atributos, resultando numa medida global de similaridade entre o novo caso e os casos na biblioteca de casos.

```
DG =  (distances * weighting_factors') /sum(weighting_factors);
final_similarity = 1- DG;
```

Vamos agora falar do vetor de pesos. Aqui está o nosso:

```
weighting_factors = [1 2 3 4 1 2 4 3 4];
```

Nós escolhemos estes valores para o nosso vetor de pesos, com base na importância relativa de cada atributo para o problema em questão, na nossa opinião. Aqui está a justificação para esses pesos:

- **gender (género)**: Peso de 1 - O género pode não ter uma influência tão significativa na ocorrência de um AVC, mas ainda assim é um fator relevante a ser considerado.
- **age (idade)**: Peso de 2 - A idade é um fator crítico na ocorrência de AVCs, com os riscos a aumentar significativamente com o envelhecimento. Portanto, merece uma ponderação maior.
- **hypertension (hipertensão)**: Peso de 3 - A hipertensão é um dos principais fatores de risco para AVC. A sua presença ou ausência pode ter um impacto significativo na previsão de um AVC.
- **heart_disease (doença cardíaca)**: Peso de 4 - Assim como a hipertensão, a presença de doença cardíaca é um fator de risco significativo para AVC e, portanto, merece uma ponderação mais alta.

- **ever_married (estado civil)**: Peso de 1 - Embora o estado civil possa ter alguma influência nos hábitos de vida e, consequentemente, no risco de AVC, a sua importância é menor em comparação com outros fatores como idade e condições médicas pré-existentes.
- **Residence_type (tipo de residência)**: Peso de 2 - O tipo de residência pode estar associado a diferentes estilos de vida e acesso a cuidados de saúde, mas o seu impacto direto na ocorrência de AVC pode ser menor em comparação com outros fatores.
- **avg_glucose_level (nível médio de glicose)**: Peso de 4 - A diabetes é um fator de risco bem conhecido para AVC, e o nível médio de glicose pode ser um indicador importante desse risco.
- **bmi (índice de massa corporal)**: Peso de 4 - O índice de massa corporal está relacionado à saúde cardiovascular e pode influenciar o risco de AVC.
- **smoking_status (estado de tabagismo)**: Ponderação de 4 - O tabagismo é um dos principais fatores de risco de AVC. A sua presença ou ausência pode ter um impacto significativo na previsão de um AVC.

4.Estudo e análise de redes neuronais

Agora iremos estudar, treinar e analisar redes neuronais *feedforward* para as tarefas de classificação, para cada ficheiro do dataset.

4.1.Ficheiro Start

Para este ficheiro, foi nos pedido que utilizássemos uma rede neuronal *feedforward* com **uma camada de 10 neurónios**, que para já as **funções de treino e ativação fossem as defaults** e que usássemos **todos os exemplos no treino**.

```
function startFile()

tempoExecucao = tic;
data = readmatrix('Start.csv','Delimiter',';','DecimalSeparator','.');

in =data(:,2:end-1)';
t=data(:,end)';
iteracoes=50;
precisaoTotal = [];

for k=1:iteracoes
    fprintf('Iteracao %d\n',k);

    net=feedforwardnet(10);   %Criar rede

    net.trainParam.showWindow=0;

    net.divideFcn = ''; % TODOS OS EXEMPLOS DE INPUT SAO USADOS NO TREINO

    net=train(net,in,t);  %Treina a rede

    out = sim(net, in);  %Simula rede /testa

    out  = (out >= 0.5);


    erro = perform(net,t,out);  %desempenho rede

     r = sum(out == t);

    precisao = r/size(out,2) *100;

    fprintf('Precisao total: %f\n', precisao);


    precisaoTotal = [precisaoTotal precisao];

end

precisaoTotal = mean(precisaoTotal);
fprintf("\nMedia Precisão Total: %.2f%\n",precisaoTotal);
fprintf("Erro: %f\n",erro);
fprintf('Tempo de execução: %.2f segundos\n',toc(tempoExecucao));

end
```

Obtivemos os seguintes resultados de tempo e métricas de desempenho (média precisão total, erro e tempo).

	Número de camadas escondidas	Número de neurónios	Funções de ativação	Função de treino	Divisão dos exemplos		Média Precisão Total	Erro	Tempo (50 iterações)
Configuração por defeito	1	10	tansig, purelin	trainlm	Sem segmentação		100	0	6,58

Podemos concluir que a rede alcançou uma precisão total de 100%, com erro 0, em todas as repetições, e apresentou sempre um tempo de execução muito bom.

