Centro Federal de Educação Tecnológica de Minas Gerais
Departamento de Computação
BACHAREL EM ENGENHARIA DE COMPUTAÇÃO

Laboratório de Inteligência Artificial

Pratica 4

Jônatas R. Tonholo

Belo Horizonte

julho de 2015

Sumário

1 Lógica Fuzzy

Foi escolhida um problema Fuzzy no qual decide-se entre ficar em casa, ir ao clube, ir ao cinema, que é a função utilizade, em relação à duas variáveis: Temperatura e Umidade relativa do Ar.

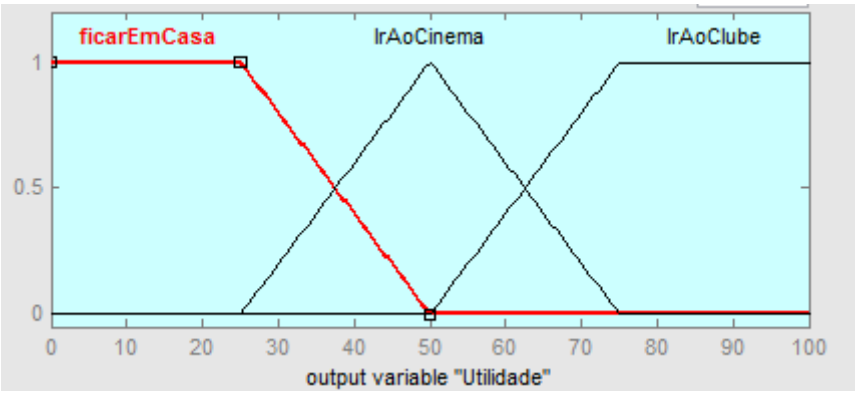

Figura 1 - Função Utilizade

Existem 4 possibilidades para a temperatura: Fria (0ºC a 12ºC), Média (12ºC a 24ºC), Quente (24ºC a 30ºC), Muito Quente (acima de 30ºC).

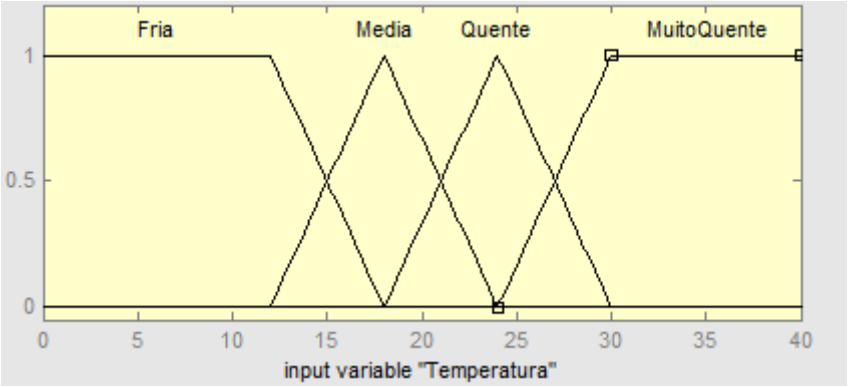

Figura 2 – Temperatura

Existem 3 possibilidades para Umidade Relativa do Ar: Baixa (até 25%), Normal (entre 25% e 75%) e Alta (acima de 85%).

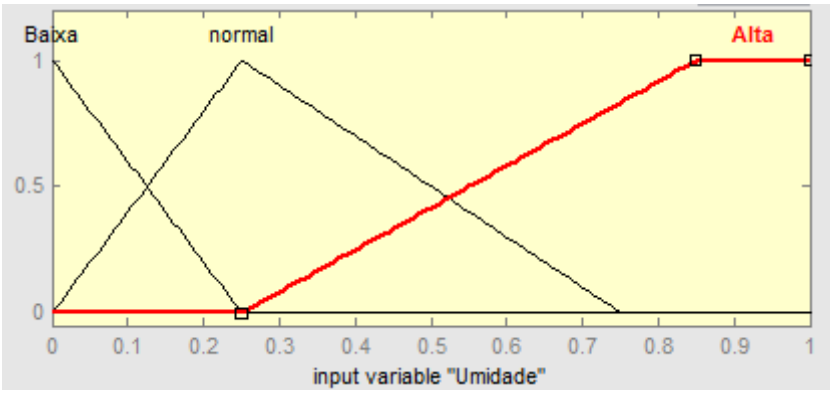

Figura 3 - Umidade Relativa do Ar

Regras

As regras a seguir definem qual decisão será tomada:

Se Temperatura **é** *MuitoQuente* **e** Umidade **é** *Alta* **Então** *FicarEmCasa*

Se Temperatura **é** *MuitoQuente* **e** Umidade **é** *Normal* **Então** *IrAoClube*

Se Temperatura **é** *MuitoQuente* **e** Umidade **é** *Baixa* **Então** *IrAoClube*

Se Temperatura **é** *Media* **e** Umidade **é** *Alta* **Então** *IrAoCinema*

Se Temperatura **é** *Media* **e** Umidade **é** *Normal* **Então** *IrAoCinema*

Se Temperatura **é** *Media* **e** Umidade **é** *Baixa* **Então** *FicarEmCasa*

Se Temperatura **é** *Fria* **e** Umidade **é** *Alta* **Então** *FicarEmCasa*

Se Temperatura **é** *Fria* **e** Umidade **é** *Normal* **Então** *IrAoCinema*

Se Temperatura **é** *Fria* **e** Umidade **é** *Baixa* **Então** *FicarEmCasa*

Surface

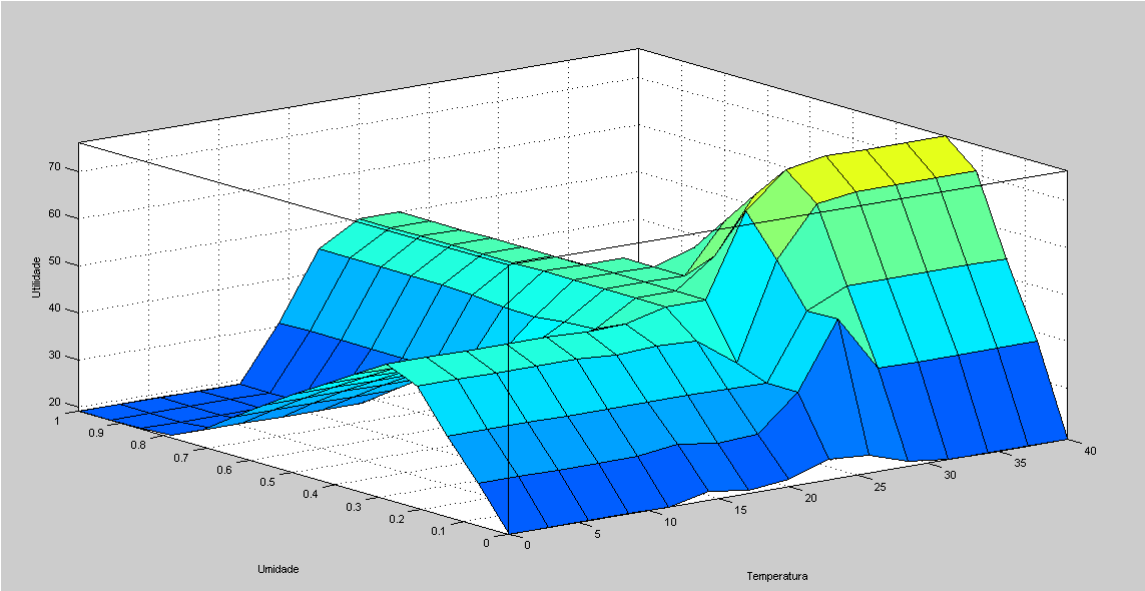

Figura 4 - Surface

Análise

A lógica Fuzzy é a melhor saída para modelar problemas imprecisos, nebulosos. Com ela, problemas que não eram discretos passam a ser modelados como discretos.

