Utilização de Redes Neuronais na Previsão de Doença Hepática

Rafael Sêco

Instituto Superior de Engenharia do Porto

Rua Dr. António Bernardino de Almeida, 431, Porto, Portugal

1231392@isep.ipp.pt

Resumo: As redes neuronais são um ramo da inteligência artificial, que se baseia no funcionamento do cérebro humano e do sistema nervoso, de maneira a processar informações. Assim, as redes neuronais têm capacidade de identificar padrões nos dados analisados, com recurso a programação. Deste modo, as redes neuronais têm diversas aplicações, em diversas áreas, nomeadamente em Engenharia Biomédica, que concilia a saúde e a engenharia. Um exemplo notável dos benefícios da aplicação de redes neuronais é a sua utilização em diagnóstico de doenças hepáticas. Assim, as redes neuronais podem ser consideradas uma ferramenta essencial para a deteção precoce de doenças, contribuindo para a melhoria da qualidade de vida dos pacientes e da eficácia não só do Serviço Nacional de Saúde, mas também de todos os hospitais privados.

Palavras-chave: Doença Hepática, Neuralnet, Nnet, Support Vector Machine, Extreme Learning Machine, Gradient Boosting Machine.

1 Introdução

O fígado é um órgão importante para a manutenção do equilíbrio do organismo, uma vez que filtra o sangue e elimina diversas substâncias tóxicas, bem como produz proteínas e triglicerídeos [1]. Apesar da grande capacidade de regeneração do fígado, este é suscetível a uma grande variedade de doenças, onde se destaca a doença hepática. A incidência crescente da doença hepática representa um desafio significativo para os sistemas de saúde em todo o mundo, requerendo abordagens inovadoras e eficazes para o seu diagnóstico e respetivo tratamento.

Neste contexto, as redes neuronais emergem como uma ferramenta promissora na compreensão e no combate à doença hepática. Ao utilizar algoritmos complexos, as redes neuronais têm a capacidade de analisar grandes conjuntos de dados, identificar certos padrões e fornecer informações valiosas para a prevenção, diagnóstico e tratamento precoces da doença hepática. Esta interseção entre a biologia do fígado e a inteligência artificial representa uma

fronteira fascinante na área biomédica, oferecendo perspetivas inovadoras para enfrentar os desafios associados a esta doença.

Ao longo deste trabalho, são abordadas diversas técnicas e utilizados vários modelos de maneira a efetuar uma previsão de pacientes com a doença hepática, sendo eles os modelos *neuralnet*, *nnet*, *Suport Vector Machine, Gradient Boosting Machine* e *Extreme Learning Machine*.

Os modelos *neuralnet* e *nnet* permitem a implementação de redes neuronais artificiais e são ferramentas essenciais na aplicação a um conjunto de dados não-lineares. Estes modelos são personalizáveis, uma vez que é possível mudar os hiperparâmetros referentes a cada um deles de maneira a aumentar a sua flexibilidade na adaptação ao conjunto de dados com que se é trabalhado. Os hiperparâmetros do modelo *neuralnet* incluem uma fórmula descritiva do que se pretende analisar, bem como dados para efetuar o treino do modelo. Além disso, é possível especificar o tamanho pretendido da camada invisível, podendo existir várias camadas. Quanto à saída deste modelo, existe apenas um *output* [2]. Quanto ao modelo *nnet*, e para além da fórmula descritiva e do conjunto de dados de treino, também é possível especificar o número de neurónios na camada invisível, ainda que apenas aceite uma só camada. Também é possível ajustar o decaimento nos pesos relativos a cada variável. Ao nível de saída, este modelo produz 2 *outputs* [3].

O modelo *Support Vector Machine* tem como objetivo encontrar um plano que consiga separar os dados em diferentes classes (hiperplano). Este modelo é ideal quando os dados se conseguem agrupar na sua maioria entre dados da mesma classe. Se uma linha reta conseguir dividir estas duas classes de dados, então diz-se que estes pontos são linearmente separáveis [4].

O modelo *Gradient Boosting Machine* cria a expectativa de alcançar um modelo solidificado a partir de um modelo mais fraco. O modelo, à medida que avança na classificação, ajusta o modelo inicial calculando a diferença entre a previsão do modelo atual e o valor real. Assim, a sua aprendizagem é gradual e ajustada na direção em que estas diferenças calculadas têm mais impacto [5].

O modelo *Extreme Learning Machine* é um algoritmo de rede neuronal *feedforward* que tende a fornecer um bom desempenho de generalização a uma velocidade extremamente rápida [6]. Para utilizar este modelo será necessário fazer uma adaptação e transformar o conjunto de dados numa matriz para ser analisado.

2 Métodos

Neste capítulo serão explicados os métodos aplicados, desde a escolha do *dataset* e a sua limpeza, bem como a aplicação deste em diferentes redes neuronais.

2.1 *Dataset* utilizado e respetiva limpeza

O conjunto de dados escolhido está presente na plataforma *Kaggle*, com o título *Liver Disease Patient Dataset 30K train data* [7]. Este conjunto de dados contém dois ficheiros em formato *csv*, sendo que um está direcionado para o treino e o outro indica que deve ser utilizado para o teste do modelo de rede neuronal.

Ao abrir o ficheiro, foi possível observar que o nome das colunas era demasiado longo e continha espaços em branco. Assim, o nome das colunas foi trocado para algo mais simples, sendo a lista de alterações a seguinte:

i. *Age of the patient* alterou para *age*
ii. *Gender of the patient* alterou para *gender*
iii. *Total Bilirubin* alterou para *Tot_Bilirubin*
iv. *Direct Bilirubin* alterou para *Dir_Bilirubin*
v. *Alkphos Alkaline Phosphotase* alterou para *Alkphos*
vi. *Sgpt Alamine Aminotransferase* alterou para *Sgpt*
vii. *Sgot Aspartate Aminotransferase* alterou para *Sgot*
viii. *Total Protiens* alterou para *Tot_Protiens*
ix. *ALB Albumin* alterou para *Albumin*
x. *A/G Ratio Albumin and Globulin Ratio* alterou para *ag_Ratio*

O ficheiro foi, depois, importado para o R Studio através da função *read_csv*, observando-se que o ficheiro continha 30691 linhas de dados. No entanto, num ficheiro com uma grande quantidade de dados, é necessário observar se existem dados em falta. Como é possível observar na figura 1, existem muitos dados em falta, cujo valor atribuído é *NA*.

```
> ValoresEmFalta <- colSums(is.na(DadosTreino))
> print(ValoresEmFalta)
        age      gender Tot_Bilirubin Dir_Bilirubin      Alkphos         Sgpt         Sgot  Tot_Protiens      Albumin 
          2         902          648          561          796          538          462          463          494 
 ag_Ratio       Result 
        559            0 
```

Figura 1 - Somatório de valores em falta em relação a cada coluna

De seguida, foram ignoradas as observações com 2 ou mais valores em falta, utilizando uma variável auxiliar para verificar se essa condição é verdadeira. Na figura 2, está demonstrado um exemplo ilustrativo do código utilizado para esse efeito. O conjunto de dados que poderemos utilizar está agora reduzido a 29408 observações, sendo que esta ação de limpeza de observações com dados em falta equivale a 1283 observações com 2 ou mais valores em falta.

```
auxiliar <- apply(DadosTreino,1,function(x)sum(is.na(x)))
DadosTreino <- DadosTreino[which(auxiliar<2),]
```

Figura 2 - Criação de uma variável auxiliar para limpar o conjunto de dados

Após esta ação, ainda restavam cerca de 2250 linhas de dados com dados em falta, divididos entre as diversas colunas, como observável na figura 3. No entanto, estas figuram muitas linhas com dados em falta e, para não influenciar

o modelo ou colocar muitos dados que não correspondem bem à realidade, foram ignoradas estas observações com dados em falta.

```
> ValoresEmFalta <- colSums(is.na(DadosTreino))
> print(ValoresEmFalta)
          age        gender Tot_Bilirubin Dir_Bilirubin       Alkphos          Sgpt          Sgot  Tot_Protiens 
            2           508           229           169           358           204           133           170 
      Albumin      ag_Ratio        Result 
          168           309             0 
>
>
> DadosTreino <- na.omit(DadosTreino)
> print(summary(DadosTreino))
      age          gender          Tot_Bilirubin    Dir_Bilirubin       Alkphos            Sgpt             Sgot       
 Min.   : 4.00   Length:27158       Min.   : 0.400   Min.   : 0.100   Min.   :  63.0   Min.   :  10.00   Min.   :  10.0  
 1st Qu.:33.00   Class :character   1st Qu.: 0.800   1st Qu.: 0.200   1st Qu.: 175.0   1st Qu.:  23.00   1st Qu.:  26.0  
 Median :45.00   Mode  :character   Median : 1.000   Median : 0.300   Median : 209.0   Median :  36.00   Median :  42.0  
 Mean   :44.13                      Mean   : 3.408   Mean   : 1.542   Mean   : 290.1   Mean   :  81.28   Mean   : 112.1  
 3rd Qu.:55.00                      3rd Qu.: 2.700   3rd Qu.: 1.300   3rd Qu.: 298.0   3rd Qu.:  62.00   3rd Qu.:  88.0  
 Max.   :90.00                      Max.   :75.000   Max.   :19.700   Max.   :2110.0   Max.   :2000.00   Max.   :4929.0  
  Tot_Protiens      Albumin         ag_Ratio          Result      
 Min.   :2.700   Min.   :0.900   Min.   :0.3000   Min.   :1.000  
 1st Qu.:5.800   1st Qu.:2.600   1st Qu.:0.7000   1st Qu.:1.000  
 Median :6.600   Median :3.100   Median :0.9000   Median :1.000  
 Mean   :6.473   Mean   :3.124   Mean   :0.9436   Mean   :1.283  
 3rd Qu.:7.200   3rd Qu.:3.700   3rd Qu.:1.1000   3rd Qu.:2.000  
 Max.   :9.600   Max.   :5.500   Max.   :2.8000   Max.   :2.000  
> ValoresEmFalta <- colSums(is.na(DadosTreino))
> print(ValoresEmFalta)
          age        gender Tot_Bilirubin Dir_Bilirubin       Alkphos          Sgpt          Sgot  Tot_Protiens 
            0             0             0             0             0             0             0             0 
      Albumin      ag_Ratio        Result 
            0             0             0 
```

Figura 3 - Número de linhas com dados em falta antes e depois de as ignorar

Após a remoção de todas as linhas com dados em falta, existiam duas observações cuja variável *Sgot* estava excessivamente alta, então estas foram eliminadas (figura 4). Estes dados poderiam ter sido introduzidos graças a um erro, potencialmente, reduzindo-se assim o valor máximo que esta variável pode tomar.

