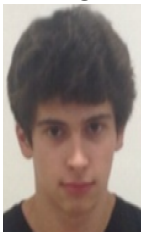
Exercício Prático Nº 3 - Grupo 3 - Sistemas de Representação de Conhecimento e Raciocínio

André Rodrigues Freitas



A74619

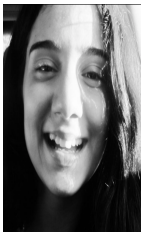
João Tiago Pereira Dias

A72095

Joel Tomás Morais

A70841

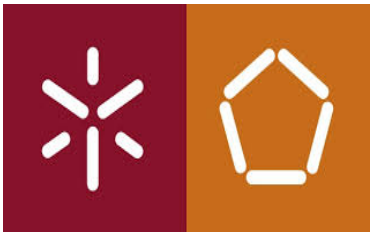
Sofia Manuela Gomes de Carvalho



A76658

21 de Maio de 2017

Universidade do Minho



Resumo

Neste último exercício prático, abandonamos a linguagem de programação em lógica estendida PROLOG e o conceito de Hard Computing para passar a trabalhar com a linguagem R, linguagem esta que permite analisar dados e com eles treinar uma máquina, através da inteligência artificial, englobada já na segunda parte desta unidade curricular, que é referente ao Soft Computing. Ao longo deste relatório são abordados os vários aspetos a ter em conta para o exercício, bem como para a sua resolução.

Conteúdo

1 Introdução

No terceiro e último exercício prático, recorremos à linguagem de programação R para utilização de sistemas não simbólicos na representação de conhecimento e no desenvolvimento de mecanismos de raciocínio, nomeadamente Redes Neuronais Artificiais (RNAs) para a resolução de problemas. Esta permitirá realizar um estudo que envolva a identificação da exaustão reconhecida e a tarefa em execução. Para tal, usamos os dados fornecidos no enunciado que são apresentados sob a forma de uma tabela constituída por um conjunto de dados recolhidos da interação humano-computador através dos dispositivos físicos rato e teclado.

2 Preliminares

Este trabalho prático pretende que seja detetado o nível de exaustão de um utilizador e a tarefa a ser executada por ele, tendo por base diversos atributos, sendo eles:

- "Performance.KDTMean" – tempo médio entre o momento em que a tecla é pressionada para baixo e o momento em que é largada;
- "Performance.MAMean" – aceleração do manuseamento rato em determinado momento. O valor da aceleração é calculado através da velocidade do rato (pixel/milissegundos) sobre o tempo de movimento (milissegundos);
- "Performance.MVMean" – velocidade do manuseamento do rato em determinado momento. A distância percorrida pelo rato (em píxeis) entre uma coordenada C1 (x1; y1) e uma C2 (x2; y2) correspondentes a time1 e time2, sobre o tempo (em milissegundos);
- "Performance.TBCMean" – tempo entre dois clicks consecutivos, entre eventos consecutivos MOUSE_UP e MOUSE_DOWN;
- "Performance.DDCMean" – período de tempo entre dois eventos MOUSE_UP consecutivos;
- "Performance.DMSMean" – distância média em excesso entre o caminho de dois clicks consecutivos;
- "Performance.ADMSLMean" – distância média das diferentes posições do ponteiro entre dois pontos durante um movimento, e o caminho em linha reta entre esses mesmos dois pontos;
- "Performance.AEDMean" – esta métrica é semelhante à anterior, no sentido em calculará a soma da distância entre dois eventos MOUSE_UP e MOUSE_DOWN consecutivos;
- "ExhaustionLevel" – nível subjetivo de exaustão mental;
- "Performance.Task" – identificação da tarefa em execução no momento da recolha dos dados.

.

3 Descrição do Trabalho e Análise de Resultados

No desenvolvimento deste terceiro exercício prático, foi utilizada a ferramenta RStudio, que permite a escrita de comandos na linguagem R num terminal. É através desses comandos que é feito o desenvolvimento de todas as redes neuronais, desde o carregamento dos dados necessários para o treino da rede e respetivos testes até à avaliação dos resultados obtidos pela rede.

Como o tema principal deste exercício é a inteligência artificial e o Soft Computing, foi-nos fornecido pelo docente um ficheiro anexo que continha todos os dados de excertos de dados biométricos para detetar a exaustão.

A primeira alteração que fizemos a estes dados foi normalizar a coluna Task, que era a única coluna que tinha valores não normalizados. Assim, procedemos à atribuição do valor 1 para work, o valor 2 para office e o valor 3 para programming. De realçar que o valor 0 não foi usado para fazer nenhuma normalização pois este é um valor numérico com características matemáticas "especiais"e que só deverá ser usado em casos muito específicos.