Depois decidimos **alterar as funções de treino e ativação** para verificar se os resultados continuariam iguais ou se iriam ser diferentes. E obtivemos estes resultados.

As funções de ativação influenciam o desempenho?									
Conf1	1	10	logsig, purelin	trainlm	Sem segmentação		100	0	7,67
Conf2	1	10	softmax, tansig	trainlm	Sem segmentação		100	0	7,7
Conf3	1	10	hardlims,tansig	trainlm	Sem segmentação		92,4	0,076	6,47
Conf4	1	10	tansig,purelin	trainlm	Sem segmentação		100	0	5,62
Conf5	1	10	tansig,logsig	trainlm	Sem segmentação		50	0,5	6,45

Dos testes realizados com as diferentes funções de ativação, podemos verificar que existem duas delas, que não dão 100% de precisão total entre elas a {hardlims e tansig} que obteve uma média de 92,4% e a {tansig,logsig} que obteve uma média de precisão total de 50%.

Pode ser justificado pelo facto, de no primeiro caso a combinação não ser adequada, enquanto que no segundo caso que resultou numa média fraca, pode ser devido à inadequação das funções de ativação para o problema específico, ou pode indicar que o problema é intrinsecamente complexo e exige mais camadas ou neurónios para ser bem modelado.

A função de treino influencia o desempenho?									
Conf1	1	10	tansig, purelin	traingd	Sem segmentação		100	0	29,72
Conf2	1	10	tansig, purelin	trainbfg	Sem segmentação		100	0	10,72
Conf3	1	10	tansig, purelin	trainc	Sem segmentação		100	0	524,65
Conf4	1	10	tansig, purelin	trains	Sem segmentação		100	0	84,57
Conf5	1	10	tansig, purelin	trainbr	Sem segmentação		73,6	0,264	5,89

Por esta tabela, conclui-mos que, só existe uma função de treino que os valores de precisão total e erro não permanecem constantes, que é a trainbr. De resto, as únicas diferenças observadas estão nos tempos de execução e na diferença significativa no tempo de execução ao usar a função de treino (trainc).

4.2.Ficheiro Train Preparado

Nesta parte do trabalho, utilizámos o ficheiro **Train** que preparámos na secção 3. O objetivo é criar redes neuronais feedforward para estudar a influência que o número e a dimensão das camadas escondidas, as diferentes funções de treino, as diferentes funções de ativação e as diferentes segmentações têm no desempenho da rede (precisão global e precisão de teste).

Abaixo segue-se o protótipo da função que é usada para resolver esta parte do trabalho.

```
function [precisaoGlobal, precisaoTeste, tempo] = trainFeedForward(neuroCamadas,FTreino,FAtivacao,divideF,trainRatio,valRatio,testRatio,savename)
```

Em que queremos como output, a **precisão Global**, **precisão Teste** e o **tempo de execução**.

Cada parâmetro da função representa o seguinte:

- **neuroCamadas:** É um vetor que especifica o número de neurónios em cada camada oculta da rede neuronal, incluindo a camada de entrada e a camada de saída.
- **FTreino:** É a função de treino utilizada para treinar a rede neuronal.
- **FAtivação:** É um vetor de strings que especifica a função de ativação para cada camada da rede neuronal, incluindo a camada de saída.
- **divideF:** É a função de segmentação usada para dividir os dados em conjuntos de treino, validação e teste.
- **trainRatio:** É a proporção dos dados utilizados para treinar a rede neuronal.
- **valRatio:** É a proporção dos dados utilizados para validação durante o treino da rede neuronal.
- **testRatio:** É a proporção dos dados utilizados para teste da rede neuronal.
- **savename:** É o nome usado para guardar as informações e resultados da rede neuronal, como precisão global e precisão de teste.

Passando os diferentes parâmetros que queremos testar como argumentos da função, torna-se mais fácil e prático alterar os parâmetros para realizar testes e experimentar diferentes configurações da rede neuronal. Essa abordagem permite uma maior flexibilidade e agilidade na realização de testes, facilitando a comparação de resultados e a otimização do desempenho da rede para diferentes cenários e requisitos específicos.

