CONHECIMENTO E RACIOCINIO

Engenharia Informática

Alexandre Ferreira
2021138219

2023/2024

DATASET

O Dataset utilizado foi o Dataset2-Stroke composto por 3 ficheiros:

- Start.csv
- Train.csv
- Test.csv

Atributos presentes no dataset:

- Gender
  - 0 – male;
  - 1 – female;
- Age
- Hypertension
  - 0 – false;
  - 1 – true;
- Heart_disease
  - 0 – false;
  - 1 – true;
- Ever_married
  - 0 – false
  - 1 – true;
- Residence_type
  - 0 – Rural;
  - 1 – Urban;
- Avg_glucose_level
- Bmi
- Smoking_status
  - 0 - never smoked;
  - 1 – formerly smoked;
  - 2 – smokes;
  - 3 – unknown;

Stroke(target)

- 0 – false;
- 1 – true;

Linhas por ficheiro:

- Train.csv: 610 linhas
- Test.csv: 10 linhas
- Start.csv: 10 linhas

O ficheiro train.csv apresenta inicialmente alguns valores de stroke inválidos e alguns dos atributos não se encontram traduzidos consoante a sua designação numérica.

PREPARAÇÃO DO DATASET

Para cumprir o ponto 3.3 do enunciado do Trabalho Prático foram tomadas as seguintes opções:

3.3 a)

Regras de conversão de dados para o ficheiro Train.csv e Script:

Atributo Smoking_status	
Never smoked	0
formerly smoked	1
Smokes	2
Unknown	3

Atributo Gender	
Male	0
Female	1

Atribute ever_married	
No	0
Yes	1

Atributo Residence_type	
Rural	0
Urban	1

```
caminho_arquivo = 'C:\Users\alex-\OneDrive - ISEC\ISEC\ExclusivoDrive\CR\MATLAB-
Drive\TrabalhoPratico\Dataset2-Stroke\Train.csv';

dados = readtable(caminho_arquivo, 'Delimiter', ';');

mapa_smoke = containers.Map({'never smoked', 'formerly smoked', 'smokes', 'Unknown'}, {0, 1, 2, 3});
dados.smoking_status = cellfun(@(x) mapa_smoke(x), dados.smoking_status);

mapa_residence = containers.Map({'Rural', 'Urban'}, {0, 1});
dados.Residence_type = cellfun(@(x) mapa_residence(x), dados.Residence_type);

mapa_married = containers.Map({'No', 'Yes'}, {0, 1});
dados.ever_married = cellfun(@(x) mapa_married(x), dados.ever_married);

mapa_gender = containers.Map({'Male', 'Female'}, {0, 1});
dados.gender = cellfun(@(x) mapa_gender(x), dados.gender);

writetable(dados, caminho_arquivo, 'Delimiter', ';');
```

Foi utilizado o método .Map para mapear os atributos com os respetivos valores numéricos e cellfun para ligar o resultado de cada map aos dados a gravar no ficheiro.

3.3 b)

Como indicado no enunciado do trabalho prático, o atributo que corresponderá à saída(target) neste dataset é o atributo presente na coluna “Stroke“.
Esta coluna inclui em determinadas linhas do ficheiro Train.cvs o valor ”Nan” que será substituído na alínea seguinte.

3.3 c)

De forma a corrigir os valores em falta da coluna “Stroke” de algumas linhas do ficheiro Train.csv, implementou-se a fase de RETRIEVE de um Sistema de Raciocinio Baseado em Casos.

Nesta implementação o objetivo foi encontrar de entre todas as linhas (com exceção das que têm o atributo “Stroke” a Nan) a linha (caso) onde todos os atributos (com exceção do Id e do Stroke) mais se assemelham a cada caso onde “Stroke” tem o valor de Nan.

Código Principal:

```
dados = readtable('C:\Users\alex-\OneDrive - ISEC\ISEC\ExclusivoDrive\CR\MATLAB-
Drive\TrabalhoPratico\Dataset2-Stroke\Train.csv', 'Delimiter', ';');

for i = 1:size(dados, 1)
    if isnan(dados.stroke(i))
        fprintf('Nan no index: %d\n', i);
        [caso_pos] = RETRIEVE(dados, i);
        fprintf('Stroke: %d\n', caso_pos.stroke);
        dados.stroke(i) = caso_pos.stroke;
    end
end

writetable(dados, 'C:\Users\alex-\OneDrive - ISEC\ISEC\ExclusivoDrive\CR\MATLAB-
Drive\TrabalhoPratico\Dataset2-Stroke\Train.csv', 'Delimiter', ';');
```

Script que acede a todas as linhas do ficheiro Train.csv e por cada uma onde a coluna “Stroke” tem o valor Nan corre a função RETRIEVE com os dados do ficheiro e com o nº linha onde esse caso ocorre.
Por fim a função retorna o caso_pos que equivale a essa mesma linha mas com o atributo “Stroke” preenchido segundo as regras da função, atualiza a linha, procura outras linhas a alterar e grava as alterações.

