Departamento de Engenharia Informática e de Sistemas
Instituto Superior de Engenharia de Coimbra
Instituto Politécnico de Coimbra

Licenciatura em Engenharia Informática

Curso Diurno

Ramo de Desenvolvimento de Aplicações

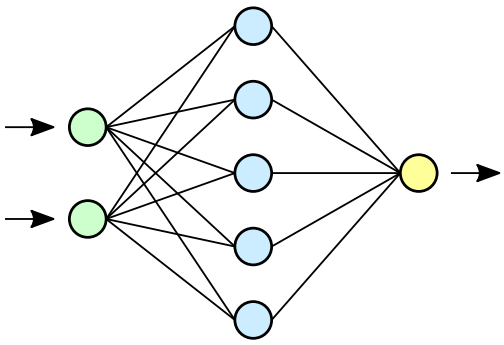
Unidade Curricular de Conhecimento e Raciocinio

Ano Lectivo de 2023/2024

Trabalho Prático

Docente Responsável: Anabela Simões

ESTUDO DE CLASSIFICAÇÃO DE DATASET ATRAVÉS RACÍOCINIO BASEADO EM CASOS E REDES NEURONAIS



Alexandre Ramalho Silva | 2021145409
André Filipe Lopes Dias | 2021140917
Coimbra, 12 de maio de 2024

Índice

1. Introdução

Este relatório tem como objetivo oferecer uma análise abrangente dos principais pontos abordados na realização do trabalho prático no âmbito da unidade curricular de Conhecimento e Raciocínio.

Utilizando o conjunto de dados "Stroke", o trabalho inicia-se com uma etapa fundamental de preparação dos dados, que envolve a conversão de atributos textuais em numéricos e a implementação de um processo de raciocínio baseado em casos (CBR) para lidar com valores ausentes.

Em seguida, a análise aprofunda-se na aplicação de redes neurais para a classificação desses casos, destacando a importância da escolha adequada de arquiteturas, funções de treinamento e de ativação.

Além disso, são exploradas técnicas de retrieve para encontrar casos semelhantes e preencher valores em falta, ampliando a robustez e precisão do modelo. O estudo visa não apenas desenvolver um modelo eficaz de classificação, mas também fornecer insights valiosos sobre a aplicação prática de técnicas de análise de dados e inteligência artificial na área da saúde.

1. Escolha do Dataset

2.1. Descrição

O dataset selecionado para a realização deste trabalho prático foi o "Stroke".
Este dataset Stroke contém informações relevantes relacionadas a casos de derrame cerebral (stroke). Ele consiste em 12 atributos, que incluem tanto valores numéricos quanto textuais, e é destinado à classificação de casos de derrame cerebral.

2.2. Características

Colunas: 12 atributos, uma mistura de valores numéricos e texto.

Linhas do Ficheiro Start: 10

Linhas do Ficheiro Train: 620 - Este conjunto de dados possui valores em falta (NA), que requerem pré-processamento.

Linhas do Ficheiro Test: 10

TARGET: Classificação binária, representada pela coluna "Stroke", com valores 1 (indicando a ocorrência de derrame cerebral) e 0 (indicando a ausência de derrame cerebral).

2. Preparação do Dataset

2.1. Conversão de atributos string para numéricos

Durante o processo de preparação do dataset, realizámos a conversão dos atributos do tipo string em valores numéricos, conforme especificado nos ficheiros Start e Test. Essa etapa foi crucial para garantir que os dados estejam num formato adequado para serem utilizados em redes neurais.
As transformações foram aplicadas manualmente da seguinte maneira:

Atributo Smoking_status:

Valores Originais: {"never smoked", "formerly smoked", "smokes", "unknown"}

Valores Convertidos: {"never smoked"->0, "formerly smoked"->1, "smokes"->2, "unknown"->3}

Gender:

Valores Originais: {"Male", "Female"}

Valores Convertidos: {"Male"->0, "Female"->1}

Residence_type:

Valores Originais: {"Rural", "Urban"}

Valores Convertidos: {"Rural"->0, "Urban"->1}

Ever married:

Valores Originais: {"No", "Yes"}

Valores Convertidos: {"No"->0, "Yes"->1}

2.2. Implementação de racíocinio baseado em casos e retrieve

Raciocínio Baseado em Casos (CBR):

O Raciocínio Baseado em Casos é um método de solução de problemas que utiliza casos passados semelhantes para resolver problemas novos. No contexto deste código, o CBR é aplicado para lidar com valores em falta (NA) no atributo "stroke" do dataset Stroke.