Primeiro verificamos se o número de funções de ativação especificadas é igual ao número de camadas ocultas mais uma (camada de saída) e se a soma dos três valores de divisão (treino, validação e teste) é igual a 1, o que é uma condição comum para garantir que toda a amostra de dados seja totalmente utilizada e sem sobreposição.

Se alguma destas verificações falhar, é exibida uma mensagem de aviso e a função retorna, interrompendo o processo de treinar a rede.

```
if length(FAtivacao) ~= (length(neuroCamadas)+1)
    disp('O FAtivacao tem de ter o mesmo tamanho que o numero de camadas da rede + 1');
    return;
end


if((trainRatio + valRatio + testRatio) ~= 1) && ~isempty(divideF)
    disp("Ratios somados tem de dar 1");
    return;
end
```

Após isso, como no enunciado nos pedem para guardar as três redes com melhores desempenhos, decidimos ir guardando sempre a melhor rede em cada treino. Por isso criá-mos duas pastas, uma onde vamos colocando a melhor rede com precisão global e a outra com a melhor precisão de teste.

```
if ~isfolder('./Redes')
    mkdir('./Redes')
end

if ~isfolder('./Redes/RedesTeste')
    mkdir('./Redes/RedesTeste')
end

if ~isfolder('./Redes/RedesGlobal')
    mkdir('./Redes/RedesGlobal')
end
```

Depois dividimos o dataset em variáveis de entrada e de target, retirando o atributo id do estudo pois não interfere em nada com os resultados.

```
data = readmatrix('Train_Completo.csv','Delimiter',',','DecimalSeparator','.');
in =data(:,2:end-1)';
t=data(:,end)';
```

A seguir iniciamos a contagem do tempo de execução do treino da rede neuronal utilizando a função “tic”, inicializamos as variáveis “precisaoGlobal” e “precisaoTeste” como vetores vazios, onde serão armazenadas as precisões global e de teste em cada iteração do treino.

O número de iterações do treino é definido como 50 na variável “iteracoes”.

As variáveis “MelhorTeste”,”MelhorIndice” e “MelhorGlobal” são inicializadas como 0. “MelhorTeste” e “MelhorGlobal” serão usadas para rastrear a melhor precisão de teste e a melhor precisão global, respetivamente. A variável “MelhorIndice” será usado para registrar o índice da melhor precisão.

```
tempoExecucao = tic;
precisaoGlobal = [];
precisaoTeste = [];
iteracoes = 50;

MelhorTeste = 0;
MelhorIndice = 0;
MelhorGlobal = 0;
```

Após isto, entramos num loop for que itera “iterações” vezes, como definido anteriormente. Dentro do loop, a cada iteração, é apresentada uma mensagem indicando o número da iteração atual.

Depois, criamos a rede neuronal utilizando a função feedforwardnet, especificando o número de neurónios em cada camada oculta com base no vetor “neuroCamadas”.

Configuramos os parâmetros de treino da rede, como a função de treino (“FTreino”), a função de divisão dos dados (“divideF”), e as proporções de divisão dos dados (trainRatio, valRatio e testRatio). Se a função de divisão não for especificada (“isempty(divideF”)), desativamos a divisão dos dados.

Em seguida, percorremos cada camada da rede e atribuímos a função de ativação correspondente, definida no vetor “FAtivacao”.

Posteriormente, treinamos a rede utilizando os dados de entrada (“in”) e os alvos (“t”) com a função “train”. Depois do treino, simulamos a rede utilizando os mesmos dados de entrada (“in”) para obter a saída prevista.

Calculamos a precisão total da rede, comparando as saídas previstas com os alvos. Armazenamos esta precisão no vetor “precisaoGlobal”. Se a precisão total obtida for maior que a melhor precisão global registada até ao momento (“MelhorGlobal”), atualizamos a melhor precisão global e guardamos a rede atual como a melhor rede global.

Seguidamente, simulamos a rede apenas no conjunto de teste e calculamos a precisão do teste. Guardamos esta precisão no vetor “precisaoTeste”. Se a precisão do teste for maior que a melhor precisão do teste registada até ao momento (“MelhorTeste”), atualizamos a melhor precisão do teste e guardamos a rede atual como a melhor rede de teste.