Função RETRIEVE:

```
function caso_pos = RETRIEVE(casos, index)

    pesos = [
        0.1 %gender
        0.1 %age
        0.1 %hypertension
        0.1 %heart_disease
        0.1 %ever_married
        0.1 %Residence_type
        0.1 %avg_glucose_level
        0.1 %bmi
        0.1 %smoking_status
    ];


    novo_caso = casos(index, :);
    %fprintf('Stroke: %s\n', novo_caso.stroke);

    %fprintf('Linhas Antes: %d\n', size(casos, 1));
    nan_indices = isnan(casos.stroke);
    casos = casos(~nan_indices, :);
    %fprintf('Linhas Depois: %d\n', size(casos, 1));

    novo_caso_array = table2array(novo_caso(:, 2:end-1));
    casos_array = table2array(casos(:, 2:end-1));

    similiaridades = zeros(size(casos, 1), 1);

    for i = 1:size(casos, 1)
        distancias = abs(novo_caso_array(1,:) - casos_array(i, :));
        %fprintf('Distancias: %d\n', distancias);
        similiaridade_final = 1 - sum(pesos.*distancias')/sum(pesos);
        %fprintf('similiaridade_final: %d\n', similiaridade_final);
        similiaridades(i) = similiaridade_final;
    end

    [similiaridade_max, max_index] = max(similiaridades);

    caso_pos = casos(max_index, :);

    fprintf('Caso %d de %d com similiaridade de %.2f%%...\n',
max_index, size(casos, 1), similiaridade_max*100);

end
```

Função onde primeiramente se ignoram as linhas onde “Stroke” tem o valor Nan e que para cálculos se ignoram as colunas “Id” e “Stroke”.

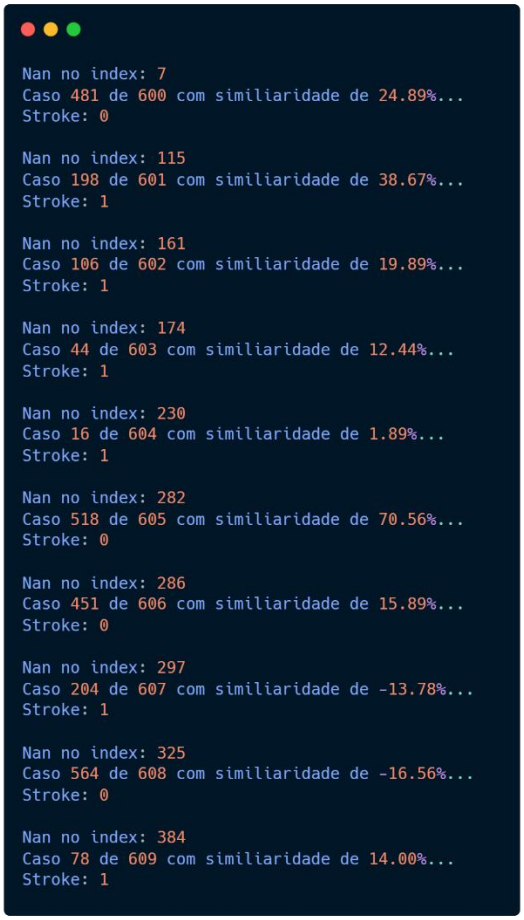
Depois deste passo, a função entra num ciclo ao longo das restantes linhas onde calcula o módulo da diferença entre cada atributo do caso a preencher “novo_caso” e o atributo correspondente na atual linha do ciclo “casos_array(i, :);”

Ainda na mesma iteração do ciclo calcula-se a similiaridade do caso aplicando a formula que subtrai a 1 o somatório da multiplicação entre cada distancia e o peso correspondente no array de pesos dividido pelo somatório de todos os valores desse mesmo array. Adiciona-se a similiaridade ao array de similiaridades.