Para obter os resultados relativos ao *Fatigue level* e à *Task*, foi decidido por todo o grupo em construir duas redes neuronais independentes uma da outra, em que cada uma teria um único respetivo *output*. Assim, este exercício prático foi desenvolvido com duas redes neuronais artificiais: uma com o *Fatigue level* como *output* e outra com o parâmetro *Task* como nodo de saída.

3.1 Estudo dos atributos mais significativos

Para selecionar apenas os atributos mais significativos para cada rede, segundo o seu objetivo em concreto (níveis de fatigue e task), recorremos a duas metodologias distintas para obter essa indicação:

- a ferramenta *Weka*;
- a biblioteca *leaps*;

3.1.1 Machine learning software Weka

Nesta primeira abordagem para resolvermos este problema de decidirmos quais os atributos mais significativos, decidimos recorrer a um *software* de *Machine Learning*, chamado *Weka*, sugerido pelo docente nas aulas práticas, que nos irá ajudar a escolher os melhores atributos para cada rede neuronal artificial.

Após fazermos o *import* dos nossos dados para o *Weka*, é necessário irmos à secção "Select attributes", onde, a partir desta, vamos poder conseguir selecionar o atributo que pretendemos usar como nodo de saída. No caso concreto do nosso trabalho, iremos ter dois *outputs* possíveis, como já foi referido acima, sendo eles o *Fatigue Level* e o *Task*.

Um dos motivos pelos quais decidimos dividir estes dois *outputs* por duas redes foi devido ao facto de não queremos que a *Fatigue Level* influencie a *Task* ou vice-versa, e foi necessário retirar um deles, cada vez que queremos que o outro seja considerado o *output*. Outra situação em que nos deparamos que motivou esta separação de *outputs* foi o facto de ter um desses *outputs* nos *inputs* quando na verdade apenas era output.

Após selecionarmos cada um (individualmente), obtivemos a lista dos atributos mais significativos para um, que serão demonstrados de seguida, nestas duas figuras:

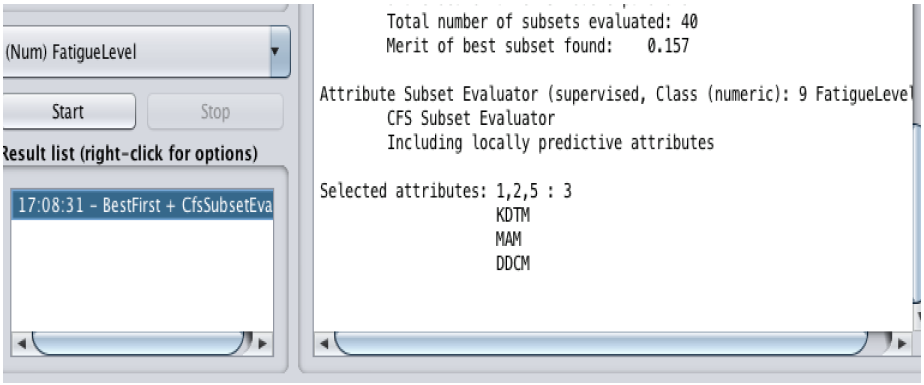


Figura 1: Fatigue como output

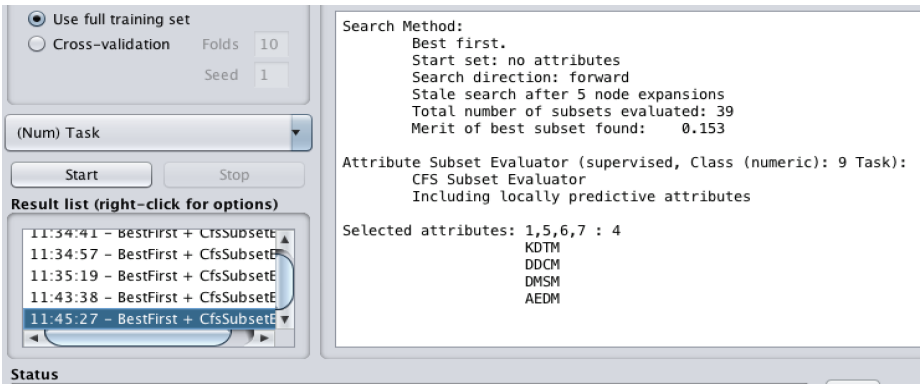


Figura 2: Task como output

Pela análise das figuras acima, é possível concluir que os três atributos mais significativos quando o output corresponde a *Fatigue level* são o KDTMean, o MAMean e o DDCMean. Quando o output é a *Task*, os três principais atributos são o KDTMean, o DDCMean e o DMSMean.

Estes resultados obtidos, contudo, não foram usados na sua totalidade pois, tendo sido obitdos pelo *Weka*, ou seja, uma ferramenta externa, o grupo decidiu "não utilizá-los"mas considerar os resultados obtidos por esta e utilizá-los como suporte e apoio à criação e desenvolvimento da rede, pois o mesmo objetivo pode ser alcançado pela utilização básica de certas funcionalidades do *RStudio*, recorrendo à biblioteca *leaps*.