Após as iterações, calculamos a média das precisões global e do teste, exibimos os resultados no ecrã e também guardamos a média das precisões em “precisaoGlobal” e “precisaoTeste”. Por fim, exibimos a melhor precisão global e a melhor precisão do teste, juntamente com o tempo total de execução do treino.

```
for k = 1:iteracoes
    fprintf('Iteracao %d\n',k);

    net = feedforwardnet(neuroCamadas);   %CRIAR

    % COMPLETAR A RESTANTE CONFIGURACAO(FUNCAO TREINO,ATIVACAO,SEGMENTACAO)
    net.trainFcn=FTreino;


    if(~isempty(divideF))
        net.divideFcn = divideF;
        net.divideParam.trainRatio = trainRatio;
        net.divideParam.valRatio = valRatio;
        net.divideParam.testRatio = testRatio;
     else
        net.divideFcn = '';
    end

    net.trainParam.showWindow=0;


    for i = 1:length(neuroCamadas) + 1
        net.layers{i}.transferFcn = FAtivacao{i};
    end
```

```
[net,tr] = train(net, in, t);  %TREINAR

%view(net);
%disp(tr);

out = sim(net, in);     %SIMULAR
%VISUALIZAR DESEMPENHO
%plotconfusion(t, out) % Matriz de confusao
%plotperf(tr)         % Grafico com o desempenho da rede nos 3 conjuntos

out  = (out >= 0.5);

erro = perform(net,t,out);  %desempenho rede
%fprintf('Erro na classificação dos 620 exemplos %f\n', erro)

precisao=0;

r = sum(out == t);

precisao = r/size(out,2) *100;

fprintf('Precisao total: %f\n', precisao);

precisaoGlobal = [precisaoGlobal precisao]; %armazenar no vetor cada valor

if(precisao > MelhorGlobal)
    MelhorGlobal = precisao;
    save(['./Redes/RedesGlobal/' savename '.mat'], 'net');
end
```

O primeiro teste que fizemos foi uma configuração por defeito, ou seja, não mexer nos parâmetros e obtivemos o seguinte resultado.

	Número de camadas escondidas	Número de neurónios	Funções de ativação	Função de treino	Divisão dos exemplos	Média Precisão Global	Média Precisão Teste	Tempo (50 iterações)	Melhor Rede (Precisão Global)	Melhor Rede (Precisão Teste)
Configuração por defeito	1	10	tansig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}	76,66	73,41	6,48	80,33	81,52

Nos próximos testes, fomos começando a alterar os valores de cada parâmetro e a registar os seus resultados.

Primeiro, fomos perceber a influência no desempenho da rede com base no **número e dimensão das camadas escondidas** e obtivemos os seguintes resultados.

O número e dimensão das camadas encondidas influencia o desempenho?										
Conf1	2	5, 5	tansig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}	76,24	72,87	7,56	79,67	84,78
Conf2	2	10,10	tansig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}	77,02	72,09	8,35	80,82	81,52
Conf3	2	20,2	tansig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}	78,63	71,76	18,52	85,57	81,52
Conf4	3	5,5,5	tansig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}	75,92	72,61	8,46	80,33	84,78
Conf5	4	10,10,10,10	tansig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}	76,88	70,04	13,32	81,97	81,52

Com base nestes resultados, podemos concluir que **quanto mais neurónios temos por camada**, melhores resultados temos. Por exemplo, na Conf3, 2 camadas com 20 neurónios cada, obtivemos uma melhor média de precisão global e teste que a Confg5 (4 camadas com 10 neurónios cada), ou seja, não é tanto o número de camadas que importa, mas sim o número de neurónios em cada camada.

A seguir, fomos estudar a influência da função de treino no desempenho da rede. Utilizámos várias e obtivemos o seguinte quadro.

A função de treino influencia o desempenho?										
Conf1	1	10	tansig, purelin	traingd	dividerand = {0.7, 0.15, 0.15}	70,81	69,46	34,81	76,06	82,61
Conf2	1	10	tansig, purelin	trainbfg	dividerand = {0.7, 0.15, 0.15}	75,11	73,44	8,63	77,87	90,22
Conf3	1	10	tansig, purelin	traincgb	dividerand = {0.7, 0.15, 0.15}	75,43	73,85	6,62	78,69	83,7
Conf4	1	10	tansig, purelin	trainr	dividerand = {0.7, 0.15, 0.15}	75,68	72,76	191,92	78,2	83,7
Conf5	1	10	tansig, purelin	trains	dividerand = {0.7, 0.15, 0.15}	73,08	73,46	80,99	76,72	83,7

Podemos concluir, que das várias funções de treino, as que obtiveram os melhores resultados foram a “traincgb” e a ” trainbfg”. A função “trainr” também obteve bons resultados mas teve um tempo de execução bastante elevado, mas a função default, “trainlm”, teve melhores valores melhores de precisão.