2 Perceptron

Foi modelado o Perceptron no Matlab, com taxa de aprendizado n=0.25, x_0=1 e vetor de pesos inicial W_i gerado aleatoriamente com números entre -0.5 e 0.5.

Para executar o código criado, no Matlab, mude para a pasta aonde o arquivo **and_or_perceptron.m** se encontra. Execute-o da seguinte maneira:

[X,AND,OR,W_AND,W_OR,Wi_AND,Wi_OR] = and_or_perceptron

A função retornará

- **X**: Matriz Binária na qual foram feitas as operações de OR e AND.
- **AND**: Matriz Coluna Binaria resultado da operação AND realizada pelo percetron.
- **OR:** Matriz Coluna Binaria resultado da operação OR realizada pelo percetron.
- **W_AND:** Vetor resultado W para AND.
- **W_OR**: Vetor resultado W para OR.
- **Wi_OR:** Vetor inicial W_i gerado aleatoriamente para OR.
- **W_AND:** Vetor inicial W_i gerado aleatoriamente para AND.

and_or_perceptron.m

```
function [X,AND,OR,W_AND,W_OR,Wi_AND,Wi_OR] = and_or_perceptron ()
    clc;close all; clear all;
    Error = 1;
    n = .25;
    x0 = 1;

    % Entradas
    X = [0 0 0;
         1 0 0;
         0 1 0;
         0 0 1;
         1 0 1;
         1 1 0;
         0 1 1;
         1 1 1];

    %Yd para And
    Yd_and = [0; 0; 0; 0; 0; 0; 0; 1];
    AND = zeros(size(X,1),1);

    % Yd para Or
    Yd_or = [0; 1; 1; 1; 1; 1; 1; 1];
    OR = zeros(size(X,1),1);

    W = createInitialW (size(X,2)+1);

    Wi_AND = W;

   %AND
    e=1;
    while e ~= 0
        e=0;
        idx = randperm(size(X,1));
        for i=1:size(X,1)
            xk=X(idx(i),1:end);
            yd=Yd_and(idx(i),1);
            trainning(xk,yd);
            if Error ~=0
                e=1;
            end
        end
    end
```

and_or_perceptron.m (Continuação)

```
    for ii=1:size(X,1)
        xk = X(ii,1:end);
        net = sum_func(xk);
        y = activation(net);
        AND(ii) = y;
    end
    W_AND = W;

    %Gerando novos W para OR
    W = createInitialW (size(X,2)+1);
    Wi_OR = W;

    %OR
    Error = 1;
    e=1;
    while e ~= 0
        e=0;
        idx = randperm(size(X,1));
        for j=1:size(X,1)
            xk=X(idx(j),1:end);
            yd=Yd_or(idx(j),1);
            trainning(xk,yd);
            if Error ~=0
                e=1;
            end
        end
    end


    for k=1:size(X,1)
        xk = X(k,1:end);
        net = sum_func(xk);
        y = activation(net);
        OR(k) = y;
    end
    W_OR = W;
    clc
%--- End of program
```

and_or_perceptron.m (Continuação)

```
% Perceptron Functions

    % Rand W function
    function W = createInitialW (size)
        W=zeros(1,size);
        for l=1:size
            r= (rand -.5);
            W(l) = r;
        end
    end

    %Step function
    function y = activation(net)
        if net <= 0
            y=0;
        else
            y=1;
        end
    end

    function net = sum_func(xk)
        net = 0;
        bias=W(1);
        net = net + (x0*bias);
        for m=1:size(xk,2)
            xki = xk(m);
            wi = W(m+1);
            net = net + (wi*xki);
        end
    end

    function y = trainning(xk,yd)
        net = sum_func(xk);
        y = activation(net);
        Error = (yd-y); %Delta Rule

        if Error ~= 0
            W(1) = W(1) + (n*Error*x0);
            for o=2:size(W,2)
                dW = (n*Error*xk(o-1));
                W(o) = W(o) + dW;
            end
        end
    end
end
```