Função cbr():

Identificação de Casos com Valores em Falta

O dataset é lido a partir do ficheiro 'Train.csv' e os casos com valores NA no atributo "stroke" são identificados e separados. Isso é feito para tratar os casos com valores em falta de maneira diferente dos casos completos.

Preenchimento dos Valores em Falta

Para cada caso com valores em falta, o CBR é utilizado para encontrar casos semelhantes do conjunto de dados completo. Essa busca por casos semelhantes é realizada pela função Retrieve.

Atualização do Dataset

Após encontrar casos semelhantes, os valores de "stroke" dos casos com valores em falta são substituídos pelos valores dos casos mais semelhantes encontrados durante o processo de Retrieve.

Função retrieve():

Cálculo da Similaridade

A similaridade entre os casos é calculada usando diferentes métricas de distância, como distância linear e euclidiana, dependendo do tipo de atributo (categórico ou numérico). Isso leva em consideração tanto a magnitude quanto a natureza dos atributos.

Ponderação dos Atributos

Os atributos são ponderados com base em sua importância relativa para determinar a similaridade global entre os casos. Esses pesos são especificados na função retrieve() através da variável weighting_factors.

Limiar de Similaridade

A similaridade calculada entre os casos é comparada com um limiar de similaridade (threshold) definido. Apenas casos com uma similaridade acima desse limiar são considerados suficientemente semelhantes para imputação de valores.

Seleção do Target "stroke"

Ao encontrar os casos mais semelhantes aos casos com valores em falta, o CBR utiliza o atributo "stroke" desses casos semelhantes como uma referência para preencher os valores em falta nos casos originais. Isto é feito garantindo que os casos selecionados sejam suficientemente semelhantes aos casos com valores em falta, de acordo com o limiar de similaridade estabelecido. Esta abordagem permite uma classificação de valores eficaz e robusta, preservando a integridade dos dados e garantindo a qualidade dos resultados finais.

2.3. Funções de similaridade e Pesos dos atributos

Funções de Similaridade:

Distância Linear

A função calculate_linear_distance(val1, val2) calcula a distância linear entre dois valores numéricos ou categoricos. Para atributos numéricos, a distância linear é a diferença absoluta entre os valores. Para atributos categóricos, a distância linear é a diferença absoluta entre os índices dos valores.

Distância Euclidiana

A função calculate_euclidean_distance(val1, val2) calcula a distância euclidiana entre dois valores numéricos. A distância euclidiana é a raiz quadrada da soma dos quadrados das diferenças entre os valores.

Pesos dos Atributos:

Os pesos atribuídos a cada atributo são especificados na função retrieve() pela variável weighting_factors. Estes pesos são usados para ponderar a contribuição de cada atributo para a similaridade global entre os casos.

Definição dos Pesos

gender: Embora o sexo também possa influenciar o risco de AVC, atribuimos um peso relativamente menor (2) em comparação com outros fatores de risco mais diretos.

age: A idade pode ser um fator importante na incidência de acidente vascular cerebral (AVC), sendo um dos fatores de risco mais significativos. Atribuimos um peso mais alto (5) a este atributo.

hypertension: A hipertensão arterial é um dos principais fatores de risco para AVC. Atribuimos um peso elevado para este atributo (4).

heart_disease: Doenças cardíacas também estão associadas ao risco de AVC. Atrbuimos um peso moderado para este atributo (4).

ever_married: O estado civil pode ter alguma correlação com o risco de AVC, mas vamos atribuimos um peso menor (1) em comparação com os fatores de saúde mais diretos.