Houve uma função que também testamos mas que não colocámos aqui, por motivos de não conseguir obter os resultados em pouco tempo, que foi a função “trainc”, uma função que demora muito, mas muito tempo a obter os resultados.

No entanto, é importante notar que a função padrão, "trainlm", apresentou valores de precisão melhores em comparação com as outras funções.

Em seguida, fomos verificar a influência das funções de ativação e obtivemos os seguintes resultados.

As funções de ativação influenciam o desempenho?										
Conf1	1	10	logsig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}	76,15	73,3	5,38	80,33	82,61
Conf2	1	10	tansig, logsig	trainlm	dividerand = {0.7, 0.15, 0.15}	39,18	39	7,99	39,18	51,09
Conf3	1	10	softmax, tansig	trainlm	dividerand = {0.7, 0.15, 0.15}	76,48	72,33	8,01	83,77	80,43
Conf4	1	10	logsig,hardlims	trainlm	dividerand = {0.7, 0.15, 0.15}	52,46	53,13	5,84	66,39	72,83
Conf5	1	10	purelin,satlin	trainlm	dividerand = {0.7, 0.15, 0.15}	39,18	40,24	7,69	39,18	54,35

Com base nesta tabela, podemos concluir que obtivemos resultados menos satisfatórios ao utilizar as combinações de funções de ativação {"tansig" e "logsig"} e {"purelin" e "satlin"}. Mais uma vez, confirmamos que as funções padrão {"tansig", "purelin"} produziram melhores resultados.

Poderá ser, porque por exemplo a função "logsig" tem uma saturação pronunciada em valores extremos, o que pode dificultar o treino da rede. Da mesma forma, a função "satlin" também tem uma propriedade de saturação, que pode limitar a capacidade da rede de aprender relações complexas nos dados.

Assim, ao utilizar as funções "tansig" e "purelin", que são mais suaves e lineares, é mais provável que a rede neuronal seja capaz de aprender com eficácia os padrões nos dados, resultando em melhores desempenhos de precisão.

Agora, fomos verificar a influência que tem a divisão de exemplos pelos conjuntos no desempenho da rede.

A divisão de exemplos pelos conjuntos influencia o desempenho?

Conf1	1	10	tansig, purelin	trainlm	dividerand = {0.33, 0.33, 0.33}	74	70,84	7,83	78,02	77,83
Conf2	1	10	tansig, purelin	trainlm	dividerand = {0.9, 0.05, 0.05}	76,45	73,35	8,55	85,25	87,07
Conf3	1	10	tansig, purelin	trainlm	dividerand = {0.05, 0.05 ,0.9}	64,09	62,92	7,39	73,93	72,31
Conf4	1	10	tansig, purelin	trainlm	dividerand = {0.5, 0 ,0.5}	80,34	66,12	208,9	82,46	71,15
Conf5	1	10	tansig, purelin	trainlm	dividerand = {0.5, 0.2 ,0.3}	76,14	72,17	9,58	79,34	77,6

Podemos ver que, uma divisão por igual dos parâmetros de segmentação (Conf1) tem um desempenho satisfatório. No entanto, quando o tamanho do conjunto de treino é muito superior aos dos conjuntos de validação e teste (Conf2), obtemos uma precisão global bastante satisfatória, por outro lado, se o valor do conjunto de teste é muito superior comparativamente aos restantes (Conf3), o desempenho da rede é prejudicado.

Ao realizar o teste de colocar o conjunto de validação a 0 (Conf4),notamos que obtemos os melhores resultados de média de precisão global mas por consequência temos um aumento significativo no tempo de execução.

Por último, após identificar os melhores parâmetros em termos do número de camadas, funções de treino, funções de ativação e divisões de conjunto de dados, decidimos combinar esses parâmetros otimizados para ver como influenciariam o desempenho da rede neuronal.

E se combinásse-mos os melhores parâmetros?									
2	20,2	tansig, purelin	trainlm	dividerand = {0.5,0,0.5}		80,65	61,29	213,04	84,43	68,85

E obtivemos o resultado esperado: a melhor média de precisão global.

