

**67660**

**67711**

**67724**

**67649**

**António Manuel Pereira do Anjo**

**Carlos Rafael Cruz Antunes**

**José Francisco Ferreira Alves de Sousa**

**Nuno André da Silva Oliveira**

**Universidade do Minho**

Escola de Engenharia

Licenciatura em Engenharia Informática

**Sistemas de Representação de Conhecimento e Raciocínio**

Ano Letivo de 2014/2015

**Relatório do Exercício 3**

# Índice

[1. Introdução 1](#_Toc419133125)

[2. Descrição das Tarefas 2](#_Toc419133126)

[2.1. Conhecimento Inicial 3](#_Toc419133127)

[2.2. Invariantes 4](#_Toc419133128)

[2.3. Manipulação do Conhecimento 5](#_Toc419133129)

[2.4. Jasper / Aplicação em Java 6](#_Toc419133130)

[3. Conclusões 9](#_Toc419133131)

1. Introdução

Neste relatório explicaremos como desenvolvemos a nossa script de R que nos permitiu resolver o problema proposto no exercício 3, que tem como objetivo utilizar, redes neuronais artificiais para processar grandes quantidades de informação.

Neste exercício foi-nos pedido que fosse encontrada uma rede neuronal artificial ideal que consiga prever o nível de fadiga de um utilizador, dados vários parâmetros. Para isto treinamos a rede previamente com uma amostra dada.

1. Descrição das Tarefas

Neste exercício trabalhamos com redes neuronais artificias, em R, para tentar calcular o nível de fadiga de um utilizador. Isto é conseguido devido ao treino prévio da rede neuronal, que é baseado num conjunto de dados iniciais. Depois da rede estar treinada, calcula-se então a fadiga do utilizador, recebendo alguns parâmetros de entrada:

* “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;

Estes parâmetros devem estar entre -1 e 1. Assim que os parâmetros sejam introduzidos a rede deve ser capaz de calcular uma aproximação da fadiga do utilizador, que varia de 1 a 7.

O nosso objetivo neste exercício é criar uma rede capaz de oferecer uma aproximação da fadiga, como explicado anteriormente.

* 1. Conhecimento Inicial

De forma a treinar a rede neuronal artificial para que esta possa ser usada para prever valores de fadiga tivemos que a treinar usando o “dataset” dado. Este “dateset” tem oito colunas relativas aos parâmetros de entrada indicados anteriormente uma coluna para o valor da fadiga, e uma última coluna para o tipo de tarefa que o utilizador estava a realizar na altura do teste.

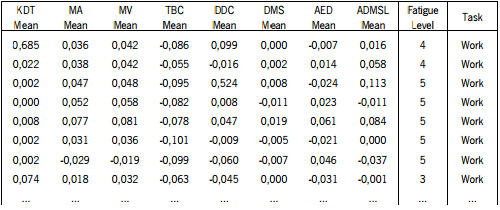


Tabela 1 - Excerto do *dataset* de treino

* 1. Invariantes

Para impedir que se adicione datas de fabrico quando isso nunca será possível, marcamos a palavra reservada como “nulo”, para que possa ser usada no respetivo invariante.

nulo(impossibru).

Depois de se ter marcado a palavra reservada como “nulo”, aplicamos os seguintes invariantes para não permitir que se adicione uma data de fabrico aos dois casos do conhecimento inicial.

+automovel(A,B,C,D,F,G,H) :: (

solucoes(

ANO,

(automovel('98-CV-32','Ford','Corcel','Vermelho',4,ANO,'Berta'),

nao(nulo(ANO))),

S),

comprimento( S,N ),

N == 0

).

+automovel(A,B,C,D,F,G,H) :: (

solucoes(

ANO,

(automovel('56-FG-89','Fiat','Uno','Bege',2,ANO,'Rui Gomes'),

nao(nulo(ANO))),

S),

comprimento( S,N ),

N == 0

).

Criamos também um invariante para não permitir adicionar um automóvel se a matrícula já se encontrar na base de dados e um invariante para manter o estado de conservação entre 1 e 10. O estado de conservação e o ano de fabrico são inteiros.

+automovel(Matricula,\_,\_,\_,\_,\_,\_) :: (

solucoes(Mat,automovel(Matricula,\_,\_,\_,\_,\_,\_),S),

comprimento( S,N ),

N == 1).

+automovel(\_,\_,\_,\_,Estado,\_,\_) :: (

number(Estado),

Estado =< 10,

Estado >= 1).

+automovel(\_,\_,\_,\_,\_,Ano,\_) :: (number(Ano)).

* 1. Manipulação do Conhecimento

De forma a manipular a base de dados, criou-se alguns predicados auxiliares. São todos simples e muitos deles foram utilizados nas aulas práticas.

nao( Questao ) :- Questao, !, fail.

nao( Questao ).

solucoes( X,Y,Z ) :- findall( X,Y,Z ).

comprimento( S,N ) :- length( S,N ).

inserir( Termo ) :- assert( Termo ).

inserir( Termo ) :- retract( Termo ),!,fail.

teste( [] ).

teste( [R|LR] ) :- R, teste( LR ).

remove( Termo ) :- retract( Termo ).

remove( Termo ) :- assert( Termo ),!,fail.

Para adicionar conhecimento ao sistema, foi criado o predicado “adicionar”, que verifica os invariantes correspondentes, e se tudo se verificar faz a inserção de um novo automóvel.

adicionar( automovel(A,B,C,D,E,F,G) ) :-

findall(Invariante,+automovel(A,B,C,D,E,F,G)::Invariante,Lista),

inserir( automovel(A,B,C,D,E,F,G) ),

teste( Lista ).