No final (já fora do ciclo) obtém-se a maior similiaridade para o caso em questão de entre todos os casos possíveis e copiam-se os dados desse caso para o “caso_pos” que acaba a ser retornado (posteriormente apenas o atributo stroke é utilizado, não substituindo os outros atributos).

Os pesos foram todos considerados igualmente importantes, assim todos o valor de 0.1

Resultado e explicação:



Tendo em conta a imagem como exemplo de print da consola no final de correr o script, confirma-se que os objetivos são cumpridos já que em cada linha do ficheiro Train.csv onde “Stroke” é inicialmente Nan é escrita uma mensagem que indica o index do ficheiro onde se encontra o caso a alterar, o index do caso mais similar com este de entre o nº de casos admissíveis e a similiaridade em percentagem, terminando por escrever o novo Stroke.

Nota:
Ao longo dos vários casos o numero de casos admissíveis vai aumentando (600 a 609), isto acontece porque à medida que se torna o valor de “Stroke” de uma linha válido, esta linha passa a ser admissível nos cálculos dos próximos casos.

3.4 a)

Utilizando uma rede neuronal feedforward com uma camada de 10 neurónios com as parametrizações das funções de treino e de ativação default, foi implementado o seguinte script.

```
dados_start = readtable('C:\Users\alex-\OneDrive - ISEC\ISEC\ExclusivoDrive\CR\MATLAB-Drive\TrabalhoPratico\Dataset2-Stroke\Start.csv');

X = dados_start{:, 2:end-1}; % entrada: tudo menos o ID e o stroke (input)
y = dados_start.stroke; % saida: stroke (target)

net = feedforwardnet(10); % 10 neurónios

tic;
net = train(net, X', y');
%net.trainFcn = 'trainbfg';
%rede.layers.transferFcn = 'logsig';
tempo_execucao = toc;

%teste com o target original
y_pred = net(X');
accuracy = sum(round(y_pred) == y') / numel(y);
erroR = mse (net, y', y_pred);

fprintf('Tempo de Train: %.2f segundos\n', tempo_execucao);
fprintf('Accuracy: %.2f%%\n', accuracy * 100);
fprintf('Erro: %.2f%%\n', erroR*100);
```

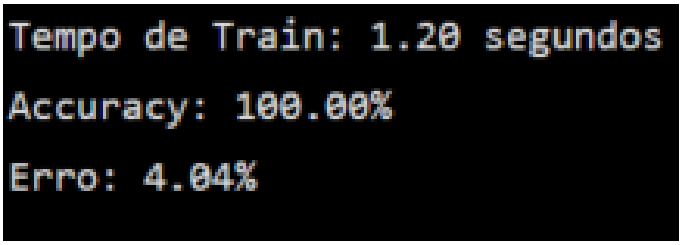
Utilizando o ficheiro Start.csv do dataset começando por dividi-lo entre X com os dados de entrada que são posteriormente utéis para o treinar o modelo (todas as colunas com exceção do Id e do Stroke) e Y com as saídas, neste caso o stroke (target).

Como indicado no enunciado todos os exemplos deste ficheiro devem ser utilizados para treino (sem segmentação), desta forma o modelo é treinado para os mesmos dados que são depois utilizados para o Teste (predict) do modelo o que não é o mais indicado neste tipo de problemas por ajustar demasiado o modelo aos dados e impossibilitar a generalização.

Depois do “teste” do modelo com os inputs iniciais o modelo cria a variável y_pred que corresponde às previsões do que será a coluna Stroke, porém estes valores são decimais já que é a forma que o modelo utiliza para representar os dados mais aproximados aos ideais.

Desta forma no calculo da accuracy a coluna y_pred é arredondada para uma comparação direta com os targets, já que estes têm valores binários(0/1). Depois é calculado o erro entre os valores ideais e os valores de y_pred.

Resultados e conclusões:

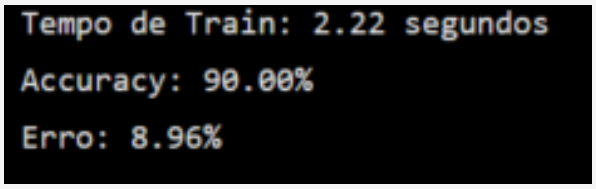


Como esperado os resultados do modelo foram muito positivos, isto deve-se ao treino ser efetuado com os mesmo dados que os usados para prever (testar) o modelo.
É de notar que a accuracy é de 100% porém os dados retornados pelo modelo, como anteriormente indicado foram arredondados de forma a serem comparados com os valores (0 ou 1) do Target inicial. Desta forma o erro real foi de 4.04 %.