4.3.Ficheiro Test

Nesta parte do trabalho, pretende-se analisar a capacidade de generalização e aprendizagem das melhores redes gravadas anteriormente.

As melhores redes obtidas foram a rede17, a rede3 e a rede13. Por isso, fomos testar com o ficheiro “Test” que tinha 10 casos não treinados, e com isso verificar os acertos treinando, com as melhores redes obtidas no ficheiro “Train”

```
function testFile()

data = readmatrix('Test.csv','Delimiter',';','DecimalSeparator','.')
in =data(:,2:end-1)';
t=data(:,end)';
tempoExecucao = tic;

%Rede com melhor desempenho- rede17
rede1 = load('Redes\RedesGlobal\rede17.mat');
rede2 = load('Redes\RedesGlobal\rede3.mat');
rede3 = load('Redes\RedesGlobal\rede13.mat');

% Extrair as redes dos arquivos carregados
net = rede3.net;


% Simular a rede global nos dados de teste
out = sim(net, in);

out = (out >= 0.5);

erro = perform(net,t,out);  %desempenho rede

precisaoTotal = 0;

%Calcular precisao Total
r = sum(out == t);
precisaoTotal = r/size(out,2) *100;

fprintf('\nPrecisão Total: %f\n',precisaoTotal);
fprintf('Erro: %f\n',erro);
fprintf('Tempo de execução: %.2f segundos\n',toc(tempoExecucao));
```

Obtivemos os resultados seguintes.

	Número de camadas escondidas	Número de neurónios	Funções de ativação	Função de treino	Divisão dos exemplos		Precisão Total	Erro	Tempo (50 iterações)
Melhores Redes									
Rede 17	1	10	tansig,purelin	trainlm	dividerand = {0.9, 0.05, 0.05}		60	0,4	0,08
Rede 3	2	20,2	tansig,purelin	trainlm	dividerand = {0.7, 0.15, 0.15}		60	0,4	0,08
Rede 13	1	10	softmax, tansig	trainlm	dividerand = {0.7, 0.15, 0.15}		70	0,3	0,04

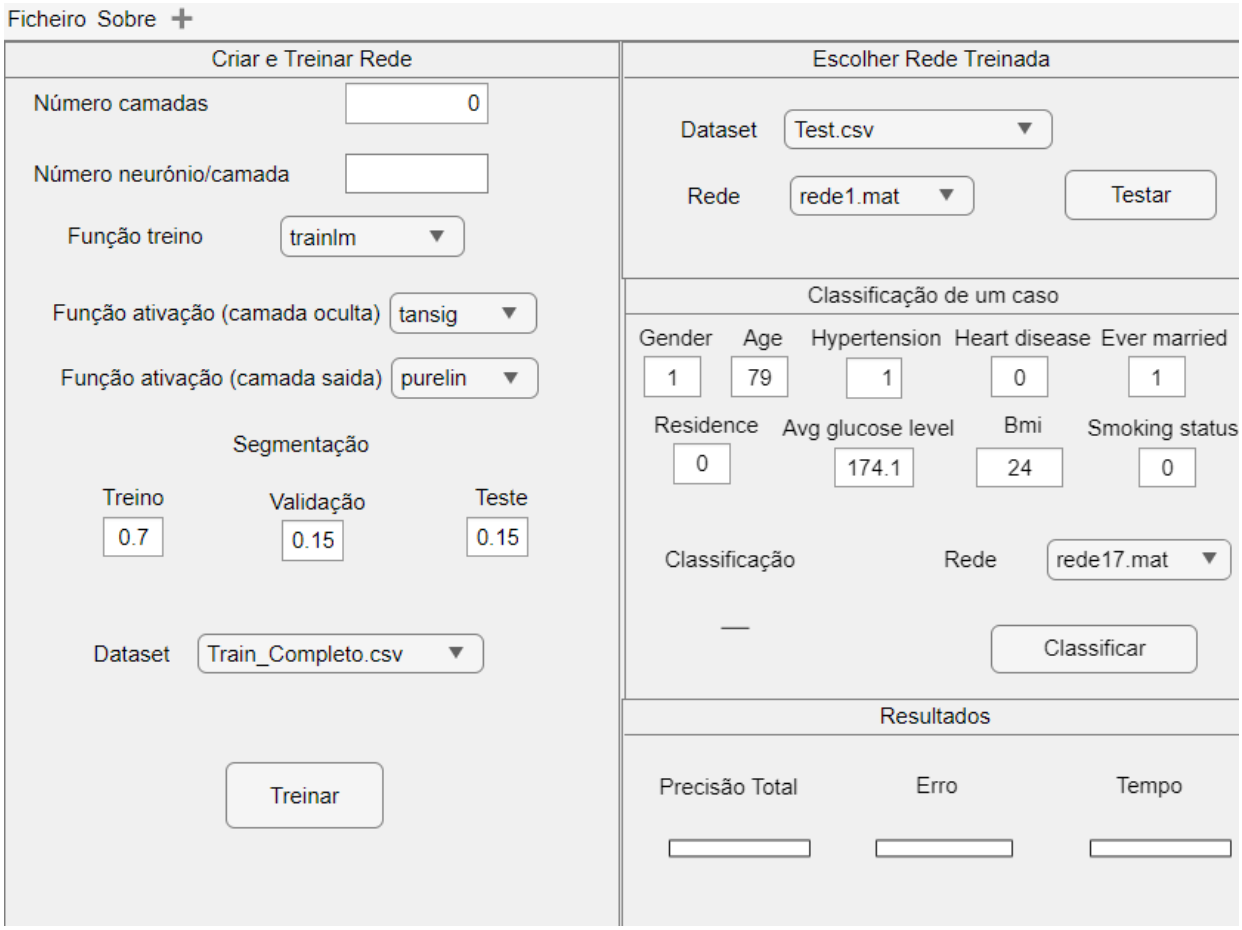
Podemos concluir que, para a primeira rede que para nós, foi a melhor rede obtida anteriormente, esta teve uma precisão total de 60%, ou seja, a rede acertou 6 em 10, o mesmo aconteceu com a segunda rede. Já a terceira rede conseguiu acertar 7 em 10 casos de teste, o que foi bastante positivo. Além disso, ela conseguiu alcançar um tempo de execução menor em comparação com as redes anteriores.