3.1.2 Utilização da biblioteca leaps

Uma alternativa para a escolha dos atributos mais relevantes para cada *output* é a biblioteca *leaps* que pode ser instalada no RStudio pelo método habitual.

Através dela são fornecidas várias funcionalidades que permitem classificar os atributos de entrada.

Face à opção de utilização de duas redes neuronais distintas, começamos por verificar primeiro quais eram os inputs mais relevantes o *output FatigueLevel*. Para tal, e após a biblioteca *leaps* estar operacional, começamos por carregar o ficheiro com os nossos dados para uma variável chamada dados. Após isto, utilizamos o comando *regsubsets* indicando o *output* (*FatigueLevel*) e os inputs ($KDTM + MAM + MVM + TBCM + DDCM + DMSM + AEDM + ADMSLM$) através de uma fórmula no primeiro argumento do comando, e ainda indicando os dados correspondentes a esses atributos. Assim, guarda-se o resultado desta instrução na variável regg1, que, logo a seguir, ao ser invocado o comando *summary* sobre esta, vai mostrar no terminal os atributos com vários níveis de asteriscos. Quanto maior o número de asteriscos, mais relevante o atributo será. Assim, como se pode ver na imagem abaixo, os três atributos mais relevantes para a *FatigueLevel* são o *DDCMean*, o *MAMean* e o *MVMean*.

```
> library(leaps)
> dados<-read.csv("C:\\Users\\André\\Documents\\SRCR\\Trabalho 3\\exaustao numerico.csv")
> regg1<-regsubsets(FatigueLevel ~ KDTM + MAM + MVM + TBCM + DDCM + DMSM + AEDM + ADMSLM, dados)
> summary(regg1)
Subset selection object
Call: regsubsets.formula(FatigueLevel ~ KDTM + MAM + MVM + TBCM + DDCM +
    DMSM + AEDM + ADMSLM, dados)
8 Variables  (and intercept)
       Forced in Forced out
KDTM       FALSE      FALSE
MAM        FALSE      FALSE
MVM        FALSE      FALSE
TBCM       FALSE      FALSE
DDCM       FALSE      FALSE
DMSM       FALSE      FALSE
AEDM       FALSE      FALSE
ADMSLM     FALSE      FALSE
1 subsets of each size up to 8
Selection Algorithm: exhaustive
         KDTM MAM MVM TBCM DDCM DMSM AEDM ADMSLM
1  ( 1 ) " "  " " " " " "  "*"  " "  " "  " "
2  ( 1 ) " "  "*" " " " "  "*"  " "  " "  " "
3  ( 1 ) " "  "*" "*" " "  "*"  " "  " "  " "
4  ( 1 ) "*"  "*" "*" " "  "*"  " "  " "  " "
5  ( 1 ) "*"  "*" "*" " "  "*"  "*"  " "  " "
6  ( 1 ) "*"  "*" "*" " "  "*"  "*"  " "  "*"
7  ( 1 ) "*"  "*" "*" "*"  "*"  "*"  " "  "*"
8  ( 1 ) "*"  "*" "*" "*"  "*"  "*"  "*"  "*"
```

Figura 3: Utilização do leaps para escolha dos atributos mais relevantes para o output FatigueLevel

De seguida, realizamos o processo anterior de forma análoga para o *output Task* e, como se pode ver na figura abaixo, os quatro atributos, que serão posteriormente utilizados, mais relevantes para uma rede com este nodo de saída são o *KDTMean*, o *DMSMean*, o *DDCMean* e o *AEDMean*.

A escolha final dos atributos a utilizar foi baseada nos resultados obtidos pelo *leaps*, pois são obtidos por uma ferramenta incluída no RStudio, que também é utilizado para as restantes fases do estudo das redes neuronais.

```
> regg2<-regsubsets(Task ~ KDTM + MAM + MVM + TBCM + DDCM + DMSM + AEDM + ADMSLM, dados)
> summary(regg2)
Subset selection object
Call: regsubsets.formula(Task ~ KDTM + MAM + MVM + TBCM + DDCM + DMSM +
    AEDM + ADMSLM, dados)
8 Variables  (and intercept)
       Forced in Forced out
KDTM       FALSE      FALSE
MAM        FALSE      FALSE
MVM        FALSE      FALSE
TBCM       FALSE      FALSE
DDCM       FALSE      FALSE
DMSM       FALSE      FALSE
AEDM       FALSE      FALSE
ADMSLM     FALSE      FALSE
1 subsets of each size up to 8
Selection Algorithm: exhaustive
         KDTM MAM MVM TBCM DDCM DMSM AEDM ADMSLM
1  ( 1 ) "*"  " " " " " "  " "  " "  " "  " "
2  ( 1 ) "*"  " " " " " "  " "  "*"  " "  " "
3  ( 1 ) "*"  " " " " " "  "*"  "*"  " "  " "
4  ( 1 ) "*"  " " " " " "  "*"  "*"  "*"  " "
5  ( 1 ) "*"  " " " " " "  "*"  "*"  "*"  "*"
6  ( 1 ) "*"  " " " " "*"  "*"  "*"  "*"  "*"
7  ( 1 ) "*"  " " "*" "*"  "*"  "*"  "*"  "*"
8  ( 1 ) "*"  "*" "*" "*"  "*"  "*"  "*"  "*"
```