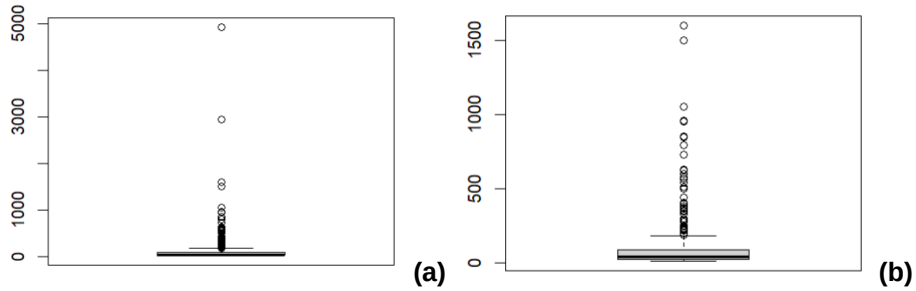


*Figura 4 - **(a)** Gráfico referente aos valores de Sgot antes da limpeza; **(b)** Gráfico da variável Sgot após a remoção dos dois valores excessivamente altos*

Uma vez que os *outliers* foram eliminados e as linhas com dados em falta foram apagadas, podemos iniciar o processo de transformação de características e de normalização dos dados.

Na variável *Result* podem estar dois números: 1 e 2. O primeiro considera que a observação é de um paciente de doença hepática, ao passo que número 2 indica que não é um caso desta doença. Assim, no âmbito de *feature engineering*, esta coluna de valores foi alterada para uma gama de valores binária,

0 e 1, em que 0 representa um caso de que não é paciente e 1 representa o caso de um paciente de doença hepática.

Também foi criada uma coluna, com o nome *Class*, cujo significado é a classificação por extenso (*not_patient* ou *patient*), para que que os resultados sejam mais facilmente comparados no futuro. Além disso, existe uma coluna *gender*, que dá informações sobre o género do paciente que está a ser observado. No entanto, esta é uma variável categórica nominal, uma vez que pode tomar os valores de *Male* ou *Female*. Assim, foi necessário criar duas colunas, cada uma referente a um género, em que os valores são variáveis numéricas discretas, para que possam ser compreendidas pelo modelo.

Após estas operações de modificação de variáveis, criação de uma variável Class e de transformação de variáveis categóricas em numéricas, obtemos um conjunto de dados com 14 variáveis, como é possível observar na figura 5.

	age	gender	Tot_Bilirubin	Dir_Bilirubin	Alkphos	Sgpt	Sgot	Tot_Protiens	Albumin	ag_Ratio	Result	Class	Female	Male
1	65	Female	0.7	0.1	187	16	18	6.8	3.3	0.90	1	patient	1	0
2	62	Male	10.9	5.5	699	64	100	7.5	3.2	0.74	1	patient	0	1
3	62	Male	7.3	4.1	490	60	68	7.0	3.3	0.89	1	patient	0	1
4	58	Male	1.0	0.4	182	14	20	6.8	3.4	1.00	1	patient	0	1
5	72	Male	3.9	2.0	195	27	59	7.3	2.4	0.40	1	patient	0	1
6	46	Male	1.8	0.7	208	19	14	7.6	4.4	1.30	1	patient	0	1
7	29	Female	0.9	0.3	202	14	11	6.7	3.6	1.10	1	patient	1	0
8	17	Male	0.9	0.3	202	22	19	7.4	4.1	1.20	0	not_patient	0	1

Figura 5 - Exemplo do conjunto de dados após feature engineering

De seguida, os dados da variável *Class* foram transformados em *factor* para futuramente serem mais fáceis de serem analisados. Na figura 6 está um exemplo dos passos efetuados para realizar essa mudança. No início, todos os valores da variável *Class* eram caracteres e foram modificados para *factors*.

```
> print(class(DadosTreino$Class))
[1] "character"
> #passar a 'is_patient' para factor
> DadosTreino["Class"] <- factor(DadosTreino$Class)
> print(class(DadosTreino$Class))
[1] "factor"
```

Figura 6 - Passagem da variável 'Class' para 'factor'

Após esta análise e transformação do conjunto de dados recorrendo a *feature engineering*, o próximo passo será realizar um *scaling*, isto é, normalizar o *dataset* para garantir que as variáveis numéricas contínuas têm a mesma escala. No entanto, as variáveis *gender, Result, Class, Female* e *Male* ficaram de fora desta normalização porque são variáveis categóricas nominais ou numéricas discretas. Na figura 7 é possível observar como se encontra o conjunto de dados normalizado.

	age	Tot_Bilirubin	Dir_Bilirubin	Alkphos	Sgpt	Sgot	Tot_Protiens	Albumin	ag_Ratio
1	0.7093023	0.004021448	0.000000000	0.060576453	0.0030150754	0.0050314465	0.59420290	0.5217391	0.240
2	0.6744186	0.140750670	0.275510204	0.310698583	0.0271356784	0.0566037736	0.69565217	0.5000000	0.176
3	0.6744186	0.092493298	0.204081633	0.208597948	0.0251256281	0.0364779874	0.62318841	0.5217391	0.236
4	0.6279070	0.008042895	0.015306122	0.058133854	0.0020100503	0.0062893082	0.59420290	0.5434783	0.280
5	0.7906977	0.046916890	0.096938776	0.064484612	0.0085427136	0.0308176101	0.66666667	0.3260870	0.040
6	0.4883721	0.018766756	0.030612245	0.070835369	0.0045226131	0.0025157233	0.71014493	0.7608696	0.400
7	0.2906977	0.006702413	0.010204082	0.067904250	0.0020100503	0.0006289308	0.57971014	0.5869565	0.320
8	0.1511628	0.006702413	0.010204082	0.067904250	0.0060301508	0.0056603774	0.68115942	0.6956522	0.360
9	0.5930233	0.004021448	0.005102041	0.110893991	0.0216080402	0.0301886792	0.59420290	0.5434783	0.280
10	0.6162791	0.002680965	0.000000000	0.071812408	0.0206030151	0.0308176101	0.46376812	0.3913043	0.200

Figura 7 - Conjunto de dados cujas variáveis numéricas contínuas foram normalizadas

A este conjunto de dados foram adicionadas as variáveis numéricas discretas *Female* e *Male*, juntamente com a variável categórica nominal *Class* (figura 8). Este conjunto de dados será utilizado em ambos os modelos *neuralnet* e *nnet*.

	age	Tot_Bilirubin	Dir_Bilirubin	Alkphos	Sgpt	Sgot	Tot_Protiens	Albumin	ag_Ratio	Female	Male	Class
1	0.7093023	0.004021448	0.000000000	0.060576453	0.0030150754	0.0050314465	0.59420290	0.5217391	0.240	1	0	patient
2	0.6744186	0.140750670	0.275510204	0.310698583	0.0271356784	0.0566037736	0.69565217	0.5000000	0.176	0	1	patient
3	0.6744186	0.092493298	0.204081633	0.208597948	0.0251256281	0.0364779874	0.62318841	0.5217391	0.236	0	1	patient
4	0.6279070	0.008042895	0.015306122	0.058133854	0.0020100503	0.0062893082	0.59420290	0.5434783	0.280	0	1	patient
5	0.7906977	0.046916890	0.096938776	0.064484612	0.0085427136	0.0308176101	0.66666667	0.3260870	0.040	0	1	patient
6	0.4883721	0.018766756	0.030612245	0.070835369	0.0045226131	0.0025157233	0.71014493	0.7608696	0.400	0	1	patient
7	0.2906977	0.006702413	0.010204082	0.067904250	0.0020100503	0.0006289308	0.57971014	0.5869565	0.320	1	0	patient
8	0.1511628	0.006702413	0.010204082	0.067904250	0.0060301508	0.0056603774	0.68115942	0.6956522	0.360	0	1	not_patient
9	0.5930233	0.004021448	0.005102041	0.110893991	0.0216080402	0.0301886792	0.59420290	0.5434783	0.280	0	1	patient
10	0.6162791	0.002680965	0.000000000	0.071812408	0.0206030151	0.0308176101	0.46376812	0.3913043	0.200	0	1	patient