Residence_type: O tipo de residência pode ter um impacto mínimo no risco de AVC. Atribuimos um peso baixo (1) para este atributo.

avg_glucose_level: Níveis elevados de glucose no sangue também podem aumentar o risco de AVC. Atribuimos um peso moderado (3) para este atributo.

bmi: O índice de massa corporal (IMC) também pode ser um indicador de risco de AVC. Atribuimos atribuir um peso moderado (3) para este atributo.

smoking_status: Embora o tabagismo seja um fator de risco para AVC, atribuimos um peso menor (2) para este atributo em comparação com os fatores de saúde mais diretos.

Aplicação dos Pesos

Durante o cálculo da similaridade entre dois casos na função retrieve(), os pesos são utilizados para ponderar a contribuição de cada atributo para a similaridade global.

Isto é feito multiplicando cada distância pelo peso correspondente e, em seguida, dividindo pela soma de todos os pesos. Essa média ponderada das distâncias é usada para calcular a similaridade final entre os casos.

3. Estudo e Análise de Redes Neuronais

3.1. Utilização do ficheiro START

Nesta fase, implementámos a função start(), que é responsável por iniciar o processo de treinamento e avaliação de uma rede neural feedforward para classificação usando o conjunto de dados de partida (Start.csv).

O conjunto de dados é lido do arquivo 'Start.csv' usando a função readtable. Os atributos de entrada são armazenados na matriz input e o atributo alvo (Stroke) é armazenado na matriz output.

Uma rede neural feedforward é criada usando a função feedforwardnet(10), com 10 neurónios na camada oculta.

A função de treinamento é definida como 'trainlm', que corresponde ao algoritmo de treinamento Levenberg-Marquardt. As funções de ativação para a camada oculta e de saída são definidas como 'tansig' e 'purelin', respetivamente.

Os exemplos são divididos em conjuntos de treinamento, validação e teste usando a função 'dividerand'. Neste caso, apenas o conjunto de treino é utilizado (100% dos exemplos).

A rede neural é treinada com os dados de entrada e saída usando a função train. A função retorna a rede treinada, que é armazenada na variável net.

A rede neural é simulada usando a função sim para gerar as previsões de saída para os dados de entrada. As previsões são então normalizadas para o intervalo [0, 1] e arredondadas para obter previsões binárias.

A precisão total da rede é calculada comparando as previsões geradas com as saídas reais. A precisão é calculada como a porcentagem de previsões corretas em relação ao total de exemplos.

O erro da rede neural é calculado usando a função perform em relação aos dados de saída reais.

A precisão total, o erro e o tempo de execução são impressos na tela.

Esta função fornece uma visão abrangente do desempenho da rede neural feedforward na classificação dos dados de entrada em relação ao atributo alvo "stroke".

A função de treino influencia o desempenho?

A função de treino influencia o desempenho?									
Conf1	1	10	tansig, purelin	traingd	dividerand = {1.0, 0.0, 0.0}		100.0%	0.0%	0.88s
Conf2	1	10	tansig, purelin	trainbfg	dividerand = {1.0, 0.0, 0.0}		100.0%	0.0%	0.69s
Conf3	1	10	tansig, purelin	trainlm	dividerand = {1.0, 0.0, 0.0}		100.0%	0.0%	0.73s
Conf4	1	10	tansig, purelin	trainr	dividerand = {1.0, 0.0, 0.0}		100.0%	0.0%	0.67s

Todas as quatro funções de treino atingiram 100% de precisão na tarefa de treino, o que significa que todas as funções de treino foram capazes de aprender a mapear corretamente os dados de entrada para os dados de saída.

A função de treino trainbfg foi a mais rápida, com um tempo de treinamento de 0,67 segundos.

A função de treino traingd foi a mais lenta, com um tempo de treinamento de 0,88 segundos.

Não é uma diferença significativa para se tornar um parâmentro diferenciador.

As funções de ativação influenciam o desempenho?