Alteração nas funções de ativação e treino:

Usando Função de treino '`trainlm`' e Função de Ativação '`tansig`':

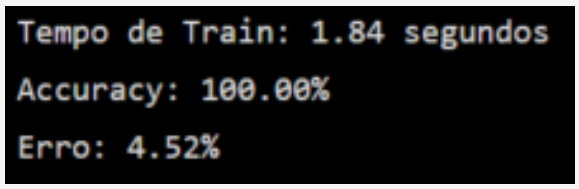
```
net.trainFcn = 'trainlm';
rede.layers.transferFcn = 'tansig';
```



Em comparação com as parametrizações default anteriormente testadas, obteve-se um tempo de Treino aproximadamente 1 segundo superior, uma accuracy 10% inferior e um erro real do dobro em percentagem.

Usando Função de treino '`trainbfg`' e Função de Ativação '`logsig`':

```
net.trainFcn = 'trainbfg';
rede.layers.transferFcn = 'logsig';
```



Em comparação com as parametrizações default inicialmente testadas, obteve-se um tempo de Treino igualmente baixo mas superior, uma accuracy arredondada de 100 % e um erro de apenas 4.52%.

Tendo realizado poucas iterações de cada uma destas parametrizações não é possível obter conclusões sólidas de qual das opções de configuração de funções de ativação e treino é melhor, estas conclusões podem ser tiradas com maior certeza na alínea seguinte.

3.4 b)

Depois de preparado na alínea 3.3) o ficheiro Train.csv é utilizado para estudar varias configurações diferentes de modelos e guardar as melhores 3 (quanto à accuracy).

Configurações testadas:
Numero de camadas: 1 ou 2;
Numero de neurónios: 5, 10 ou 15;
Funções de treino: 'trainlm', 'trainbfg', 'traingd';
Funções de ativação: 'tansig', 'purelin', 'logsig';

Para testar todas as configurações possíveis com as opções anteriores foi desenvolvido o seguinte código:

```
dados_train = readtable('C:\Users\alex-\OneDrive - ISEC\ISEC\ExclusivoDrive\CR\MATLAB-Drive\TrabalhoPratico\Dataset2-
Stroke\Train.csv');
X = dados_train{:, 2:end-1}; % entrada: tudo menos o ID e o stroke (input)
y = dados_train.stroke; % saída: stroke (target)

p_train = 0.7;
p_val = 0.15;
p_test = 0.15;
[TrainInd, ValInd, TestInd] = dividerand(size(X,1), p_train, p_val, p_test);

X_train = X(TrainInd,:);
y_train = y(TrainInd,:);
X_val = X(ValInd,:);
y_val = y(ValInd,:);
X_test = X(TestInd,:);
y_test = y(TestInd,:);

function [rede, tempo_treino, accuracy, erro_real] = treinar_rede(X_train, y_train, X_val, y_val, X_test, y_test, num_camadas,
num_neuronios, funcao_treino, funcoes_ativacao1, funcoes_ativacao2)
    rede = feedforwardnet(num_neuronios);

    rede.trainFcn = funcao_treino;
    if num_camadas == 1
        rede.layers{1}.transferFcn = funcoes_ativacao1;
    else
        rede.layers{1}.transferFcn = funcoes_ativacao1;
        rede.layers{2}.transferFcn = funcoes_ativacao2;
    end
    X_train_val = [X_train; X_val];
    y_train_val = [y_train; y_val];

    % aplicar o conjunto de validação
    rede.divideFcn = 'dividerand';
    rede.divideParam.trainRatio = size(X_train, 1) / size(X_train_val, 1);
    rede.divideParam.valRatio = size(X_val, 1) / size(X_train_val, 1);
    rede.divideParam.testRatio = 0; % nao usar o conjunto de teste no treino
    tic;
    rede = train(rede, X_train_val', y_train_val');
    tempo_treino = toc;

    y_predict_test = rede(X_test');

    accuracy = sum(round(y_predict_test) == y_test') / numel(y_test);
    erro_real = mse(rede, y_test', y_predict_test);
end
```

Inicialmente são ignoradas as colunas de id e stroke (no X) e os dados são divididos entre treino validação e teste com uma proporção de 70%, 15% e 15% respetivamente.