Figura 8 - Exemplo do conjunto de dados após tratamento dos dados

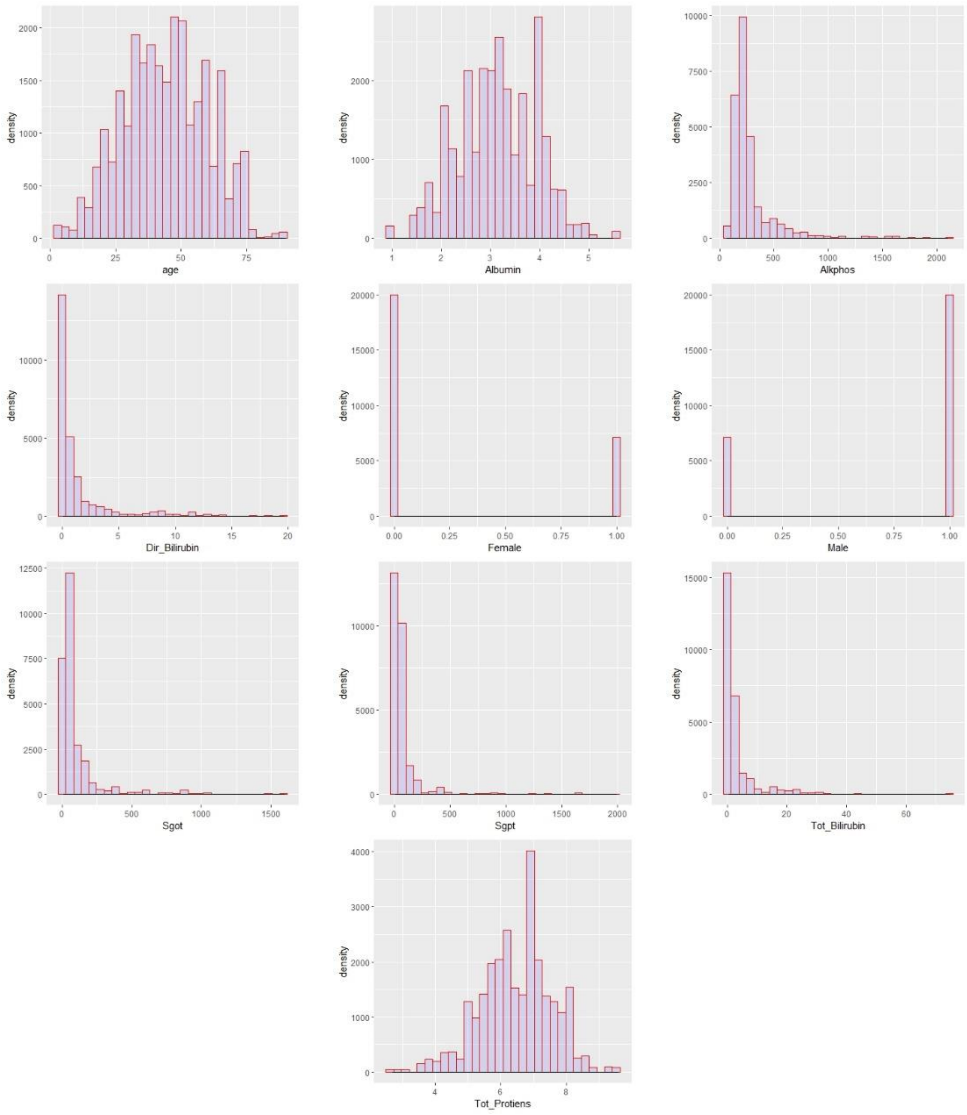
Para aumentar a facilidade de escolha dos dados para uso nos modelos de redes neuronais, foram criados dois *dataframes* diferentes: o primeiro referente inteiramente a casos cujas observações são de pacientes e o segundo a casos que não são de pacientes, como demonstrado na figura 9. Neste momento, obtemos um total de pacientes igual a 19391 (71,63%) e 7680 casos de não pacientes (28,37%), representando percentagens particularmente desniveladas.

```
liverTreino_patient <- liverTreino[liverTreino$Class == "patient",]
liverTreino_not_patient <- liverTreino[liverTreino$Class == "not_patient",]
```

Figura 9 - Separação das observações consoante a variável Class

Após os tratamentos dos dados, é possível observar pelos histogramas representados na tabela 1 que existe uma maior incidência de observações referentes ao sexo masculino, bem como que existem observações de várias variáveis (*Alkphos, Sgot, Sgpt, Dir_Bilirubin* e *Tot_Bilirubin*) que se encontram muito distantes do seu valor médio.

Tabela 1 - Histogramas relativos à distribuição dos dados



2.2 Descrição das redes *neuralnet* e *nnet*

Em ambas as redes foram utilizados parâmetros de entrada referentes às variáveis descritas anteriormente (*age, Tot_Bilirubin, Dir_Bilirubin, Alkphos, Sgpt, Sgot, Tot_Protiens, Albumin, ag_Ratio, Female* e *Male*), sendo a variável de saída a variável *Class*, que pode tomar o valor de *patient* ou *not_patient*. Na figura 10 está representada a arquitetura generalizada da rede *neuralnet.*

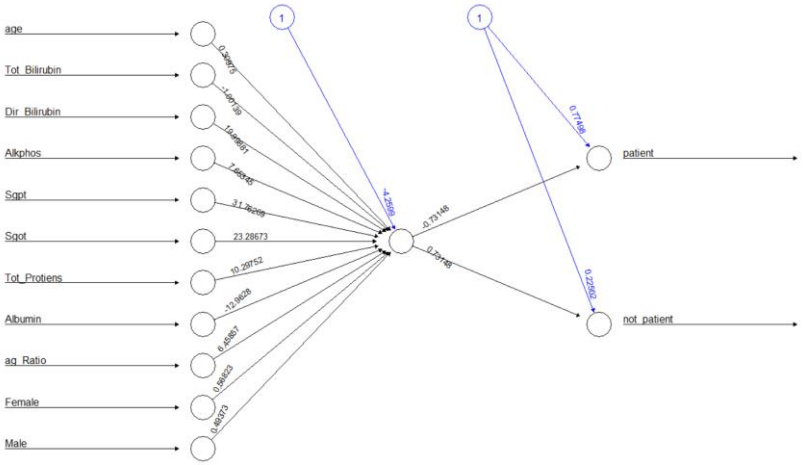


Figura 10 - Arquitetura generalizada da rede neuralnet

A rede *neuralnet*, de uma forma genérica, aceita parâmetros como uma descrição simbólica do modelo (uma fórmula), um *dataframe* com os valores especificados na fórmula, o número de neurónios presentes na camada invisível, um valor lógico que indica se a camada de saída deve ser linear, um algoritmo para treinar a rede neuronal (*backprop* ou *rprop*), uma taxa de aprendizagem do algoritmo, número de repetições, limiar de convergência (*threshold*), pesos iniciais de cada variável, entre outros parâmetros. No capítulo referente aos resultados, serão avaliadas e discutidas as influências que estes parâmetros terão nos resultados finais.

Quanto à rede *nnet*, a sua arquitetura está representada genericamente na figura 11. Foi escolhido um tamanho de 10 unidades na camada não-visível. A rede *nnet* aceita parâmetros de entrada como a fórmula simbólica dos dados e respetivo *dataframe*, pesos de cada variável, número de neurónios presentes na camada invisível, número máximo de iterações, entre outros parâmetros. Semelhante à rede *neuralnet*, a influência dos parâmetros de entrada da rede *nnet* nos resultados serão estudados no capítulo de resultados.

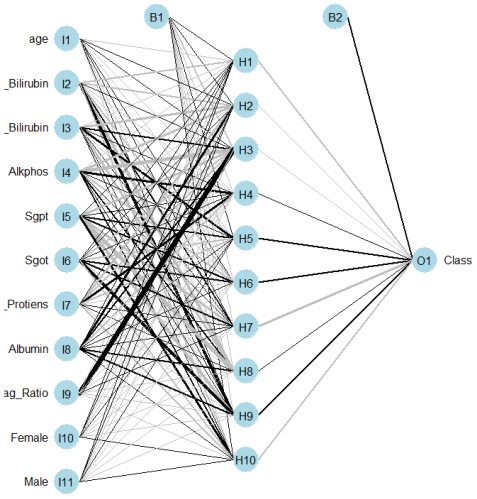


Figura 11 - Arquitetura genérica da rede nnet com 10 unidades na camada não-visível

2.3 Aplicação de métodos de avaliação

Ao interagir com um modelo de aprendizagem, é necessário avaliar as suas fraquezas e os seus pontos fortes, bem como a sua performance global. A avaliação do modelo é importante para avaliar a exatidão deste durante as fases iniciais da investigação e também desempenha um papel na monitorização do mesmo [8]. Para tal, podem ser aplicados diferentes métodos, tais como o método *hold-out* e o método de *cross-validation*.

O método de *hold-out* surge neste âmbito como um método de divisão de dados. De uma forma geral, uma percentagem grande dos dados é usada para treinar o modelo, enquanto que o restante é utilizado para o testar. O objetivo desta técnica é selecionar o melhor modelo com base na sua precisão no conjunto de dados de teste e compará-lo com outros modelos [9]. Este método fornecerá características de desempenho como exatidão, sensibilidade, sensitividade e matriz de confusão. O conjunto de dados será dividido de duas maneiras diferentes, de acordo com as seguintes percentagens:

i. *Training Set* com 80% dos dados totais e 20% para testar os modelos;
ii. *Training Set* com 70% dos dados totais e 30% para testar os modelos.

Por outro lado, o método de *cross-validation* aparece como maneira de fornecer uma estimativa imparcial do erro por generalização. Este método ajuda a obter uma medida mais clara de como é que o nosso modelo se desempenha e permite ajustar os parâmetros de entrada do modelo. Assim, em vez de se

treinar o modelo com apenas um conjunto de dados, podemos treinar com vários subconjuntos de dados. Na figura 12 encontra-se um exemplo ilustrativo de como é possível dividir o conjunto de dados:

i. *Testing Set* com 20% dos dados totais
ii. *Training Set* com 80% dos dados totais, sendo que 10% destes estão alocados para a validação e o restante para o treino do modelo

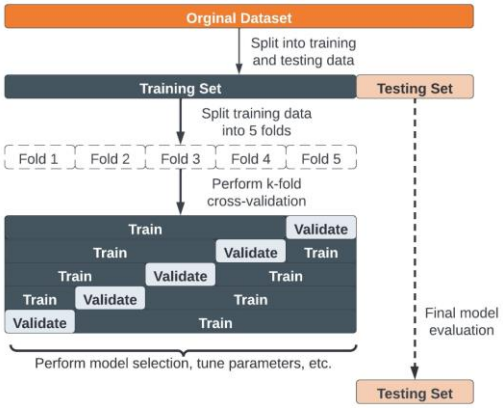


Figura 12 - Exemplo da divisão a efetuar no conjunto de dados para o método de cross-validation

Implementou-se o *10-fold cross-validation*, balanceando o conjunto de dados. No total, existiam 7680 observações possíveis de analisar quanto a casos de não-pacientes então, de maneira a balancear o conjunto de dados, alterou-se a ordem das linhas do conjunto de dados referentes a pacientes, para que se pudessem escolher 7680 observações diferentes de cada vez que o programa efetuasse este passo. De seguida, reservaram-se 20% dos dados disponíveis para testar o modelo futuramente e retiraram-se esses dados para que não estivessem presentes na fase de treino e validação. Na figura 13 encontra-se um exemplo do código efetuado para realizar este passo.

```
liverTreino_patient <- liverTreino_patient[sample(1:nrow(liverTreino_patient)),]
liverTreino_not_patient <- liverTreino_not_patient[sample(1:nrow(liverTreino_not_patient)),]

teste <- rbind(liverTreino_patient[6142:7680,],liverTreino_not_patient[6142:7680,])

liverTreino_patient <- liverTreino_patient[-c(6142:7680),]
liverTreino_not_patient <- liverTreino_not_patient[-c(6142:7680),]
```