As funções de ativação influenciam o desempenho?									
Conf1	1	10	logsig, purelin	trainlm	dividerand = {1.0, 0.0, 0.0}		100.0%	0.0%	0.88s
Conf2	1	10	poslin, logsig	trainlm	dividerand = {1.0, 0.0, 0.0}		90.0%	10.0%	0.71s
Conf3	1	10	tansig, purelin	trainlm	dividerand = {1.0, 0.0, 0.0}		100.0%	0.0%	0.73s
Conf4	1	10	poslin, purelin	trainlm	dividerand = {1.0, 0.0, 0.0}		100.0%	0.0%	0.67s

Todas as quatro funções de ativação atingiram 100% de precisão na tarefa de ativação, o que significa que todas as funções de ativação foram capazes de aprender a mapear corretamente os dados de entrada para os dados de saída.

A confg4, teve o melhor desempenho, com tempo de execução de 0.67s.

A confg1, teve o pior desempenho, com tempo de execução de 0.88s.

Estes resultados sugerem que a função de ativação não tem grande impacto no desempenho da rede neuronal.

3.2. Utilização do ficheiro TRAIN

Nesta fase, implementámos a função train(), que é responsável por carrega os dados de treinamento do arquivo 'Train_atualizado.csv' e divide os dados em entrada (X) e target (y).

Inicializa variáveis para acompanhar a melhor precisão, o melhor erro e a melhor rede neural encontrada durante o treinamento. Trata a definição da arquitetura da rede neuronal, incluindo o número de neurónios na camada oculta.

Indica a função de treinamento (Levenberg-Marquardt) e as funções de ativação para as camadas ocultas e de saída da rede.

Configura a divisão dos dados em conjuntos de treinamento, validação e teste.

Itera sobre 10 treinamentos diferentes, registrando a precisão total, o erro e o tempo de execução de cada um.

Verifica se os resultados de precisão e erro atuais são melhores do que os anteriores e atualiza os valores correspondentes.

Calcula a média da precisão total, do erro e do tempo de execução das 10 iterações.

Guarda a melhor rede neural encontrada num arquivo .mat, substituindo arquivos anteriores se necessário.

Essencialmente, a função Train() realiza uma procura exaustiva por configurações de rede neural que otimizem a precisão na classificação do atributo alvo "stroke" com base nos dados de treino fornecidos.

	Número de camadas escondidas	Número de neurónios	Funções de ativação	Função de treino	Divisão dos exemplos		Precisão Global	Erro	Precisão do Teste
Configuração por defeito	1	10 30 15	tansig,tansig,tansig, purelin	trainlm	dividerand = {0.975, 0.01, 0.015}		81.44%	18.56%	73.33%

A arquitetura da rede feedforward influencia o desempenho?

A arquitetura da rede feedforward influencia o desempenho?									
Conf1	3	10 30 15	tansig, tansig, tansig, purelin	trainlm	dividerand = {0.975, 0.01, 0.015}		75.62%	24.38%	68.33%
Conf2	4	10 20 30 15	tansig, tansig, tansig, tansig, purelin	trainlm	dividerand = {0.975, 0.01, 0.015}		78.55%	21.45%	70%
Conf3	2	20 20	tansig, tansig, purelin	trainlm	dividerand = {0.975, 0.01, 0.015}		80.76%	19.24%	70.56%
Conf4	1	20	tansig, purelin	trainlm	dividerand = {0.975, 0.01, 0.015}		80.70%	19.30%	73.33%

As quatro configurações conseguiram resultados bastante semelhantes, o que sugere que, para o problema específico em questão, diferentes arquiteturas de rede feedforward não tiveram um impacto significativo no desempenho geral.

A Conf3 obteve o melhor desempenho global, embora a diferença em relação às outras configurações não seja muito grande.

Isto pode indicar que a combinação específica de parâmetros e funções de ativação utilizada na Conf3 foi mais adequada para o problema em questão.

A função de treino influencia o desempenho?

A função de treino influencia o desempenho?