Para alterar informação de algum automóvel, criou-se o predicado “substituir”, que verifica que o termo a ser substituído existe e então remove-o e adiciona o novo termo, admitindo que se obedece a todos os invariantes.

substituir(automovel(Av,Bv,Cv,Dv,Ev,Fv,Gv),automovel(A,B,C,D,E,F,G)):-

automovel(Av,Bv,Cv,Dv,Ev,Fv,Gv),

remove( automovel(Av,Bv,Cv,Dv,Ev,Fv,Gv) ),

adicionar( automovel(A,B,C,D,E,F,G) ).

Para interrogar o sistema adaptamos o predicado “demo” dado nas aulas práticas para agora receber uma lista de pedidos e formular a respetiva resposta. Responde “verdadeiro” se todas as questões forem verdadeiras, responde “falso” ou “desconhecido” se uma das questões for falsa ou for de conhecimento imperfeito, respetivamente.

demo([],\_).

demo([Questao|Questoes],verdadeiro) :-

Questao,

nao( reservado(Questao)),

demo(Questoes,verdadeiro).

demo([Questao|Questoes],falso) :-

-Questao, !.

demo([Questao|Questoes],desconhecido) :-

nao( Questao ),

nao( -Questao ), !.

* 1. Jasper / Aplicação em Java

Como foi pedido no enunciado do trabalho prático, fizemos a ligação do SICStus Prolog ao JAVA, de maneira a que o utilizador final utilizasse o programa desenvolvido em JAVA para interagir com o Prolog.

A nossa aplicação possui um interface trivial, para o tratamento do conhecimento, assim como consulta e demonstração do mesmo. Além disso possui uma classe bastante trivial, chamada de “Automovel”, que pode ser equiparada ao predicado “automóvel” presente no nosso script Prolog.

A ligação com o SICStus é feita utilizando a biblioteca “Jasper”, e para simplificar o desenvolvimento do programa, criamos uma classe “Connection”, que trabalha com objetos do tipo “Automovel”. Dentro da classe “Connection” utilizamos os predicados desenvolvidos no Prolog para obter as mais diversas informações. De seguida, iremos apresentar alguns exemplos de métodos de comunicação com o SICStus.

Neste caso fornecemos a *query* e o mapa que mapeia as variáveis aos termos e obtemos uma resposta de verdadeiro ou falso recorrendo ao método *query* da biblioteca.

/\*\*

\* **Adiciona um automóvel 'a', utilizando o predicado 'adicionar'.**

\*

\* **@param** a Automóvel a adicionar.

\* **@return** Verdadeiro se adicionar com sucesso, falso caso contrário

\* **@throws** SPException Caso o Jasper encontre um problema.

\*/

public boolean **addAutomovel**(Automovel a) throws SPException {

String query = "adicionar(automovel(A,B,C,D,E,F,G)).";

Map map = new HashMap();

map.put("A", sicstus.newTerm(a.getMatricula()));

map.put("B", sicstus.newTerm(a.getMarca()));

map.put("C", sicstus.newTerm(a.getModelo()));

map.put("D", sicstus.newTerm(a.getCor()));

map.put("E", sicstus.newTerm(a.getEstado()));

map.put("F", sicstus.newTerm(a.getAno()));

map.put("G", sicstus.newTerm(a.getDono()));

return sicstus.query(query, map);

}

Já neste exemplo fornecemos a *query*, querendo obter os resultados que tornam este predicado verdadeiro, ou seja, os automóveis “verdadeiros”, utilizando o método *nextSolution* da classe *Query* do Jasper.

/\*\*

\* **Obtém todos os automóveis.** Utiliza o predicado

\* demo([automovel(A,B,C,D,E,F,G)],verdadeiro).

\*

\* **@return** Um map que associa cada matricula ao automóvel.

\* **@throws** SPException Caso o Jasper encontre um problema.

\* **@throws** InterruptedException Caso o Jasper encontre um problema.

\* **@throws** Exception Caso o Jasper encontre um problema.

\*/

public Map<String, Automovel> **getAutomoveis**() throws SPException, InterruptedException, Exception {

Map<String, Automovel> automoveis = new HashMap<>();

Map<String, Term> map = new HashMap();

Query q = sicstus.openPrologQuery(

"demo([automovel(A,B,C,D,E,F,G)],verdadeiro).", map);

while (q.nextSolution()) {

automoveis.put(map.get("A").toString(),

new Automovel(

map.get("A").toString(), map.get("B").toString(),

(String) map.get("C").toString(),

(String) map.get("D").toString(),

(int) map.get("E").getInteger(),

(int) map.get("F").getInteger(),

(String) map.get("G").toString()));

}

return automoveis;

}

De notar ainda que todo o código é acompanhado de o respetivo Javadoc, feito para organizar todo o projeto e o tornar mais compreensível.

1. Conclusões

Assim, através da implementação de conhecimento inicial e seguinte manipulação do mesmo utilizando várias técnicas e mecanismos de raciocínio adequados que nos permitiram trabalhar com conhecimento imperfeito num contexto de uma base de dados de automóveis, foi desenvolvido um sistema de representação de conhecimento e raciocínio com a capacidade de caraterizar um universo de comércio automóvel, assim como uma interface em Java que interaja com o sistema referido.