Figura 13 - Reserva dos dados para a fase de teste

Uma vez que queremos realizar um *10-fold cross-validation*, a intenção será guardar várias métricas obtidas em diferentes vetores, sendo estas a exatidão e o erro quadrático médio. Também é relevante guardar o modelo treinado para que seja possível testar futuramente o melhor e o pior modelo da fase de validação. Além disso, é interessante saber qual o estado do processo e, por isso, também foi criada uma barra de progresso (figura 14).

```
k <- 10
erro_cv <- NULL
exatidao <- NULL
vetor <- NULL

#barra de progresso:
library(plyr)
pbar <- create_progress_bar('text')
pbar$init(k)
```

Figura 14 - Criação de vetores para guardar métricas e barra de progresso

Foi implementado um ciclo *for*, para que repetisse o processo exatamente 10 vezes. Em cada iteração, reordenava os dados referentes a pacientes para que o modelo fosse treinado e validado com o máximo de observações diferentes. Na variável *dados_cv* encontram-se 12282 observações e uma nova coluna com dados numéricos discretos para que fosse possível calcular o erro quadrático médio. Foram escolhidos 90% destes dados, de forma aleatória, para treinar o modelo e os restantes 10% serviam para realizar a validação do mesmo. Foi implementada uma variável auxiliar para que fosse possível alterar o tamanho da camada invisível, de maneira a estudar a influência do tamanho desta camada nos resultados obtidos. Em cada ciclo, o modelo era construído, guardado e validado, sendo que também eram calculadas as métricas de erro quadrático médio e exatidão. No final do ciclo, a barra de progresso mudaria. A implementação do processo de *10-fold cross-validation* pode ser observado na figura 15.

```
for (i in 1:k){

  liverTreino_patient <- liverTreino_patient[sample(1:nrow(liverTreino_patient)),]
  dados_cv <-rbind(liverTreino_not_patient[1:6141,],liverTreino_patient[1:6141,])
  dados_cv["Binario"] <- ifelse(dados_cv$Class == "not_patient",0,1)

  index <- sample(1:nrow(dados_cv),round(0.9*nrow(dados_cv)))
  treino <- dados_cv[index,]
  validacao <- dados_cv[-index,]

  tamanho <-i
  modelo <- nnet(formula_nn,data=treino,size=tamanho)
  plotnet(modelo)
  previsao <- predict(modelo,validacao[,1:11],type='raw')

  vetor[i] <- list(modelo)

  predicted_size_category <- ifelse(previsao[,1] < 0.5,"not_patient","patient")
  print(mean(predicted_size_category==validacao$Class)) #Testing Accuracy = %


  previsao <- previsao*(max(dados_cv$Binario)-min(dados_cv$Binario))+min(dados_cv$Binario)

  resultados <- (validacao$Binario)*(max(dados_cv$Binario)-min(dados_cv$Binario))+min(dados_cv$Binario)

  erro_cv[i] <- sum((resultados - previsao)^2)/nrow(validacao)
  exatidao[i] <- mean(predicted_size_category==validacao$Class)

  pbar$step()
}
```

Figura 15 - Implementação de 10-fold cross-validation

No final deste ciclo, é possível observar os vetores com as métricas guardadas referentes a cada iteração, bem como calcular a média aritmética do desempenho do modelo geral (figura 16).

```
print("Vetor contendo os valores de erro:")
print(erro_cv)

print("Vetor com valores de exatidão:")
print(exatidao)

print("Media de exatidao da rede nnet com size=10 utilizando cross-validation: ")
print(mean(exatidao))

print("Media de Mean Squared Error da rede nnet com size=10 utilizando cross-validation: ")
print(mean(erro_cv))
```

Figura 16 - Obtenção das métricas guardadas

No final, é possível testar os modelos com melhor e pior desempenhos, utilizando os dados previamente reservados para teste (figura 17) e os modelos treinados que foram guardados após cada iteração.

```
modelo <- vetor[[which.max(exatidao)]]
#modelo<-modelo[modelo[1]]

#print(which.max(exatidao))
teste_final <- predict(modelo,teste[,1:11],type='raw')
predicted_size_category <- ifelse(teste_final[,1] < 0.5,"not_patient","patient")

print("A exatidão do modelo que apresentou melhores resultados na validação é:")
print(mean(predicted_size_category==teste$Class)) #Testing Accuracy = %


modelo <- vetor[[which.min(exatidao)]]
#modelo<-modelo[modelo[1]]

#print(which.max(exatidao))
teste_final <- predict(modelo,teste[,1:11],type='raw')
predicted_size_category <- ifelse(teste_final[,1] < 0.5,"not_patient","patient")

print("A exatidão do modelo que apresentou piores resultados na validação é:")
print(mean(predicted_size_category==teste$Class)) #Testing Accuracy = %
```

Figura 17 - Teste do melhor e do pior modelo, previamente guardados

3 Resultados e Discussão

Neste capítulo serão abordados os resultados obtidos e respetiva discussão, com o objetivo de perceber qual o melhor modelo e a melhor arquitetura da rede para o conjunto de dados que se quer analisar. O conjunto de dados analisado estava balanceado, com o mesmo número de pacientes e de não-pacientes. Existem exceções, todas elas devidamente explicadas.

3.1 Resultados obtidos para o método de *hold-out*

Numa primeira fase foi utilizado o método de *hold-out*, uma vez que é mais simples de estruturar. Ambos os modelos *neuralnet* e *nnet* foram testados, com

o intuito de entender qual a influência que um *dataset* balanceado tem no desempenho destes dois modelos. O fluxograma referente aos processos implementados com *hold-out* encontra-se na figura 18.

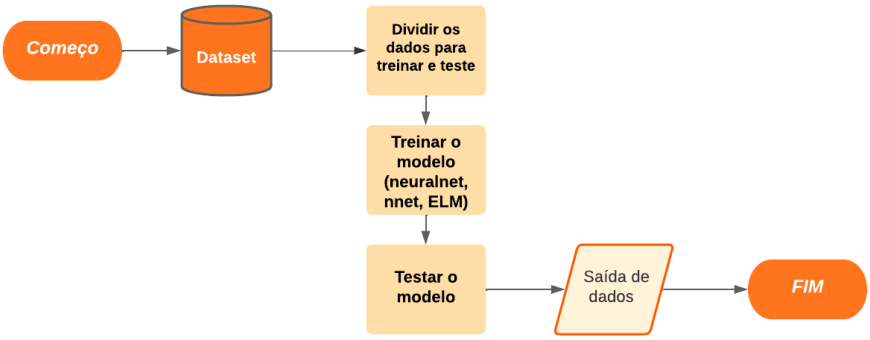


Figura 18 – Fluxograma para o processo implementado no método de hold-out

3.1.1. Influência de um *dataset* balanceado no desempenho do modelo *neuralnet*

O conjunto total de dados, após a sua limpeza, continha 71,63% de casos relativos a pacientes de doença hepática. O restante, 28,37%, era referente a não-pacientes. No gráfico da figura 19 é possível observar a disparidade de valores entre as duas classes.

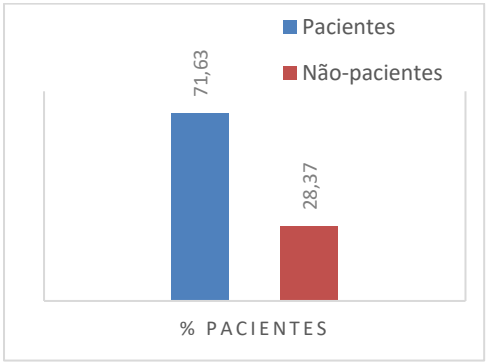


Figura 19 - Relação entre a percentagem de pacientes e a percentagem de não-pacientes

Fez-se a divisão de dados para treino e para teste, sem balancear o conjunto de dados, de forma propositada. Assim, os dados para treino representavam 80% das observações, cerca de 21656 observações, enquanto que o restante foi separado para teste. O modelo *neuralnet*, na sua forma mais simples e básica, apresentou uma exatidão de 72,3%, especificidade de 75,5% e sensibili-

dade de 53,6%. Contudo, a matriz de confusão retrata uma realidade esperada, uma vez que o modelo treinou com tantos exemplos de casos relativos a pacientes, que quando apareceram casos contrários demonstrou uma grande fraqueza em distingui-los bem. Na figura 20 encontra-se a matriz de confusão para este teste. A percentagem de exatidão exibida deve-se ao facto de haver muitas observações para teste e grande parte referente a pacientes de doença hepática, o que ajuda a subir essa métrica.

	Predicted	
Real	not_patient	patient
not_patient	367	1157
patient	317	3574

Figura 20 - Matriz de confusão dos resultados para um dataset não balanceado no modelo neuralnet

No entanto, urge a necessidade de balancear o conjunto de dados para aumentar a capacidade do modelo em generalizar melhor e conseguir distinguir ambos os casos da melhor forma. Assim, 6144 (cerca de 80%) das observações de não-pacientes foram colocadas em conjunto com o mesmo número de observações relativas a pacientes num mesmo conjunto de treino. Para o conjunto de teste, foram colocados os restantes 1536 (cerca de 20%) casos de não-pacientes juntamente com o mesmo número de observações de pacientes num conjunto de treino. Na figura 21 encontra-se representado o pensamento efetuado para realizar a operação de balanceamento do *dataset*.

```
#Treinar de forma balanceada com peso de 80%:
liver_train <- rbind(liverTreino_patient[1:6144,],liverTreino_not_patient[1:6144,])
liver_teste <- rbind(liverTreino_patient[6145:7680,],liverTreino_not_patient[6145:7680,])
```

Figura 21 - Balanceamento do dataset com o mesmo número de dados de pacientes e não-pacientes

Assim, utilizando o formato mais básico do modelo *neuralnet*, obteve-se uma exatidão de 70,8%, especificidade de 80,1% e sensibilidade de 65,9%. A matriz de confusão não se encontra tão desfigurada, contudo o modelo assinalou muitas observações como *not_patient* quando na verdade deveria ser *patient* (figura 22). Apesar de obter uma exatidão ligeiramente inferior ao teste anterior, o modelo apresenta uma melhor capacidade de generalizar e deverá ser utilizado um conjunto de dados balanceado.