Figura 4: Utilização do leaps para escolha dos atributos mais relevantes para o output Task

3.2 Tratamento dos dados

Para garantir que os dados presentes nos registos que usamos para os casos de teste são representativos dos restantes, fizemos mais uma alteração no ficheiro de dados inicialmente fornecido, reorganizando a ordenação de cada linha da tabela de uma forma aleatória, garantindo assim que os dados não estão enviesados.

Este era o plano, porém acabamos por não aplicar esta alteração pois a mesma ou mantinha os resultados previamente obtidos ou inviabilizava todo o trabalho realizado sendo que em varios dos testes realizados não foram atingidos erros aceitáveis. Com esta aborgadem, os erros obtidos chegaram à ordem dos 100%.

3.3 Identificação da(s) topologia(s) de rede mais adequada(s)

Relativamente a esta parte, o que achámos mais adequado foi usar um método iterativo mudando a nossa topologia de rede e analisar de que maneira essas alterações iriam/poderiam influenciar o nosso resultado.

O nosso ponto de partida foi criar apenas uma camada intermédia, com apenas dois neurónios. Esta situação inicial serviu para vermos que alterações estariam a ser feitas no nosso erro final. De seguida, optámos por acrescentar mais camadas intermédias, assim como mais neurónios.

Os resultados foram bastante constantes, ou seja, o erro final não estava a sofrer grandes alterações significativas, mesmo com grandes números de neurónios e camadas. O que estava de facto a mudar eram o número de *steps*, o *error* e o tempo de aprendizagem desta rede.

De todos os testes realizados e após a análise dos resultados obtidos para cada hipótese, escolhemos para o *output Task* a rede que se apresenta abaixo. Essa rede é composta por duas camadas intermédias, sendo que a primeira tem quatro neurónios e a segunda tem dois.

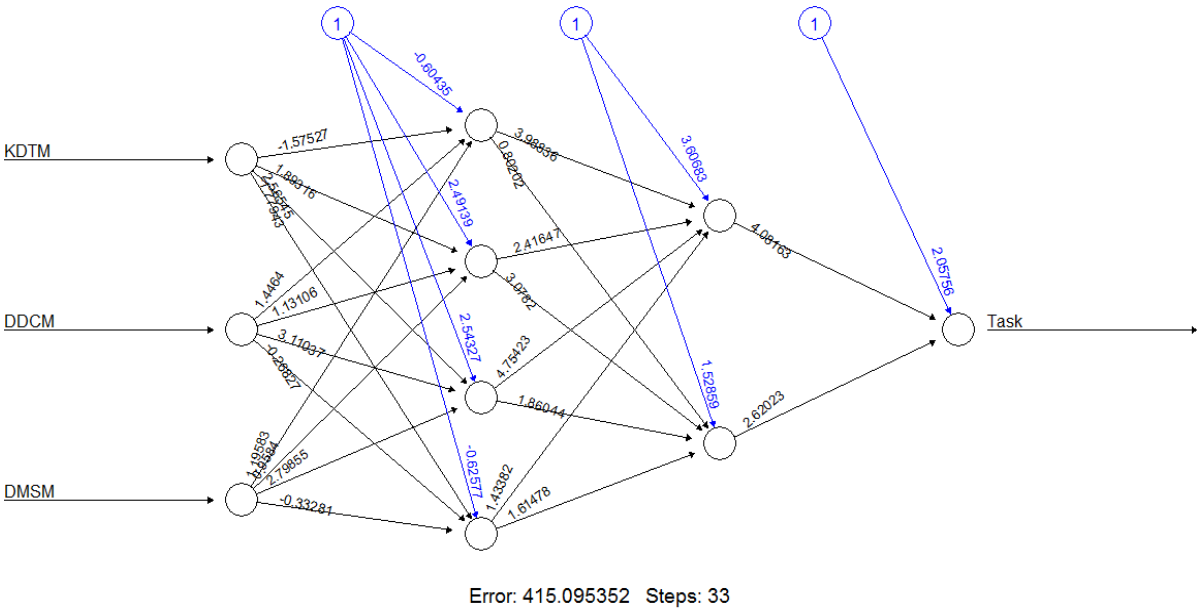


Figura 5: Rede neuronal para o output Task

Já para o *output FatigueLevel* a rede escolhida é composta por três camadas intermédias, a primeira com dez neurónios, a segunda com cinco neurónios e a terceira com três neurónios.