5.Aplicação gráfica

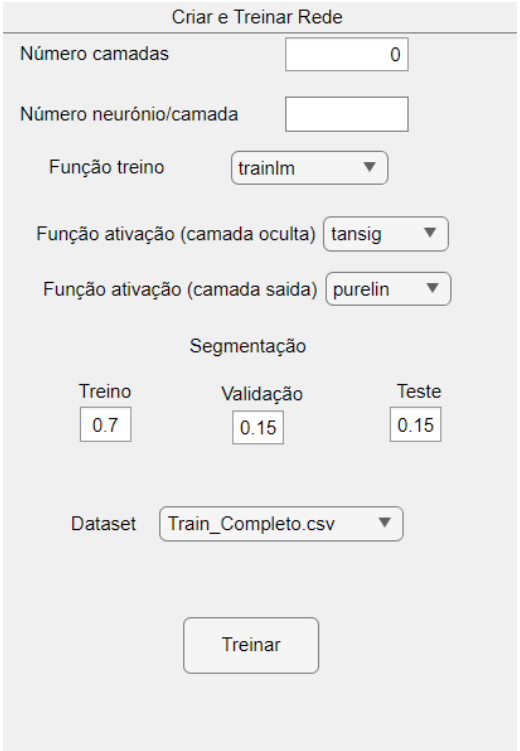
Foi-nos proposto desenvolver uma aplicação gráfica simples em MATLAB, de carácter opcional, e nós aceitámos o desafio. Esta app tinha de oferecer as seguintes funcionalidades:

- Criar uma rede neuronal, permitindo a configuração do número de camadas, números de neurónios por camada, funções de ativação, funções de treino, rácios de segmentação.
- Treinar a rede neuronal criada após a escolha de um dataset.
- Carregar uma rede neuronal previamente treinada e gravada em disco (obtidas na alínea 4.2) e usar a RN para um dataset, mostrando os desempenhos obtidos.
- Permitir introduzir os valores dos atributos de apenas um caso e verificar qual a classificação dada pela rede. Para testar e verificar se a rede acerta ou falha, pode usar os casos do ficheiro TEST.