A função treinar_rede que tem como objetivo a criação do modelo segundo a parametrização passada utilizando os dados de train e de validação para treinar a rede

e utilizar os dados do test para testar a rede, acabando por retornar as métricas obtidas e a rede criada.

Nesta função tanto os dados de treino quanto os de validação serão usados para ajustar os parâmetros da rede.

rede.divideFcn = 'dividerand'; : Define o método de divisão dos dados durante o train da rede. Neste caso, foi usado o método 'dividerand', que divide os dados de maneira aleatória.
rede.divideParam.trainRatio = size(X_train, 1) / size(X_train_val, 1); e **rede.divideParam.valRatio = size(X_val, 1) / size(X_train_val, 1); :** Estas linhas definem as proporções dos dados de treino e validação em relação aos conjuntos de dados **(X_train_val e y_train_val)** combinados.

O parâmetro **trainRatio** especifica a proporção de dados usados para treino, enquanto **valRatio** especifica a proporção de dados usados para validação.

rede.divideParam.testRatio = 0; : Define teste como zero, o que significa que nenhum dado de teste é usado durante o treino da rede. Isto foi implementado para a rede não ser treinada com a influencia dos dados com que será testada no passo seguinte.

Depois do modelo ser criado treinado e testado a função retorna as métricas obtidas e a própria rede.

```
%num_camadas = [1,2]
num_neuronios = [5, 10, 15];
funcoes_treino = {'trainlm', 'trainbfg', 'traingd'};
funcoes_ativacao1 = {'tansig', 'purelin', 'logsig'};
funcoes_ativacao2 = {'tansig', 'purelin', 'logsig'};

melhores_redes = cell(3, 1);
melhor_desempenho_redes = zeros(3, 8);

for num_camadas = 1:2
    for i = 1:length(num_neuronios)
        for j = 1:length(funcoes_treino)
            for k = 1:length(funcoes_ativacao1)
                if num_camadas == 2
                    for l = 1:length(funcoes_ativacao2)
                        fprintf('\n---------------------------------\n');
                        fprintf('\nConfigurações:\n');
                        fprintf('Num_neuronios: %d\n', num_neuronios(i));
                        fprintf('Num_camadas: %d\n', num_camadas);
                        fprintf('Funcao_treino: %s\n', funcoes_treino{j});
                        fprintf('Funcao_ativacao1: %s\n', funcoes_ativacao1{k});
                        fprintf('Funcao_ativacao2: %s\n', funcoes_ativacao2{l});
                        [rede, tempo_treino, accuracy, erro_real] = treinar_rede(X_train, y_train, X_val, y_val, X_test, y_test,
                                    num_camadas, num_neuronios(i), funcoes_treino{j}, funcoes_ativacao1{k}, funcoes_ativacao2{l});
                        fprintf('Tempo de Execução: %.2f segundos\n', tempo_treino);
                        fprintf('Accuracy: %.2f%%\n', accuracy*100);
                        fprintf('Erro: %.2f%%\n', erro_real*100);
                        [~, indice_pior_rede] = min(melhor_desempenho_redes(:, 1));
                        if isempty(melhores_redes{indice_pior_rede}) || accuracy > melhor_desempenho_redes(indice_pior_rede, 1)
                            melhores_redes{indice_pior_rede} = rede;
                            melhor_desempenho_redes(indice_pior_rede, :) = [accuracy, erro_real, tempo_treino, num_neuronios(i),
                                    num_camadas, j, k, l];
                        end
                    end
                else
                    fprintf('\n--------------------------------\n');
                    fprintf('\nConfigurações:\n');
                    fprintf('Num_neuronios: %d\n', num_neuronios(i));
                    fprintf('Num_camadas: %d\n', num_camadas);
                    fprintf('Funcao_treino: %s\n', funcoes_treino{j});
                    fprintf('Funcao_ativacao1: %s\n', funcoes_ativacao1{k});
                    [rede, tempo_treino, accuracy, erro_real] = treinar_rede(X_train, y_train, X_val, y_val, X_test, y_test,
                                num_camadas, num_neuronios(i), funcoes_treino{j}, funcoes_ativacao1{k}, '');
                    fprintf('Tempo de Execução: %.2f segundos\n', tempo_treino);
                    fprintf('Accuracy: %.2f%%\n', accuracy*100);
                    fprintf('Erro: %.2f%%\n', erro_real*100);
                    [~, indice_pior_rede] = min(melhor_desempenho_redes(:, 1));
                    if isempty(melhores_redes{indice_pior_rede}) || accuracy > melhor_desempenho_redes(indice_pior_rede, 1)
                        melhores_redes{indice_pior_rede} = rede;
                        melhor_desempenho_redes(indice_pior_rede, :) = [accuracy, erro_real, tempo_treino, num_neuronios(i),
                                num_camadas, j, k, 0];
                    end
                end
            end
        end
    end
end

etiquetas = {'Accuracy', 'Erro_Real', 'Tempo_Treino', 'Num_Neuronios', 'Num_Camadas', 'Funcao_Treino', 'Funcao_Ativacao1',
                                'Funcao_Ativacao2'};
melhor_desempenho_redes_tabela = array2table(melhor_desempenho_redes, 'VariableNames', etiquetas);
writetable(melhor_desempenho_redes_tabela, 'melhor_desempenho_redes.csv');

for i = 1:3
    nome_arquivo = sprintf('melhor_rede_%d', i);
    nome_rede = melhores_redes{i};
    save(nome_arquivo, 'nome_rede');
end
```