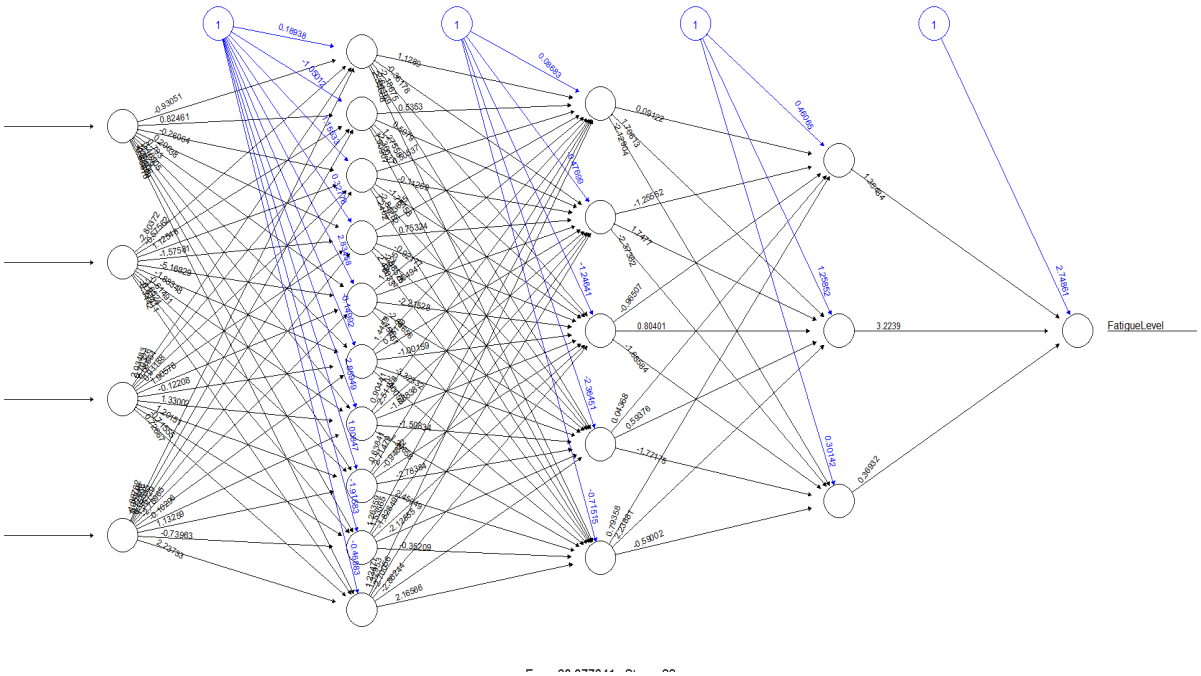


Figura 6: Rede neuronal para o output FatigueLevel

3.4 Seleção das regras de aprendizagem para treinar a(s) rede(s)

Para a criação das redes neuronais artifciais é necessário definir regras de treino de modo a obter uma rede que produza resultados consistentes. Isso pode ser conseguido no RStudio através do comando *neuralnet* que necessita de alguns argumentos, estando entre eles as regras de aprendizagem da rede.

O primeiro argumento recebido é a fórmula, que consiste na descrição do modelo que vai ser usado, ou seja, quais os atributos que corresponderão aos nodos de entrada e quais os atributos de *output* escolhidos. Tendo em conta a opção de criar duas redes, comecemos por analisar a rede para o *output Task*. Neste caso, a fórmula escolhida teve como atributos de *input* o *KDTMean*, o *DDCMean* e o *DMSMean*, sendo estes os três atributos mais relevantes obtidos pela utilização do biblioteca *leaps*. A escolha desta fórmula e deste número de atributos surge após vários testes em que se analisou o erro da rede, tendo sido esta a que obteve melhores resultados.

De seguida, aparece o argumento treino, que corresponde à porção dos dados utilizada para treinar a rede. Como de costume, após alguns testes, mas também após uma análise detalhada do ficheiro de dados, concluímos que das linhas 198 a 844 estaria o conjunto de dados de treino que permitira obter resultados mais consistentes, visto que possui uma maior variedade de resultados no *output* e um menor erro, ficando assim as restantes primeiras 197 linhas reservadas para os posteriores testes.

Depois, foi necessário decidir a topologia da rede relativamente ao número de camadas intermédias e nodos por cada uma dessas camadas. Após vários

testes com diferentes números de nodos e camadas chegamos à conclusão que este fator não estava a afetar significativamente o erro, não sofrendo qualquer variação. Posto isto, decidimos optar por uma rede com um reduzido número de nodos, pois isso permitiu um ganho de alguns décimos ou centésimos de segundo no tempo de treino da rede, sem qualquer contrapartida. Mais concretamente optamos por duas camadas intermédias, tendo a primeira quatro nodos e a segunda dois (hidden=c(4,2)).