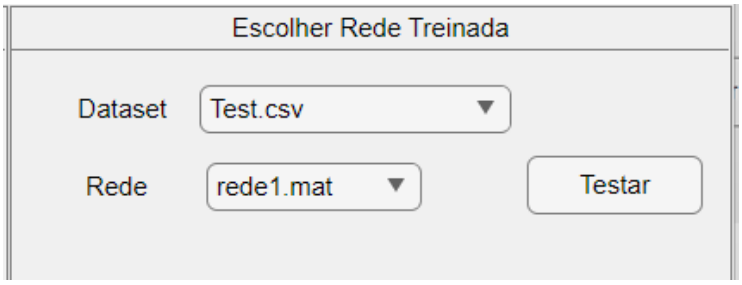
Na imagem abaixo, encontra-se um esboço da nossa interface gráfica.



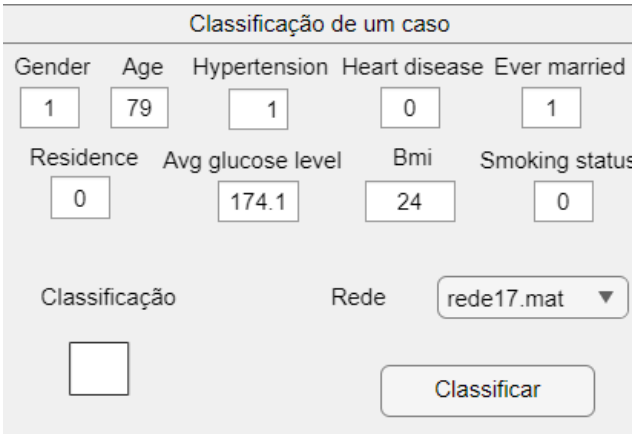
À esquerda é onde o utilizador, pode criar uma rede neuronal com os valores dos parâmetros que quiser, desde o número de camadas até aos valores de segmentação, após isso pode treiná-la e verificar o seu desemprenho.



À direita em cima, o utilizador pode escolher uma rede já treinada, escolher o dataset e verificar a precisão total, o erro e o tempo de execução.



Também temos na nossa aplicação, a funcionalidade de permitir introduzir os valores dos atributos de apenas um caso e verificar qual a classificação dada pela rede.



Podemos concluir que a nossa aplicação cumpriu com sucesso todas as funcionalidades propostas, oferecendo uma plataforma intuitiva e eficaz para trabalhar com redes neuronais. Os principais pontos a serem destacados são:

1. Facilidade de Utilização:
    - A interface da aplicação foi desenhada de forma intuitiva, permitindo que os utilizadores configurem facilmente os parâmetros da rede neuronal, escolham conjuntos de dados e avaliem o desempenho da rede.
2. Flexibilidade na Configuração da Rede:
    - Os utilizadores têm flexibilidade para definir o número de camadas, o número de neurónios em cada camada e as funções de ativação, permitindo a criação de redes neuronais adaptadas a diferentes problemas e conjuntos de dados.
3. Avaliação Abrangente:
    - A aplicação oferece uma avaliação abrangente do desempenho da rede neuronal, incluindo precisão total, erro médio e tempo de execução. Isso fornece insights valiosos sobre a eficácia da rede em lidar com os dados de entrada.
4. Classificação de Casos Individuais:
    - A capacidade de classificar casos individuais permite uma análise detalhada do comportamento da rede neuronal em casos específicos, o que é útil para compreender como a rede está a tomar decisões e para identificar possíveis áreas de melhoria.

No geral, a nossa aplicação oferece uma solução completa e robusta para trabalhar com redes neuronais, ajudando os utilizadores a criar, treinar, avaliar e aplicar eficazmente modelos de aprendizagem em uma variedade de cenários e problemas.

6.Conclusão

Este trabalho explorou várias configurações de redes neuronais feedforward para classificação de dados. Investigámos o impacto do número de camadas e neurónios, testámos diferentes funções de treino e ativação, e analisámos a divisão dos dados de treino, validação e teste. Salientamos a importância da seleção cuidadosa desses parâmetros para melhorar o desempenho da rede.

Testámos a capacidade de generalização das melhores redes obtidas, observando como se comportam ao classificar novos dados. Estes testes são cruciais para avaliar a eficácia das redes em situações do mundo real.

Adicionalmente, desenvolvemos uma aplicação MATLAB para facilitar a experimentação com diferentes configurações de rede. Esta ferramenta proporcionou uma abordagem prática e interativa na avaliação do desempenho das redes.

Em suma, este trabalho forneceu insights valiosos sobre a configuração e treino de redes neuronais para tarefas de classificação.