Conf1	1	10 30 15	tansig,tansig,tansig, purelin	trainlm	dividerand = {0.975, 0.01, 0.015}	81.44%	18.56%	73.33%
Conf2	1	10 30 15	tansig,tansig,tansig, purelin	trainbfg	dividerand = {0.975, 0.01, 0.015}	70.52%	29.48%	67.78%
Conf3	1	10 30 15	tansig,tansig,tansig, purelin	traingd	dividerand = {0.975, 0.01, 0.015}	65.51%	34.49%	62.22%
Conf4	1	10 30 15	tansig,tansig,tansig, purelin	trains	dividerand = {0.975, 0.01, 0.015}	74.59%	25.41%	76.67%

Conf1 (trainlm): Esta configuração parece ter o melhor desempenho geral, com uma precisão de 81.44%. A função de treino trainlm é conhecida por ser bastante eficaz para muitos problemas, especialmente quando se trata de conjuntos de dados grandes ou complexos.

Conf2 (trainbfg): Embora ainda tenha um desempenho razoável, a precisão de 70.52% é menor em comparação com a Conf1. A função de treino trainbfg (BFGS Quasi-Newton Backpropagation) pode não ser tão adequada para o teu problema específico ou pode exigir uma configuração diferente de parâmetros.

Conf3 (traingd): Esta configuração tem o pior desempenho entre as quatro. A função de treino traingd (Gradient Descent Backpropagation) é bastante básica e pode não ser adequada para problemas mais complexos ou para conjuntos de dados com muitas variáveis.

Conf4 (trains): Embora não seja tão bom quanto Conf1, Conf4 tem um desempenho bastante decente com uma precisão de 74.59%. A função de treino trains (Scaled Conjugate Gradient Backpropagation) é conhecida por ser eficiente e rápida para muitos problemas.

A função de ativação influencia o desempenho?

As funções de ativação influenciam o desempenho?

Conf1	1	10 30 15	tansig,tansig,tansig, purelin	trainlm	dividerand = {0.975, 0.01, 0.015}	81.44%	18.56%	73.33%
Conf2	1	10 30 15	tansig,logsig,tansig,purelin	trainlm	dividerand = {0.975, 0.01, 0.015}	81.07%	18.93%	65.56%
Conf3	1	10 30 15	tansig,logsig,softmax,purelin	trainlm	dividerand = {0.975, 0.01, 0.015}	75.72%	24.28%	68.89%
Conf4	1	10 30 15	tansig,tansig,purelin, purelin	trainlm	dividerand = {0.975, 0.01, 0.015}	80.59%	19.41%	65.56%

O estudo comparativo das configurações de redes neurais artificiais revelou que as funções de ativação desempenham um papel significativo no seu desempenho. A configuração que utilizou a combinação de funções tansig, tansig, tansig, purelin obteve o melhor resultado, com uma precisão de 81.44%.

No entanto, mesmo a configuração com o desempenho mais baixo, utilizando as funções tansig, logsig, softmax, purelin, alcançou uma precisão de 75.72%. Isso demonstra que, embora a escolha das funções de ativação influencie o desempenho, a diferença entre elas não foi muito significativa, com uma variação máxima de apenas 6%.

Portanto, enquanto algumas combinações de funções de ativação podem ser ligeiramente mais adequadas para o problema em questão, é importante considerar outros fatores, como a complexidade do problema e a natureza dos dados, ao selecionar a configuração ótima da rede neural.

A divisão de exemplos pelos conjuntos influencia o desempenho?

A divisão de exemplos pelos conjuntos influencia o desempenho?									
Conf1	1	10 30 15	tansig,tansig,tansig, purelin	trainlm	dividerand = {0.975, 0.01, 0.015}		81.44%	18.56%	73.33%
Conf2	1	10 30 15	tansig,tansig,tansig, purelin	trainlm	dividerand = {0.99, 0.00, 0.01}		76.56%	23.44%	73.40%
Conf3	1	10 30 15	tansig,tansig,tansig, purelin	trainlm	dividerand = {0.8, 0.1, 0.1}		79.46%	20.54%	71.48%
Conf4	1	10 30 15	tansig,tansig,tansig, purelin	trainlm	dividerand = {0.8, 0.15, 0.05}		79.33%	20.67%	68.06%

O estudo comparativo teve uma discrepância média de 5%.