O quarto argumento (lifesign="full") foi colocado para permitir obter informação no terminal sobre a rede enquanto esta é calculada, tendo sido apenas importante para saber o tempo de treino, embora nesta escala estejamos a lidar com tempos muitos pequenos que até poderiam ser considerados irrelevantes.

Temos ainda o quinto argumento (linear.ouput=FALSE) que está relacionado com os valores que o *output* toma. Neste caso, o *output* toma valores discretos, visto que tanto para a *Task* como para o *FatigueLevel*, temos valores numéricos específicos que representam no caso da *Task* tarefas específicas e no caso da *FatigueLevel* os vários níveis de fadiga, não podendo ser obtidos valores que não sejam algum daqueles que foram previamente definidos. Devido a isso, foi escolhida a opção FALSE, que é utilizada para quando o *output* toma uma classe de valores, em oposição à opção TRUE, utilizada para quando o *output* toma valores contínuos.

Por último, resta referir o atributo *threshold*, que afeta o critério de paragem da aprendizagem da rede. As variações deste parâmetro não consituíram variações significativas no erro das redes, por isso, mais uma vez, optamos pela versão que conduziu a um tempo de treino menor de entre os testes realizados e que é consistente com a versão mais utilizada nesta unidade curricular ao longo das aulas. Assim, o valor escolhido foi o 0.1.

```
> treino<-dados[198:844,]
> formula<-Task ~ KDTM + DDCM + DMSM
> rnaexaustao<-neuralnet(formula,treino,hidden=c(4,2),lifesign="full",linear.output = FALSE,threshold=0.1)
```

Figura 7: Comandos necessários para a implementação das regras de aprendizagem da rede cujo output é Task

Para a rede cujo *output* é o *FatigueLevel* o processo foi análogo, sendo que ocorreram variações em relação à rede já analisada na fórmula utilizada, pois o *output* é obviamente diferente e foram escolhidos quatro atributos baseados nos resultados da ferramenta *leaps*, ou seja, o *DDCMean*, o *MAMean*, o *MVMean* e *AEDmean*.

Também ocorreram modificações no número de nodos e camadas intermédias (c(10,5,3)) da rede numa tentativa de diminuir o erro desta. Assim, o maior número de neurónios e de camadas intermédias e consequente aumento do número de sinapses deveria aumentar a capacidade de aprendizagem da rede, o que traria melhores resultados, algo em que tivemos dificuldades.

Convém ainda referir que utilizamos para o treino desta rede as linhas 712 a 844, visto que continham uma grande variedade de casos, ficando assim reservadas as primeiras 711 linhas para casos de teste.

```
> treino<-dados[712:844,]
> formula<-FatigueLevel ~ MAM + MVM + DDCM + AEDM
> rnaexaustao<-neuralnet(formula,treino,hidden=c(10,5,3),lifesign="full",linear.output = FALSE,threshold=0.1)
```

Figura 8: Comandos necessários para a implementação das regras de aprendizagem da rede cujo output é FatigueLevel

3.5 Identificação da exaustão

Neste próximo capítulo, o nosso é objetivo é conseguirmos identificar o nível de exaustão em que a pessoa se encontra, sendo que esse nível se refere ao FatigueLevel.

Apresentamos de seguida os dois testes que queremos executar:

MAM	MVM	DDCM	AEDM
0.0363465191	4.167764e-02	9.900291e-02	-0.0071098412
0.15918549229637	0.199314993857410	0.240448907889	0.115046120572765

Código dos testes:

```
> formula<-FatigueLevel ~ MAM + MVM + DDCM + AEDM
> rnaexaustao<-neuralnet(formula,treino,hidden=c(10,5,3),lifesign="full",
linear.output = FALSE,threshold=0.1)
hidden: 10, 5, 3    thresh: 0.1    rep: 1/1    steps:      12 error: 60.09639
time: 0.09 secs
> teste <- data.frame(MAM=0.0363465191,MVM=4.167764e-02,DDCM=9.900291e-02,
AEDM=-0.0071098412)
> teste[2,] <- data.frame(MAM=0.15918549229637,MVM=0.199314993857410,
DDCM=0.240448907889,AEDM=0.115046120572765)
> rnaexaustao.results <- compute(rnaexaustao,teste)
> rnaexaustao.results
$neurons
$neurons[[1]]
  1           MAM           MVM          DDCM           AEDM
1 1 0.0363465191 0.0416776400 0.0990029100 -0.0071098412
2 1 0.1591854923 0.1993149939 0.2404489079  0.1150461206

$neurons[[2]]
  [,1]         [,2]          [,3]         [,4]
1    1 0.7106106451 0.6462027032 0.3526562206
2    1 0.6720848798 0.6478132570 0.4147480330
          [,5]          [,6]          [,7]         [,8]
1 0.6003811447 0.9684612737 0.3011136627 0.2049096851
2 0.4879857060 0.9586694462 0.3408499579 0.2313665480
          [,9]         [,10]         [,11]
1 0.4253785982 0.9490418434 0.8341539702
2 0.3459051158 0.9423096077 0.8365540408

$neurons[[3]]
  [,1]            [,2]          [,3]         [,4]
1    1 0.0004227445205 0.9993758628 0.9959460626
```

```
2    1 0.0006284709617 0.9991808552 0.9952907225
          [,5]         [,6]
1 0.9995155635 0.9996296990
2 0.9994596191 0.9995549956

$neurons[[4]]
  [,1]         [,2]         [,3]          [,4]
1    1 0.9995615329 0.9923769008 0.01714473476
2    1 0.9995608534 0.9923667596 0.01715368042


$net.result
          [,1]
1 0.9984479082
2 0.9984478669

> print(round(rnaexaustao.results$net.result))
  [,1]
1    1
2    1
```