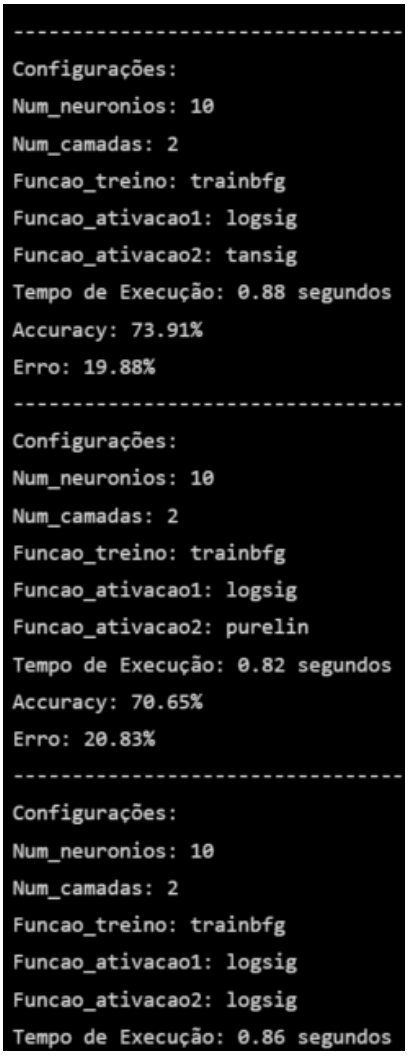
Com este código são inicializadas as diversas opções de parametrização e os arrays que no final são utilizados para criar os 3 ficheiros com as 3 melhores redes e o ficheiro melhor_desempenho_redes.csv com as métricas das 3 melhores redes.

Para isso o código resume-se a 5 ciclos que correm todas as combinações possíveis entre as parametrizações possíveis, dentro deste ciclo são mostradas as métricas aplicadas, é chamada a função treinar_rede, escritos os resultados e atualizados os arrays das melhores redes e dos melhores desempenhos.

Por fim é criado o ficheiro com as métricas e os resultados das melhores redes (‘melhor_desempenho_redes.csv’) e 3 ficheiros com as melhores 3 redes (segundo a accuracy).

Conclusões e resultados:

Exemplo de alguns resultados na consola:



Resultado do ficheiro melhor_desempenho_redes.csv:

3 melhores redes e resultados associados, tendo em conta a seguinte legenda:

Função de Treino:
Trainlm: 1, Trainbfg: 2, Traingd: 3

Função de Ativação:
Tansig:1, Purelin: 2, Logsig: 3

Accuracy	Erro_Real	Tempo_Treino	Num_Neuronios	Num_Camadas	Funcao_Treino	Funcao_Ativacao1	Funcao_Ativacao2
0.77173913...	0.19299108...	1.1685586	15	1	1	1	0
0.76086956...	0.17963353...	1.137675	10	2	1	1	1
0.78260869...	0.17826780...	1.0731712	15	2	2	1	1

Conclusões e resultados:

Concluindo de forma geral, confirma-se que a accuracy é bastante inferior à obtida na alínea 3.4 a) e isto deve-se principalmente deixar de utilizar os mesmo dados para treino e teste o que faz com que o modelo consiga prever com menor accuracy os targets ideais.