	Predicted	
Real	not_patient	patient
not_patient	1324	212
patient	684	852

Figura 22 - Matriz de confusão dos resultados para um dataset balanceado com o modelo neuralnet

3.1.2. Influência de um dataset balanceado no desempenho do modelo *nnet*

Repetindo o processo anterior, voltou-se a colocar o conjunto de dados não balanceado para proceder ao estudo da influência que este poderá causar no desempenho do modelo *nnet*.

Este modelo, com 10 unidades na camada invisível, e utilizando um conjunto de dados não balanceado como o da figura 19 previamente mencionado, alcançou uma exatidão de 77,4%, especificidade de 81,3% e sensibilidade de 63%. A matriz de confusão obtida, presente na figura 23, demonstra que ainda existiram muitos casos em que o modelo previu que a observação pertencia à *Class patient* quando na verdade era *not_patient.* Este é um aspeto que pode ser melhorado ao utilizar um conjunto de dados balanceado.

	Predicted	
Real	not_patient	patient
not_patient	728	796
patient	427	3464

Figura 23 - Matriz de confusão dos resultados para um dataset não balanceado com o modelo nnet

Após ser efetuado o mesmo balanceado do conjunto de dados como no ponto 3.1.1., o modelo *nnet* com 10 unidades na camada invisível alcançou uma exatidão de 80%, uma especificidade de 88,9% e uma sensibilidade de 74,5%. A sua matriz de confusão encontra-se muito mais sólida, apresentando menos casos de resultados errados, como é possível observar na figura 24. É possível notar que o balanceamento do conjunto de dados faz todo o sentido neste modelo e é imperativo que este balanceamento seja implementado para evitar o *overfitting* de dados.

	Predicted	
Real	not_patient	patient
not_patient	1404	132
patient	481	1055

Figura 24 - Matriz de confusão dos resultados para um dataset balanceado com o modelo nnet

3.1.3. Influência das percentagens de treino e teste num *dataset* balanceado

Uma vez que deverá ser utilizado um conjunto de dados balanceado, resta saber qual a percentagem mais acertada de observações para treino e teste. Foram realizadas várias experiências, variando a percentagem de observações alocadas para treinar e testar os dois modelos, *neuralnet* e *nnet*. Na tabela 2 encontram-se organizados os resultados obtidos, na forma de matriz de confusão.

Tabela 2 - Matrizes de confusão obtidas em relação aos modelos e percentagens de treino e teste

	Modelo *neuralnet*	**Modelo *nnet***
Treino: 70% Teste: 30%	Predicted Real not_patient patient not_patient 2003 301 patient 1050 1255	Predicted Real not_patient patient not_patient 2036 268 patient 722 1583
Treino: 80% Teste: 20%	Predicted Real not_patient patient not_patient 1324 212 patient 684 852	Predicted Real not_patient patient not_patient 1404 132 patient 481 1055

Quanto ao modelo *neuralnet* na sua forma mais básica, este não altera a sua exatidão na classificação do conjunto de dados de teste. Contudo, o modelo *nnet* demonstra menos exatidão quando se depara com menos dados para treinar e mais dados para testar, sendo que as percentagens ideais para este modelo são de 80% para treino e 20% para teste.

3.1.4. Aplicação do algoritmo *Extreme Learning Machine*

O algoritmo *Extreme Learning Machine* foi aplicado utilizando a biblioteca *elmNNRcpp*. Ao contrário de outros modelos, este necessita que as entradas de dados para previsão sejam feitas utilizando matrizes.

Numa primeira tentativa, utilizou-se apenas um neurónio na camada invisível para que fosse possível observar a exatidão dos resultados. A tabela de confusão da figura 25 mostra algo peculiar, uma vez que o número de falsos negativos equivale ao número de verdadeiros negativos, bem como o número de falsos positivos é igual ao número de verdadeiros positivos. Assim sendo, a exatidão obtida foi de 50%.

```
                Predicted
Real          not_patient patient
  not_patient         927     612
  patient             927     612
```

Figura 25 - Matriz de confusão para o algoritmo Extreme Learning Machine, com 1 neurónio

Numa segunda tentativa, alterou-se o número de neurónios na camada invisível para 2 e a matriz de confusão melhorou, diminuindo o número de falsos

positivos e de falsos negativos (figura 26). Ao aumentar o tamanho desta camada, aumenta também a capacidade preditiva do algoritmo, atingindo aproximadamente 56% de exatidão.

		Predicted	
Real		not_patient	patient
	not_patient	891	648
	patient	712	827

Figura 26 - Matriz de confusão para o algoritmo Extreme Learning Machine, com 2 neurónios

Uma vez que a exatidão do algoritmo tende a aumentar com o aumento do número de neurónios, foi utlizada a técnica de *cross-validation* com a mudança do número de neurónios. A experiência encontra-se relatada na secção 3.2.6.

3.2 Resultados obtidos para o método de *cross-validation*

Numa segunda fase foi utilizado o método de *cross-validation*, uma vez que é mais árduo de implementar. Ambos os modelos *neuralnet* e *nnet* foram testados, com o intuito de entender qual a influência que um *dataset* balanceado tem no desempenho destes dois modelos. O fluxograma referente aos processos implementados com *cross-validation* encontra-se na figura 27.

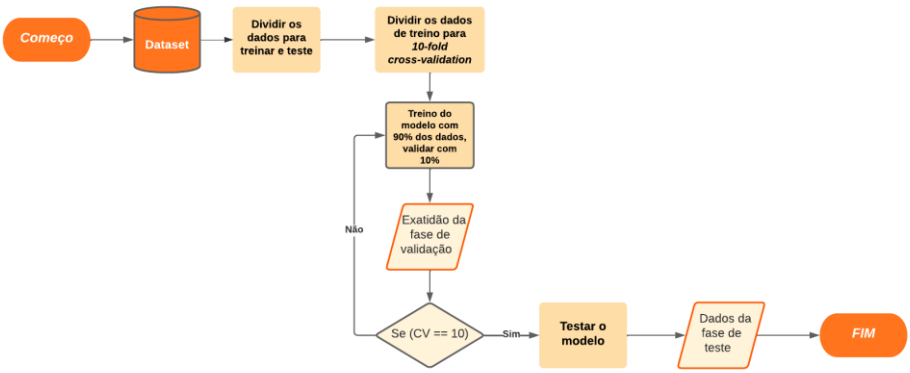


Figura 27 - Fluxograma do processo desenvolvido para a técnica de cross-validation

3.2.1. Aplicação de *10-fold cross-validation* ao modelo *nnet* com camada invisível de tamanho variável

Após a implementação do método de *10-fold cross-validation* referido na secção 2.3. do presente relatório, testou-se este método com um tamanho da

camada invisível variável entre 1 e 10. Cada iteração implica o acréscimo de um neurónio à camada invisível. Assim, como demonstrado na figura 28, obtiveram-se as métricas de validação do modelo *nnet*, sendo que o modelo com melhor desempenho demonstrou uma exatidão de 80,37% e um erro quadrático médio de 0,142 com um tamanho da camada invisível de 10 neurónios. Convém notar que estas métricas correspondem à fase de validação, então este modelo foi recuperado e testado, tendo apresentado uma exatidão de 79,69%.

Já o modelo com pior desempenho demonstrou uma exatidão de validação de apenas 69,87%, sendo que esta aumentou em fase de teste para 71,99%.

```
"Vetor contendo os valores de erro:"
 [1] 0.1899626 0.1917789 0.1800992 0.1881827 0.1817893 0.1693994 0.1669777 0.1431061 0.1470485 0.1416448
[1] "Vetor com valores de exatidão:"
 [1] 0.6986971 0.7035831 0.7271987 0.7166124 0.7125407 0.7434853 0.7434853 0.7931596 0.7866450 0.8037459
[1] "Media de exatidao da rede nnet com tamanho variável utilizando 10-fold cross-validation: "
[1] 0.7429153
[1] "Media de Mean Squared Error da rede nnet com tamanho variável utilizando 10-fold cross-validation: "
[1] 0.1699989
[1] "A exatidão do modelo que apresentou melhores resultados na validação é:"
[1] 0.7969461
[1] "A exatidão do modelo que apresentou piores resultados na validação é:"
[1] 0.719948
```

Figura 28 - Resultados obtidos para 10-fold cross-validation ao modelo nnet com tamanho variável

3.2.2. Aplicação de *10-fold cross-validation* ao modelo *nnet* com camada invisível de tamanho fixo

Os resultados obtidos anteriormente permitem aferir que o melhor desempenho deste método é aplicado ao modelo *nnet* com 10 neurónios na camada invisível. Assim, este número permanecerá fixo neste estudo.

Na figura 29 encontram-se os resultados obtidos, sendo que o valor médio de exatidão do modelo *nnet* com 10 neurónios é de 77,4% e o erro quadrático médio é de 0.154.

Quando ao teste dos modelos com melhor e pior desempenho, estes obtiveram uma exatidão de 79,2% e 73%, respetivamente.

```
"Vetor contendo os valores de erro:"
 [1] 0.1524365 0.1758350 0.1667424 0.1450278 0.1613634 0.1455281 0.1418505 0.1466079 0.1519716 0.1572349
[1] "Vetor com valores de exatidão:"
 [1] 0.7776873 0.7141694 0.7605863 0.7866450 0.7614007 0.7793160 0.8078176 0.7899023 0.7793160 0.7833876
[1] "Media de exatidao da rede nnet com tamanho fixo utilizando 10-fold cross-validation: "
[1] 0.7740228
[1] "Media de Mean Squared Error da rede nnet com tamanho fixo utilizando 10-fold cross-validation: "
[1] 0.1544598
[1] "A exatidão do modelo que apresentou melhores resultados na validação é:"
[1] 0.7920728
[1] "A exatidão do modelo que apresentou piores resultados na validação é:"
[1] 0.7300195
```

Figura 29 - Resultados obtidos para 10-fold cross-validation ao modelo nnet com tamanho fixo

3.2.3. Aplicação de *10-fold cross-validation* ao modelo *nnet* com *decay* variável

Como continuação do ponto anterior, o modelo *nnet* foi testado variando o hiperparâmetro *decay*, que controla a penalização aplicada aos pesos da rede e pode, por isso, ajudar a prevenir o *overfitting*. Neste teste, o parâmetro *decay* variou entre cada iteração num intervalo de 0,01 a 0,1, sendo aumentado em intervalos de 0,01.