Visto que o erro resultante do nosso *FatigueLevel* como *output* é demasiado elevado, era de esperar que os devidos testes não tivessem um resultado adequado, como tal aconteceu nestes (e mais alguns) testes por nós realizados, em que está a identificar o nível de exaustão como sendo sempre de nível 1, algo que já tentámos perceber como resolver, mas infelizmente sem sucesso.

3.6 Reconhecimento da tarefa em execução

De seguida estão a ser apresentados dois casos para reconhecermos qual a tarefa que está a ser executada.

No nosso primeiro caso, conseguimos identificar, como seria de esperar, que a tarefa corresponde à tarefa 1, já no segundo caso, surgui-nos algo que não estaríamos à espera, caso este que será explicado mais à frente.

KDTM	DDCM	DMSM
0.00006422618927	0.018949826004444	-0.01721424291766
1.954466e-04	0.113942555970	0.01972536166231

Exemplos:

```
> formula<-Task ~ KDTM + DDCM + DMSM
> rnaexaustao<-neuralnet(formula,treino,hidden=c(4,2),lifesign="full",
linear.output = FALSE,threshold=0.1)
hidden: 10, 5, 3    thresh: 0.1    rep: 1/1    steps:      23 error: 415.07335
time: 0.02 secs
> teste <- data.frame(KDTM=0.00006422618927,DDCM=0.018949826004444,
DMSM=-0.01721424291766)
> teste[2,] <- data.frame(KDTM=1.954466e-04,DDCM=0.113942555970,DMSM=0.01972536166231)
> rnaexaustao.results <- compute(rnaexaustao,teste)
```

```
> rnaexaustao.results
$neurons
$neurons[[1]]
  1             KDTM        DDCM           DMSM
1 1 0.00006422618927 0.018949826 -0.01721424292
2 1 0.00019544660000 0.113942556  0.01972536166

$neurons[[2]]
  [,1]         [,2]         [,3]         [,4]
1    1 0.8161895270 0.7417722294 0.2861084908
2    1 0.8557966229 0.7751163361 0.2721629361
          [,5]         [,6]         [,7]         [,8]
1 0.7488597898 0.5628167394 0.3449541436 0.6875740841
2 0.7444078488 0.5093528648 0.3566502300 0.7204406648
          [,9]         [,10]        [,11]
1 0.6285426022 0.08753307631 0.3554043788
2 0.5965398370 0.09546192091 0.3120899474

$neurons[[3]]
  [,1]            [,2]         [,3]         [,4]
1    1 0.0004051826150 0.9811548825 0.9819044745
2    1 0.0003754607954 0.9825038696 0.9834121905
            [,5]         [,6]
1 0.004744079753 0.1732110683
2 0.004568283727 0.1807949026

$neurons[[4]]
  [,1]         [,2]         [,3]         [,4]
1    1 0.9999340888 0.9631585244 0.9998813313
2    1 0.9999362440 0.9630236876 0.9998847948


$net.result
          [,1]
1 0.9998789768
2 0.9998789546

> print(round(rnaexaustao.results$net.result))
  [,1]
1    1
2    1
```

No segundo exemplo apresentado acima, mais concretamente o exemplo em que são atribuídos estes valores aos nodos de entrada KDTM=1.954466e-04, DDCM=0.113942555970 e DMSM=0.01972536166231, não obtivemos aquilo que seria de esperar. Para estes *inputs*, o valor esperado para o nodo de saída seria 2 sendo todavia obtido o valor 1, como é possível concluir pelo resultado referido acima. Isto realça o facto de as redes neuronais artificiais serem imprevisíveis.

4 Conclusões e Sugestões

Fazendo uma avaliação global da realização deste projeto, deparámo-nos com algumas dificuldades que são naturais da pouca experiência e da recente aprendizagem da linguagem R, mas que tentámos ultrapassar da melhor forma possível.