As redes com melhores resultados foram as que têm mais neurónios e/ou mais numero de camadas.
A função de ativação da primeira camada que leva a uma maior accuracy é a "Tansig".

Concluindo quanto a alterações de parâmetros específicos:

	Número de camadas escondidas	Número de neurónios	Funções de ativação	Função de treino	Divisão dos exemplos	Precisão	Erro Real
O número e dimensão das camadas encondidas influencia o desempenho?							
Conf1	1	5	tansig	trainlm	{0.7, 0.15, 0.15}	71.74%	19.45%
Conf2	1	10	tansig	trainlm	{0.7, 0.15, 0.15}	67.39%	20.91%
Conf3	1	15	tansig	trainlm	{0.7, 0.15, 0.15}	77.17%	19.30%
Conf4	2	5	tansig, tansig	trainlm	{0.7, 0.15, 0.15}	70.65%	20.59%
Conf5	2	10	tansig, tansig	trainlm	{0.7, 0.15, 0.15}	76.09%	17.96%
Conf6	2	15	tansig, tansig	trainlm	{0.7, 0.15, 0.15}	65.22%	22.63%

Pode-se concluir com este estudo que em geral mas não em regra (como se comprova pelas configurações Conf1 e Conf6) o aumento do numero de camadas escondidas da rede bem como o aumento do número de neurónios conduzem a um aumento da precisão.
Um dos motivos para a Conf6 não apresentar bons resultados pode ser o overfitting da rede devido a uma demasia de neurónios e nº camadas.
Em alguns casos, uma rede mais complexa pode manipular melhor a complexidade dos dados, levando a um melhor desempenho. No entanto, em outros casos, uma rede mais simples pode evitar o overfitting e generalizar melhor para dados de teste não treinados.

	camadas escondidas	Número de neurónios	Funções de ativação	Função de treino	Divisão dos exemplos	Precisão	Erro Real
A função de treino influencia o desempenho?							
Conf1	1	10	purelin	trainlm	dividerand = {0.7, 0.15, 0.15}	69.57%	20.03%
Conf2	1	10	purelin	trainbfg	dividerand = {0.7, 0.15, 0.15}	68.48%	20.06%
Conf3	1	10	purelin	traingd	dividerand = {0.7, 0.15, 0.15}	68.48%	22.46%
Conf1	2	10	tansig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}	68.48%	21.20%
Conf2	2	10	tansig, purelin	trainbfg	dividerand = {0.7, 0.15, 0.15}	73.91%	19.91%
Conf3	2	10	tansig, purelin	traingd	dividerand = {0.7, 0.15, 0.15}	65.22%	21.70%

Diferentes algoritmos de treino têm diferentes propriedades, como a capacidade de evitar minimos locais e a sensibilidade dos hiperparâmetros. A seleção da função de

treino influencia diretamente a qualidade do ajuste dos pesos da rede neuronal aos dados.
Conclui-se que a função de treino “trainbfg” foi a configuração que resultou em melhores resultados usando 2 camadas escondidas, porém a rede com apenas 1 camada conseguiu obter melhores resultados utilizando a função “trainlm”.

As funções de ativação influenciam o desempenho?							
Conf1	2	15	tansig, tansig	trainlm	{0.7, 0.15, 0.15}	65.22%	22.63%
Conf2	2	15	tansig, purelin	trainlm	{0.7, 0.15, 0.15}	72.83%	20.32%
Conf3	2	15	tansig,logsig	trainlm	{0.7, 0.15, 0.15}	35.87%	24.83%
Conf4	2	15	purelin, tansig	trainlm	{0.7, 0.15, 0.15}	69.57%	19.28%
Conf5	2	15	purelin, purelin	trainlm	{0.7, 0.15, 0.15}	69.57%	20.62%
Conf6	2	15	purelin,logsig	trainlm	{0.7, 0.15, 0.15}	35.87%	25.63%
Conf7	2	15	logsig, tansig	trainlm	{0.7, 0.15, 0.15}	71.74%	20.43%
Conf8	2	15	logsig, purelin	trainlm	{0.7, 0.15, 0.15}	69.57%	20.25%
Conf9	2	15	logsig,logsig	trainlm	{0.7, 0.15, 0.15}	35.87%	25.49%

As funções de ativação determinam a saída de cada neurónio, têm um papel importante na capacidade da rede neuronal modelar relações complexas de dados.
Quanto às funções de ativação retira-se que a utilização da função de ativação ”logsig” na segunda camada escondida resultou em menores valores de precisão (sempre de 35.87%).