A conf1 obteve o melhor resultado global.

A conf2 obteve a melhor precisão de teste.

Concluimos que este estudo não é muito conclusivo, uma vez que tanto a precisão global e precisão de teste tiveram uma discrepância mínima.

3.3. Utilização do ficheiro TEST

A função Test(), é responsável por testar a capacidade de generalização da melhor rede neural treinada anteriormente utilizando o conjunto de dados de teste.

Carrega os dados de teste do arquivo 'Test.csv'.

Divide os dados de teste em entrada (X) e alvo (y).

Carrega a melhor rede neural treinada, que foi previamente salva em um arquivo .mat.

Utiliza a rede neural para fazer previsões no conjunto de dados de teste.

Calcula a precisão total das previsões comparando-as com os resultados reais.

Exibe a precisão total no conjunto de dados de teste.

Essencialmente, a função Test() avalia o desempenho da melhor rede neural treinada no conjunto de dados de teste não visto durante o treinamento, fornecendo uma medida de sua capacidade de generalização para casos não observados anteriormente.

Utilização do TEST para as 3 redes com melhor desempenho

	Número de camadas escondidas	Número de neurónios	Funções de ativação	Função de treino	Divisão dos exemplos		Precisão Global	Erro
99.5082.mat	3	20 20 20	tansig, softmax,logsig,purelin	trainlm	dividerand = {0.99, 0.0, 0.01}		70.0%	30.0%
99.3443.mat	3	20 20 20	tansig, softmax,logsig,purelin	trainlm	dividerand = {0.99, 0.0, 0.01}		90.0%	10.0%
99.8361.mat	3	20 20 20	tansig, softmax,logsig,purelin	trainlm	dividerand = {0.99, 0.0, 0.01}		70.0%	30.0%

Com base nos resultados apresentados nos arquivos “99.5082.mat”, “99.3443.mat” e “99.8361.mat”, podemos fazer as seguintes observações:

Arquivo “99.5082.mat”:A rede neural treinada neste arquivo obteve uma precisão de 70% nos casos avaliados. Isso significa que, para os dados específicos usados para avaliação, a rede acertou corretamente 70% das vezes.

Arquivo “99.3443.mat”: A rede salva no arquivo “99.3443.mat” teve um desempenho ainda melhor, com uma precisão de 90%. Essa foi a rede com o melhor desempenho entre as três avaliadas.

Arquivo “99.8361.mat”: Neste arquivo, a rede também obteve uma precisão de 70%. Isso significa que, para os dados de teste específicos usados aqui, a rede acertou corretamente 70% das vezes possíveis.

4. Conclusão

Ao escolher o conjunto de dados "Stroke" e realizar uma preparação meticulosa dos dados, incluindo a implementação de estratégias como o raciocínio baseado em casos (CBR) e retrieve, conseguimos desenvolver um modelo preciso e confiável.

Por exemplo, ao lidar com valores ausentes, o uso do CBR nos permitiu preencher lacunas nos dados, melhorando a integridade do conjunto de dados e aumentando a confiabilidade das previsões.

Além disso, ao analisar diferentes arquiteturas de redes neurais, observamos como escolhas específicas, como funções de ativação e métodos de treinamento, podem influenciar diretamente a precisão do modelo.

Por exemplo, a implementação da função de treinamento "trainlm" resultou em uma precisão de 81,44%, destacando sua eficácia para este problema específico.

Esses exemplos concretos ilustram o potencial transformador da análise de dados na área da saúde, capacitando profissionais médicos com ferramentas avançadas para oferecer diagnósticos mais rápidos e precisos, além de intervenções mais eficazes, salvando vidas e melhorando a qualidade de vida dos pacientes.