Na fase de validação, o modelo *nnet* com melhor exatidão foi o modelo cujo parâmetro *decay* assumia o valor de 0,02, ainda com um erro quadrático médio pertencente ao grupo dos mais baixos. Em fase de teste, este modelo apresentou uma exatidão de 79,53%. Na figura 30 encontram-se os resultados obtidos.

```
"Vetor contendo os valores de erro:"
 [1] 0.1623575 0.1463111 0.1553974 0.1423913 0.1568725 0.1504134 0.1546489 0.1709230 0.1535863 0.1541005
[1] "Vetor com valores de exatidão:"
 [1] 0.7703583 0.7980456 0.7671010 0.7874593 0.7671010 0.7809446 0.7719870 0.7410423 0.7752443 0.7833876
[1] "Media de exatidao da rede nnet com tamanho fixo utilizando 10-fold cross-validation: "
[1] 0.7742671
[1] "Media de Mean Squared Error da rede nnet com tamanho fixo utilizando 10-fold cross-validation: "
[1] 0.1547002
[1] "A exatidão do modelo que apresentou melhores resultados na validação é:"
[1] 0.7953216
[1] "A exatidão do modelo que apresentou piores resultados na validação é:"
[1] 0.7641326
```

Figura 30 - Resultados obtidos para 10-fold cross-validation ao modelo nnet com decay variável

3.2.4. Aplicação de *10-fold cross-validation* ao modelo *neuralnet* com camada invisível de tamanho fixo

O método de *10-fold cross-validation* foi aplicado ao modelo *neuralnet*, com uma camada invisível com 1 neurónio. É de notar que o erro quadrático médio em cada iteração aumentou significativamente em relação ao modelo *nnet*, nas mesmas condições. O valor médio desta métrica é de 0,625 e o valor médio da exatidão do modelo é 71,11%.

Os modelos com melhor e pior desempenho foram testados, novamente com os dados de teste referidos na secção 2.3. do presente relatório, apresentando uma exatidão de 71,47% e 71,15%, respetivamente.

De uma forma geral, os valores de exatidão mantiveram-se num intervalo relativamente pequeno, dando a ideia de que este modelo é bastante mais sólido do que o modelo *nnet*. Os resultados obtidos podem ser observados na figura 31.

```
"Vetor contendo os valores de erro:"
 [1] 0.6218759 0.6215940 0.6318975 0.6323564 0.6194604 0.6190522 0.6283318 0.6257159 0.6242829 0.6260331
[1] "Vetor com valores de exatidão:"
 [1] 0.7027687 0.7271987 0.7149837 0.7141694 0.7019544 0.7182410 0.6954397 0.7214984 0.7133550 0.7019544
[1] "Media de exatidao da rede neuralnet com tamanho fixo utilizando 10-fold cross-validation: "
[1] 0.7111564
[1] "Media de Mean Squared Error da rede neuralnet com tamanho fixo utilizando 10-fold cross-validation: "
[1] 0.62506
[1] "A exatidão do modelo que apresentou melhores resultados na validação é:"
[1] 0.7147498
[1] "A exatidão do modelo que apresentou piores resultados na validação é:"
[1] 0.711501
```

Figura 31 - Resultados obtidos para 10-fold cross-validation ao modelo neuralnet com tamanho fixo

3.2.5. Aplicação de *repeated 10-fold cross-validation* a um *Support Vector Machine*

Através da biblioteca *caret*, exemplificada na figura 32, foi possível treinar um modelo *Support Vector Machine* com o conjunto de dados balanceado, utilizando um método de *10-fold cross-validation* repetido três vezes.

Uma vez que foi passado o argumento '*tuneLength = 20'*, então esta fase de treino irá avaliar o desempenho do modelo para 20 valores diferentes de hiperparâmetros.

```
library(caret)
formula_nn <- Class~age+Tot_Bilirubin+Dir_Bilirubin+Alkphos+Sgpt+Sgot+Tot_Protiens+Albumin+ag_Ratio+Female+Male

trctrl <- trainControl(method = "repeatedcv", number = 10,repeats=3)

svmModel <- train(formula_nn, data = liver_treino, method = "svmLinear",
                  trControl=trctrl,tuneLength = 20,preProc = c("center","scale"))

model_results <- predict(svmModel,newdata=liver_teste[1:11]) #we use predict instead of compute

print(confusionMatrix(model_results, liver_teste$Class ))
```

Figura 32 - Algoritmo de treino e teste de um Support Vector Machine

Este método de *Support Vector Machine* com Kernel Linear, na fase de teste, provou ter uma exatidão de aproximadamente 71%, bem como uma especificidade de 0,51 e sensibilidade 0,91. No entanto, este modelo prevê bastantes casos como sendo de não-pacientes e, por isso, o número de falsos-positivos aumenta de maneira significativa (figura 33).

	Reference	
Prediction	not_patient	patient
not_patient	1405	757
patient	134	782

Figura 33 - Matriz de confusão obtida através do método de Support Vector Machine

3.2.6. Aplicação de *10-fold cross-validation* a um modelo *Gradient Boosting Machine*

Ao implementar este método, foi efetuado um tuning de diversos parâmetros, como representado na figura 34.

```
  interaction.depth  n.trees  Accuracy   Kappa
  1                   50      0.7235519  0.4471043
  1                  100      0.7280274  0.4560542
  1                  150      0.7333980  0.4667960
  2                   50      0.7349459  0.4698864
  2                  100      0.7747398  0.5494779
  2                  150      0.8094065  0.6188102
  3                   50      0.7598488  0.5196937
  3                  100      0.8210445  0.6420878
  3                  150      0.8724784  0.7449552

Tuning parameter 'shrinkage' was held constant at a value of 0.1
Tuning parameter 'n.minobsinnode' was held constant at
 a value of 10
Accuracy was used to select the optimal model using the largest value.
```

Figura 34 - Tuning de vários parâmetros do algoritmo Gradient Boosting Machine

Na figura 35, é possível observar que a exatidão obtida durante a fase de validação aumenta consoante aumenta o número de iterações. Além disso, a exatidão do modelo também aumenta quanto maior for o parâmetro *Max Tree Depth.*

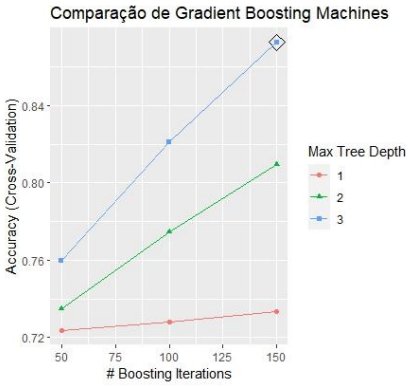


Figura 35 - Comparação de Gradient Boosting Machines durante a fase de validação

A aplicação do método de *Gradient Boosting Machine*, utilizando uma *10-fold cross-validation* demonstrou uma grande capacidade de exatidão, com aproximadamente 86,7%. As métricas de sensibilidade e especificidade (0,947 e 0,788, respetivamente) também apresentam melhores valores do que os apresentados por outros modelos.

Ao nível da matriz de confusão, é possível verificar que foi possível reduzir os casos de falsos positivos e de falsos negativos, como demonstrado na figura 36.

	Reference	
Prediction	not_patient	patient
not_patient	1457	327
patient	82	1212

Figura 36 - Matriz de confusão para os resultados do modelo Gradient Boosting Machine

3.2.7. Aplicação do algoritmo *Extreme Learning Machine* com *cross-validation*

Como abordado na secção 3.1.5. do presente relatório, o algoritmo *Extreme Learning Machine* apresentou um aumento de exatidão consoante o aumento do número de neurónios. Em vez do método de *hold-out*, será mais interessante realizar uma *cross-validation* com alteração dos hiperparâmetros para conseguir chegar a um resultado que permita aferir um intervalo de variação do número de neurónios na camada invisível. Foi utilizado o algoritmo apresentado na secção 2.3., com o valor do número de neurónios a ser incrementado em cada iteração por 200, variando o intervalo de estudo de 200 a 2000 neurónios.

Na figura 37 encontram-se os resultados obtidos, sendo relevante que de facto quanto maior o número de neurónios utilizados, maior será a percentagem de exatidão. Neste caso, o modelo com melhor desempenho de exatidão na fase de validação é também o modelo que apresenta um maior erro quadrático médio, sendo que na fase de teste obteve cerca de 78,95% de exatidão.

```
  |                                                                                                                      |   0%[1]
0.6946254
  |============                                                                                                          |  10%[1]
0.7149837
  |========================                                                                                              |  20%[1]
0.747557
  |====================================                                                                                  |  30%[1]
0.7565147
  |================================================                                                                      |  40%[1]
0.7508143
  |============================================================                                                          |  50%[1]
0.764658
  |========================================================================                                              |  60%[1]
0.7687296
  |====================================================================================                                  |  70%[1]
0.776873
  |============================================================================================                          |  80%[1]
0.7850163
  |====================================================================================================                  |  90%[1]
0.7947883
  |======================================================================================================================| 100%[1]
"Vetor contendo os valores de erro:"
 [1] 0.6341860 0.6626713 0.6714626 0.7028536 0.7581857 1.0160181 0.8158467 0.7808453 0.9732848 1.9353024
[1] "Vetor com valores de exatidão:"
 [1] 0.6946254 0.7149837 0.7475570 0.7565147 0.7508143 0.7646580 0.7687296 0.7768730 0.7850163 0.7947883
[1] "Media de exatidao do algoritmo Extreme Learning Machine com tamanho variável utilizando 10-fold cross-validation: "
[1] 0.755456
[1] "Media de Mean Squared Error do algoritmo Extreme Learning Machine com tamanho variável utilizando 10-fold cross-validation
"
[1] 0.8950656
[1] "A exatidão do modelo que apresentou melhores resultados na validação é:"
[1] 0.7894737
[1] "A exatidão do modelo que apresentou piores resultados na validação é:"
[1] 0.7157245
```

Figura 37 - Resultados obtidos para o método de cross-validation para o algortimo Extreme Learning Machine

3.3 Influência da atribuição de valores aos dados em falta

De maneira a compreender se a limpeza do conjunto de dados foi bem efetuada na secção 2.1, os dados apagados (representados na figura 3) foram restaurados, atribuindo às variáveis numéricas contínuas a mediana do conjunto de valores da mesma variável e atribuindo às variáveis categóricas nominais a moda do conjunto de valores da mesma variável. Para isso, os dados foram separados em termos de *Class* (*patient* ou *not_patient*) para calcular as medianas e as modas e manter uma diferenciação entre os valores referentes a estas duas classes.