A divisão de exemplos pelos conjuntos influencia o desempenho?							
Conf1	1	10	tansig	trainlm	dividerand = {0.7, 0.15, 0.15}	67.39%	20.91%
Conf2	1	10	tansig	trainlm	dividerand = {0.9, 0.05, 0.05}	70.97%	25.77%
Conf3	1	10	tansig	trainlm	dividerand = {0.75, 0.15, 0.10}	72.13	21.37%
					dividerand = {Train, Val, Test}		

Como esperado o aumento do conjunto de teste por diminuição do conjunto de test leva a melhores resultados.

3.4 c)

Nesta alínea foram utilizadas as 3 melhores redes obtidas na alínea anterior aplicando o ficheiro Start.csv do dataset para treinar a rede e mostrar as métricas obtidas.

Para isso este foi o código desenvolvido, que se baseia numa simplificação dos códigos já utilizados excluíndo a fase de treino.

```
dados_test = readtable('C:\Users\alex-\OneDrive - ISEC\ISEC\ExclusivoDrive\CR\MATLAB-Drive\TrabalhoPratico\Dataset2-Stroke\Test.csv');

X_teste = dados_test{:, 2:end-1}; % entrada: tudo menos o ID e o stroke (input)
y_teste = dados_test.stroke; % saída: stroke (target)

num_redes = 3; % Número de redes treinadas
precisao_redes = zeros(num_redes, 1);
erro_real_redes = zeros(num_redes, 1);

for i = 1:num_redes
    nome_arquivo_rede = sprintf('melhor_rede_%d.mat', i);
    load(nome_arquivo_rede, 'nome_rede');

    y_predict = nome_rede(X_teste');
    precisao_redes(i) = sum(round(y_predict) == y_teste') / numel(y_teste);
    erro_real_redes(i) = mse(nome_rede, y_teste', y_predict);
end

for i = 1:num_redes
    fprintf('\n---------------------------------\n');
    fprintf('\nMétricas da Rede %d:\n', i);
    fprintf('Precisão: %.2f%%\n', precisao_redes(i)*100);
    fprintf('Erro: %.2f%%\n', erro_real_redes(i)*100);
end
```

Resultados e conclusões:

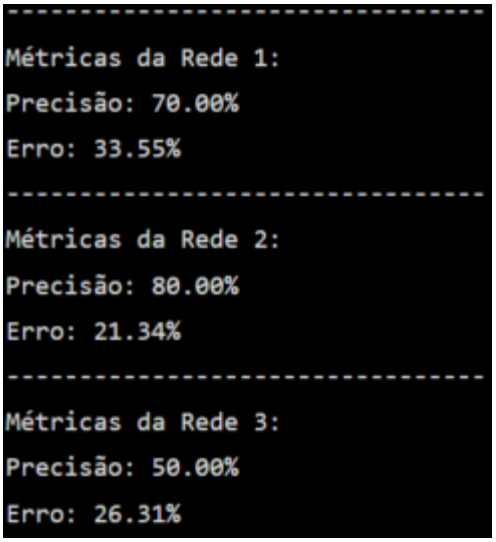
Sabendo que as características das redes são(como indicado na alínea anterior) respetivamente:

Accuracy	Erro_Real	Tempo_Trei...	Num_Neuro...	Num_Cama...	Funcao_Trei...	Funcao_Ativ...	Funcao_Ativ...
0.77173913...	0.19299108...	1.1685586	15	1	1	1	0
0.76086956...	0.17963353...	1.137675	10	2	1	1	1
0.78260869...	0.17826780...	1.0731712	15	2	2	1	1

Com:
Função de Treino:
Trainlm: 1, Trainbfg: 2, Traingd: 3

Função de Ativação:
Tansig:1, Purelin: 2, Logsig: 3

Métricas obtidas



As redes aplicadas no ficheiro da alínea anterior conseguiram resultados melhores, no ficheiro “Test.csv” aplicando a rede 3 com 2 camadas e 15 neuronios os resultados foram piores, provavelmente porque ocorreu overfitting (a rede ajustou os seus parâmetros demasiado ao conjunto da alínea anterior, não conseguindo generalizar neste caso).