Em primeiro lugar, a deteção dos obstáculos do projeto que poderiam impedir o bom funcionamento da nossa rede, obstáculos alguns de baixo grau de dificuldade, como o tratamento dos dados iniciais, foram ultrapassados com uma certa facilidade. Porém, existiram "contratempos"maiores, como encontrar uma topologia de rede e casos de treino e de teste que nos permitissem o chegar ao menor erro, sendo este problema resolvido com um pouco de imaginação na criação de casos que, após analisados de forma concisa, nos pudessem dar um rumo para o caminho a percorrer para a desconstrução e consequente resolução do exercício em mãos.

Superado este percalço, todo o trabalho segue uma linha de raciocínio bastante semelhante à seguida nas aulas sendo que assim apenas existe a necessidade de importar essas capacidades aprendidas e utilizá-las de forma útil na resolução deste projeto. Todavia, apesar de realizarmos esta importação de conhecimentos não conseguimos chegar a resultados satisfatórios para estes problemas.

Em suma, fundamentamos as nossas escolhas, o que é notório na própria estrutura do trabalho, e apesar de algumas arestas por limar e resultados menos conseguidos estamos satisfeitos com o plano de ação que elaboramos e o método de trabalho imposto. Contudo, as conclusões não satisfazem os padrões pelos quais este grupo se tem regido, não conseguindo responder de forma sucinta e simplesmente correta ao problema em análise.

5 Bibliografia

- Textos pedagógicos disponibilizados na página da Unidade Curricular;
- "Biblioteca neuralnet- http://cran.r-project.org/web/packages/neuralnet/neuralnet.pdf;
- Perelli, Layne P. Fatigue Stressors in Simulated Long-Duration Flight. Effects on Performance, Information Processing, Subjective Fatigue, and Physiological Cost. No. SAM-TR-80-49. SCHOOL OF AEROSPACE MEDICINE BROOKS AFB TX, 1980;
- Pimenta A., Carneiro D., Novais P., Neves J., Detection of Distraction and Fatigue in Groups through the Analysis of Interaction Patterns with Computers, Intelligent Distributed Computing VIII, Springer-Verlag - Studies in Computational Intelligence, David Camacho, Lars Braubach, Salvatore Venticinque and Costin Badica (Eds) Vol. 570, pp 29-39, ISBN: 978-3-319-10421-8, 2014;
- Pimenta A., Carneiro D., Novais P., Neves J., Monitoring Mental Fatigue through the Analysis of Keyboard and Mouse Interaction Patterns, Hybrid Artificial Intelligent Systems - 8th International Conference HAIS 2013, Jeng-Shyang Pan, Marios M. Polycarpou, Michał Woźniak, André C. P. L. F. de Carvalho, Héctor Quintián, Emilio Corchado (eds), Lecture Notes in Computer Science, Vol 8073, ISBN 978-3-642-40845-8, pp 222-231, 2013;

6 Anexos

6.1 Exemplos de execução da aplicação

Nesta secção, é possível encontrar todos os comandos executados na ferramenta RStudio realizados para treinar as redes neuronais artificiais para resolver este exercício e calcular o seu erro.

Código relativo à rede com o *output Task*:

```
> teste<-dados[1:197,]
> treino<-dados[198:844,]
> formula<-Task ~ KDTM + DDCM + DMSM
> rnaexaustao<-neuralnet(formula,treino,hidden=c(4,2),lifesign="full",
linear.output = FALSE,threshold=0.1)
hidden: 10, 5, 3    thresh: 0.1    rep: 1/1    steps:      22 error: 415.08622
time: 0.02 secs
> teste.01<-subset(teste,select=c("KDTM","DDCM","DMSM"))
> rnaexaustao.resultados<-compute(rnaexaustao,teste.01)
> resultados<-data.frame(atual=teste$Task,
previsao=rnaexaustao.resultados$net.result)
> resultados$previsao<-round(resultados$previsao,digits=3)
> rmse(c(teste$Task),c(resultados$previsao))
[1] 0.1424941
```

Código relativo à rede com o *output FatigueLevel*:

```
> teste<-dados[1:711,]
> treino<-dados[712:844,]
> formula<-FatigueLevel ~ MAM + MVM + DDCM + AEDM
> rnaexaustao<-neuralnet(formula,treino,hidden=c(10,5,3),lifesign="full",
linear.output = FALSE,threshold=0.1)
hidden: 10, 5, 3    thresh: 0.1    rep: 1/1    steps:      32 error: 1139.58406
time: 0.23 secs
> teste.01<-subset(teste,select=c("MAM","MVM","DDCM","AEDM"))
> rnaexaustao.resultados<-compute(rnaexaustao,teste.01)
> resultados<-data.frame(atual=teste$FatigueLevel,
previsao=rnaexaustao.resultados$net.result)
> resultados$previsao<-round(resultados$previsao,digits=3)
> rmse(c(teste$FatigueLevel),c(resultados$previsao))
[1] 1.750979274
```