Assim, 2250 observações foram restauradas e a avaliação dos diferentes modelos e métodos foi realizada seguindo um *dataset* balanceado, com o mesmo número de observações referentes a pacientes e não-pacientes. No total, 16738 observações eram passíveis de serem utilizados, sendo que foi efetuada uma divisão de 80% (13390 casos) para treinar os diferentes modelos e 20% (3348 casos) para testar os mesmos.

Ao aplicar o método *hold-out* ao modelo *neuralnet*, são obtidas as métricas de exatidão, sensibilidade e especificidade de 71%, 66% e 80,5%, respetivamente. A matriz de confusão obtida pode ser observada na figura _. Já no caso do modelo *nnet*, a exatidão obtida foi de 77,6%, com uma sensibilidade de 73,4% e especificidade de 84%, com a matriz de confusão obtida representada na figura 38.

Real	Predicted not_patient	Predicted patient
not_patient	1475	219
patient	750	904

(a)

Real	Predicted not_patient	Predicted patient
not_patient	1483	211
patient	537	1117

(b)

*Figura 38 - **(a)** Matriz de confusão obtida para o modelo neuralnet; **(b)** Matriz de confusão obtida para o modelo nnet*

Já o método de *10-fold cross-validation* foi aplicado utilizando os modelos *neuralnet, nnet, Extreme Learning Machine, Support Vector Machine* e *Gradient Boosting Machine*.

Foi utilizada, numa primeira fase, uma rede neuronal *neuralnet* com apenas um neurónio na camada invisível. Escolheu-se o melhor modelo da fase de validação e, na fase de teste, obteve-se uma exatidão de 71% e um erro quadrático médio de 0,62 (figura 39).

```
"Vetor contendo os valores de erro:"
 [1] 0.6234144 0.6272034 0.6207460 0.6192745 0.6154593 0.6204364 0.6210023 0.6204838 0.6157659 0.6179048
[1] "Vetor com valores de exatidão:"
 [1] 0.7154593 0.7206871 0.6878267 0.6908140 0.7087379 0.7102315 0.7005228 0.6855863 0.7229276 0.7050037
[1] "Media de exatidao da rede neuralnet com tamanho fixo utilizando 10-fold cross-validation: "
[1] 0.7047797
[1] "Media de Mean Squared Error da rede neuralnet com tamanho fixo utilizando 10-fold cross-validation: "
[1] 0.6201691
[1] "A exatidão do modelo que apresentou melhores resultados na validação é:"
[1] 0.7102748
[1] "A exatidão do modelo que apresentou piores resultados na validação é:"
[1] 0.7150538
```

Figura 39 - 10-fold cross-validation utilizando o modelo neuralnet

Numa segunda fase, foi testada uma rede neuronal *nnet* com 10 neurónios na camada invisível. Os resultados obtidos, relativos à exatidão do modelo em fase de teste, situaram-se entre os 75,7% e os 79,96%, com um erro quadrático médio de 0,15. Os resultados obtidos podem ser observados na figura 40.

```
"Vetor contendo os valores de erro:"
 [1] 0.1598763 0.1511822 0.1538624 0.1489320 0.1556604 0.1552594 0.1505898 0.1436769 0.1600603 0.1678632
[1] "Vetor com valores de exatidão:"
 [1] 0.7595220 0.7669903 0.7647498 0.7766990 0.7617625 0.7669903 0.7976102 0.7789395 0.7729649 0.7401046
[1] "Media de exatidao da rede nnet com tamanho fixo utilizando 10-fold cross-validation: "
[1] 0.7686333
[1] "Media de Mean Squared Error da rede nnet com tamanho fixo utilizando 10-fold cross-validation: "
[1] 0.1546963
[1] "A exatidão do modelo que apresentou melhores resultados na validação é:"
[1] 0.7995818
[1] "A exatidão do modelo que apresentou piores resultados na validação é:"
[1] 0.7574671
```

Figura 40 - 10-fold cross-validation utilizando o modelo nnet

Após a utilização dos modelos principais, foram testadas outras opções na tentativa de descobrir um outro modelo que possa ser implementado com maior exatidão. Assim, o algoritmo *Extreme Learning Machine* apresenta uma exatidão entre 72,3% e 78,7%, com um erro quadrático médio de 0,83 (figura 41).

```
"Vetor contendo os valores de erro:"
 [1] 0.6309268 0.6470986 0.7942292 0.7169932 0.8385919 0.8316915 0.8547865 1.1099279 0.9130023 0.9405744
[1] "Vetor com valores de exatidão:"
 [1] 0.7147125 0.6997760 0.7303958 0.7430919 0.7348768 0.7393577 0.7610157 0.7707244 0.7841673 0.7737117
[1] "Media de exatidao do algoritmo Extreme Learning Machine com tamanho variável utilizando 10-fold cross-validation: "
[1] 0.745183
[1] "Media de Mean Squared Error do algoritmo Extreme Learning Machine com tamanho variável utilizando 10-fold cross-validation:
"
[1] 0.8277822
[1] "A exatidão do modelo que apresentou melhores resultados na validação é:"
[1] 0.787037
[1] "A exatidão do modelo que apresentou piores resultados na validação é:"
[1] 0.7231183
```

Figura 41 - 10-fold cross-validation utilizando o algoritmo Extreme Learning Machine

Utilizando um *Support Vector Machine*, com o mesmo método de *10-fold cross-validation*, obteve-se uma exatidão de 71%, sensibilidade de 91,6% e especificidade de 50% na fase de teste. A matriz de confusão obtida está representada na figura 42.

	Reference	
Prediction	not_patient	patient
not_patient	1552	826
patient	142	828

Figura 42 - 10-fold cross-validation utilizando um Support Vector Machine

Utilizando uma *Gradient Boosting Machine*, com o mesmo método utilizado anteriormente, obteve-se uma exatidão de 86,8%, sensibilidade de 94% e especificidade de 79,4% na fase de teste. A matriz de confusão obtida está representada na figura 43.

	Reference	
Prediction	not_patient	patient
not_patient	1593	340
patient	101	1314

Figura 43 - 10-fold cross-validation utilizando Gradient Boosting Machine

3.4 *Feature elimination* como tentativa melhorar resultados

Ao aplicar a técnica de *feature elimination*, visou-se estudar a influência do género do paciente nos resultados. Este processo tem como objetivo escolher variáveis que podem vir a ser irrelevantes ou prejudiciais aos modelos. Para este estudo, eliminaram-se as variáveis *Male* e *Female*, na tentativa de entender se estas causavam uma menor exatidão ao estudar a previsão de doenças hepáticas, bem como foi aplicado juntamente com uma técnica de *cross-validation*.

Quanto ao modelo *neuralnet*, obteve-se uma exatidão 73,35% na fase de teste, apresentando um erro quadrático médio de 0,62. Na figura 44 estão representados os resultados obtidos.

```
"Vetor contendo os valores de erro:"
 [1] 0.6098717 0.6247251 0.6225863 0.6204404 0.6145607 0.6188030 0.6167926 0.6311299 0.6201884 0.6216646
[1] "Vetor com valores de exatidão:"
 [1] 0.7223127 0.7125407 0.7157980 0.6905537 0.7157980 0.7231270 0.6921824 0.7451140 0.6921824 0.7019544
[1] "Media de exatidao da rede neuralnet com tamanho fixo e threshold fixo utilizando 10-fold cross-validation: "
[1] 0.7111564
[1] "Media de Mean Squared Error da rede neuralnet com tamanho fixo e threshold fixo utilizando 10-fold cross-validation: "
[1] 0.6200763
[1] "A exatidão do modelo que apresentou melhores resultados na validação é:"
[1] 0.7335932
[1] "A exatidão do modelo que apresentou piores resultados na validação é:"
[1] 0.7111761
```

Figura 44 - Resultados obtidos para o modelo neuralnet

Também o modelo *nnet* foi testado, com um tamanho de 10 neurónios na camada invisível, obtendo-se uma exatidão na fase de teste de 81,16% e um erro quadrático médio de 0,14. Os resultados obtidos estão representados na figura 45.

```
"Vetor contendo os valores de erro:"
 [1] 0.1273306 0.1402063 0.1316166 0.1503192 0.1409678 0.1388163 0.1658604 0.1406147 0.1328509 0.1540964
[1] "Vetor com valores de exatidão:"
 [1] 0.8135179 0.8192182 0.8118893 0.7882736 0.8143322 0.7890879 0.7491857 0.7947883 0.8045603 0.7491857
[1] "Media de exatidao da rede nnet com tamanho fixo utilizando 10-fold cross-validation: "
[1] 0.7934039
[1] "Media de Mean Squared Error da rede nnet com tamanho fixo utilizando 10-fold cross-validation: "
[1] 0.1422679
[1] "A exatidão do modelo que apresentou melhores resultados na validação é:"
[1] 0.811566
[1] "A exatidão do modelo que apresentou piores resultados na validação é:"
[1] 0.7550357
```

Figura 45 - Resultados obtidos para o modelo nnet

Quanto ao algoritmo *Extreme Learning Machine*, este alcançou uma exatidão de 82,84% com um erro quadrático médio de 0,85 (figura 46).

```
"Vetor contendo os valores de erro:"
 [1] 0.8692547 0.8996229 0.8079952 0.8272147 0.9460483 0.9591042 0.8034034 0.7673059 0.7913265 0.8378813
[1] "Vetor com valores de exatidão:"
 [1] 0.7947883 0.8314332 0.8045603 0.8110749 0.8127036 0.8241042 0.8200326 0.8094463 0.8232899 0.8143322
[1] "Media de exatidao do algoritmo Extreme Learning Machine com tamanho variável utilizando 10-fold cross-validation: "
[1] 0.8145765
[1] "Media de Mean Squared Error do algoritmo Extreme Learning Machine com tamanho variável utilizando 10-fold cross-validation:
"
[1] 0.8509157
[1] "A exatidão do modelo que apresentou melhores resultados na validação é:"
[1] 0.82846
[1] "A exatidão do modelo que apresentou piores resultados na validação é:"
[1] 0.8281352
```

Figura 46 - Resultados obtidos para o algoritmo Extreme Learning Machine

O modelo *Support Vector Machine* apresentou uma exatidão de 69,66%, com uma sensibilidade de 0,89 e uma especificidade de 0,49, apresentando a matriz de confusão representada na figura 47.

	Reference	
Prediction	not_patient	patient
not_patient	1378	773
patient	161	766

Figura 47 - Matriz de confusão obtida para o modelo Support Vector Machine

Por último, também o algoritmo Gradient Boosting Machine foi testado, apresentando uma exatidão de 87,4%, sensibilidade de 0,94 e especificidade de 0,81. A matriz obtida para este teste está representada na figura 48.

	Reference	
Prediction	not_patient	patient
not_patient	1449	298
patient	90	1241

Figura 48 - Matriz de confusão obtida para o modelo Gradient Boosting Machine

4 Discussão

Ao longo deste capítulo, os resultados obtidos serão analisados e discutidos, comparando com a literatura. Por fim, irá ser realizada uma descrição de como o modelo escolhido poderia ser implementado num sistema computacional.

4.1 Análise dos resultados obtidos

Quanto ao método de *hold-out* implementado numa primeira fase e devido aos resultados obtidos, foi possível chegar a uma conclusão de que a divisão dos dados para treino e teste deve-se situar nos 80% e 20%, respetivamente. Também foi possível entender a importância do balanceamento dos dados para alcançar a melhor exatidão dos modelos utilizados, de uma forma imparcial. Ao utilizar este método, deparamo-nos com uma exatidão maior do modelo *nnet* com 10 neurónios em relação ao modelo *neuralnet* com 1 neurónio. Infelizmente não foi possível testar o modelo com mais do que um neurónio na camada invisível. Ainda utilizando este método, foi possível observar que mais neurónios na camada invisível do algoritmo *Extreme Learning Machine* implicaria uma maior exatidão.

Assim, implementando o método de *10-fold cross-validation*, obteve-se uma melhor exatidão para este algoritmo, utilizando 2000 neurónios na camada invisível. Quanto ao modelo *nnet*, observou-se que os melhores parâmetros de tamanho e *decay* tomavam os valores de 10 e 0,02, respetivamente. Também o algoritmo *Support Vector Machine* foi testado e, uma vez que não foi atingida

uma exatidão muito boa (por volta dos 71%), é possível concluir que os dados observados e analisados não são lineares. Assim, como não é possível dividi-los quanto à sua linearidade, implementou-se o método *Gradient Boosting Machine*, alcançando um melhor resultado (aproximadamente de 87%) utilizando os seguintes parâmetros:

i. *interaction.depth* assume o valor de 3
ii. *n.trees* assume o valor de 150
iii. *shrinkage* assume o valor de 0,1
iv. *n.minobsinnode* assume o valor de 10

De seguida, estudou-se a influência dos dados previamente apagados. Os dados foram recuperados e as observações com apenas um dado em falta receberam um valor aproximado do esperado, atribuindo um valor de mediana ou moda consoante se era paciente ou não. Assim, as técnicas de *hold-out* e *cross-validation* foram repetidas para os diferentes modelos, não havendo diferenças significativas com estes novos dados aproximados.

Por último, retiraram-se as variáveis no que ao género dos pacientes dizia respeito, uma vez que se suspeitava que estas poderiam estar a influenciar negativamente a aprendizagem e a tomada de decisão por parte dos modelos. Ao realizar este estudo, foi possível observar que os modelos *neuralnet, nnet, Extreme Learning Machine* e *Gradient Boosting Machine* melhoraram os seus resultados de exatidão, ao passo que o modelo *Support Vector Machine* piorou, demonstrando que os dados continuaram não lineares.

4.2 Comparação com a literatura

Em comparação com os métodos utilizados em [10] para o mesmo *dataset*, os resultados obtidos para o modelo *Support Vector Machine* ficaram na mesma gama de valores, por volta dos 70%, o que infere que existe uma falta de linearidade por parte do conjunto de dados e que a limpeza dos mesmo está bem efetuada. Tal como em [11], os resultados obtidos para este modelo estão no intervalo mencionado.

Singh et al [12], com um conjunto de dados diferente do usado neste trabalho, utilizou um modelo *Support Vector Machine*, alcançando uma exatidão inferior quando comparada com este projeto. No mesmo documento, referem que utilizaram um algoritmo *lightGBM*, semelhante ao modelo *Gradient Boosting Machine*, em que também atingiram uma exatidão inferior à atingida no presente relatório.

Agarwal et al [13] propuseram uma metodologia com um algoritmo Extreme Learning Machine, não alcançando um resultado tão bom quanto ao demonstrado neste relatório.

4.3 Descrição da implementação do modelo criado num sistema computacional

Idealmente, seria bastante interessante aplicar um modelo *nnet* com 10 neurónios na camada invisível e um *decay* de 0,02 ou um algoritmo de *Gradient Boosting Machine* com os parâmetros descritos em 4.1. a uma página *Web*, uma vez que foram modelos que se destacaram pela sua rapidez e pela sua capacidade de fazerem previsões bastante sólidas.

Esta página *Web* poderia ser realizada com recurso à biblioteca *Shiny* em R Studio, uma vez que permite a construção destas interfaces [14]. Assim, seria só necessário projetar esta interface, uma vez que os modelos em R já estariam prontos a utilizar.

A página Web poderia ter um fim médico, uma vez que os resultados das análises poderiam ser colocados nesta interface. Esta iria normalizar os dados introduzidos e descartar o género do paciente em causa. Enquanto o paciente espera pela consulta, o modelo daria o seu parecer e o médico poderia concordar ou discordar. Caso o modelo não acertasse, o médico daria a sua opinião e o peso de cada variável poderia ser repensado.

5 Conclusão

Em síntese, a elaboração deste trabalho académico evidenciou que as redes neuronais constituem uma forte solução para a resolução de diversos problemas na área da Engenharia Biomédica, destacando-se a área de diagnóstico de doenças hepáticas.

O trabalho desenvolvido permitiu analisar diferentes tipos de arquiteturas e métodos diferentes, e quando comparado com a literatura existente, demonstrou melhores resultados. Assim, é possível implementar um destes modelos num sistema computacional utilizando recursos presentes na linguagem R. Para isso, foi necessário compreender o que as mudanças no conjunto de dados a ser analisado poderiam despoletar nos modelos de aprendizagem, bem como compreender a necessidade de balancear os mesmos.

Também se entendeu a diferença de metodologias (entre utilizar *hold-out* ou *cross-validation*) e o que cada uma nos pode oferecer, não esquecendo da importância de realizar *feature selection* para aperfeiçoar os modelos.

Em suma, as redes neuronais representam um campo crescente, dada a precisão que têm vindo a apresentar nas suas aplicações em diagnósticos de doenças, a nível geral da Medicina. Este avanço tecnológico permite melhorar significativamente a qualidade de diagnósticos, beneficiando a saúde e bem-estar dos pacientes.

Referências

[1] T. Zanin, “Fígado: onde fica, funções e principais doenças,” junho 2022. [Online]. Available: https://www.tuasaude.com/funcao-do-figado/. [Acedido em 4 janeiro 2024].

[2] S. Fritsch, F. Guenther, M. N. Wright, M. Suling e S. M. Mueller, “Package 'neuralnet',” 7 fevereiro 2019. [Online]. Available: https://cran.r-project.org/web/packages/neuralnet/neuralnet.pdf. [Acedido em 21 dezembro 2023].

[3] B. Ripley e W. Venables, “Package 'nnet',” 2 maio 2023. [Online]. Available: https://cran.r-project.org/web/packages/nnet/nnet.pdf. [Acedido em 21 dezembro 2023].

[4] GeeksforGeeks, “Support Vector Machine Classifier Implementation in R with Caret package,” [Online]. Available: https://www.geeksforgeeks.org/support-vector-machine-classifier-implementation-in-r-with-caret-package/. [Acedido em 28 dezembro 2023].

[5] H. Singh, “Understanding Gradient Boosting Machines,” 3 novembro 2018. [Online]. Available: https://towardsdatascience.com/understanding-gradient-boosting-machines-9be756fe76ab. [Acedido em 6 janeiro 2024].

[6] G.-B. Huang, Q.-Y. Zhu e C.-K. Siew, “Extreme learning machine: Theory and applications,” em *Neurocomputing*, Singapura, 2006, pp. 489-501.

[7] A. Shrivastava, “Liver Disease Patient Dataset 30K train data,” [Online]. Available: https://www.kaggle.com/datasets/abhi8923shriv/liver-disease-patient-dataset/data. [Acedido em 22 dezembro 2023].

[8] Domino Data Lab, “Model Evaluation,” Domino, [Online]. Available: https://domino.ai/data-science-dictionary/model-evaluation. [Acedido em 25 dezembro 2023].

[9] K. Devi, “Understanding Hold-Out Methods for Training Machine Learning Models,” Comet ML, 14 agosto 2023. [Online]. Available: https://www.comet.com/site/blog/understanding-hold-out-methods-for-training-machine-learning-models/. [Acedido em 25 dezembro 2023].

[10] P. Kumar e R. S. Thakur, “Early Detection of the Liver Disorder from Imbalance Liver Function Test Datasets,” *International Journal of Innovative Technology and Exploring Engineering (IJITEE),* vol. 8, 2019.

[11] M. B. Priya, P. L. Juliet e P. R. Tamilselvi, “Performance Analysis of Liver Disease Prediction Using Machine Learning Algorithms,”

International Research Journal of Engineering and Technology, vol. 5, 2018.

[12] V. Singh, M. K. Gourisaria e H. Das, “Performance Analysis of Machine Learning Algorithms for Prediction of Liver Disease,” em *2021 IEEE 4th International Conference on Computing, Power and Communication Technologies*, Malásia, 2021.

[13] C. Agarwal, G. Singh e A. Mishra, “ELM-Based Liver Disease Prediction Model”.

[14] R Studio, “Shiny,” [Online]. Available: https://www.rstudio.com/products/shiny/. [Acedido em 7 janeiro 2024].