Resultados Execução 1:	
X = 0 0 0 1 0 0 0 1 0 0 0 1 1 0 1 1 1 0 0 1 1 1 1 1 AND = 0 0 0 0 0 0 0 1 OR = 0 1 1 1 1 1 1 1	W_AND = -0.8667 0.2580 0.5386 0.2981 W_OR = -0.1622 0.3875 0.4561 0.3295 WI_AND = 0.3833 0.2580 0.2886 -0.2019 WI_OR = 0.3378 0.1375 0.4561 0.3295

Resultados Execução 2:	
X = 0 0 0 1 0 0 0 1 0 0 0 1 1 0 1 1 1 0 0 1 1 1 1 1 AND = 0 0 0 0 0 0 0 1 OR = 0 1 1 1 1 1 1 1	W_AND = -0.7569 0.4537 0.1921 0.1612 W_OR = -0.0555 0.3581 0.1523 0.1225 WI_AND = -0.2569 -0.2963 0.4421 -0.3388 WI_OR = 0.4445 -0.1419 -0.3477 -0.3775

Resultados Execução 3:	
X = 0 0 0 1 0 0 0 1 0 0 0 1 1 0 1 1 1 0 0 1 1 1 1 1 AND = 0 0 0 0 0 0 0 1 OR = 0 1 1 1 1 1 1 1	W_AND = -1.0582 0.3904 0.4777 0.4233 W_OR = -0.1738 0.2954 0.4149 0.4003 WI_AND = 0.4418 -0.3596 0.2277 -0.0767 WI_OR = 0.3262 0.2954 -0.0851 0.1503

O código será entregue juntamente com o relatório no arquivo compactado.

3 WEKA

Parte 1 – Árvores de Decisão

Executando o algoritmo com as opções padrões, o tamanho da árvore obtida foi de 207. A taxa de acerto para “Percentage Split” 66% foi de 92.1995 % e para “Percentage Split” 33% foi de 91.5667 %.

Após habilitar “reducedErrorPruning” o tamanho da árvore obtida foi de 161. A taxa de acerto para “Percentage Split” 66% foi de 92.4552 % e para “Percentage Split” 33% foi de 89.0042 %.

Ocorreu uma leve melhora na taxa de acerto para “Percentage Split” de 66% e uma leve piora para “Percentage Split” de 33%. Essa mudança não é significativa, entretanto o tamanho da árvore sofreu uma redução significativa pois é feita uma poda onde cada nó é substituído pela sua classe mais popular e a mudança só é mantida caso a eficácia da predição não tenha sido afetada.

Parte 2 – Bayes Ingênuo

Executando o algoritmo com as opções padrões, a taxa de acerto para “Percentage Split” 66% foi de 78.0051 % e para “Percentage Split” 33% foi de 78.0409 %.

Habilitando a opção “UseSupervisedDiscretization” a taxa de acerto para “Percentage Split” 66% foi de 90.3453 % e para “Percentage Split” 33% foi de 89.9124 %.

Foi uma melhora significativa. Ao utilizar a supervisão discretizada a dimensão do problema foi reduzida, com isso o algoritmo obteve melhores taxas de acerto, pois o Naive Bayes é sensível à dimensão do problema.

Parte 3 – Redes Neurais

A tabela 1 apresenta os resultados dos treinamentos das redes.

Tempo de Treino	n	Unidades na Camada Oculta					
		5		10		20	
100	0.1	1.8 s	89.3223 %	3.39 s	88.2353 %	6.25 s	88.8107 %
	0.3	1.8 s	85.8696 %	3.35 s	89.5141 %	6.27 s	88.8107 %
300	0.1	5.27 s	90.7928 %	9.78 s	89.1944 %	18.7 s	89.6419 %
	0.3	5.27 s	89.3223 %	9.74 s	90.0895 %	18.69 s	89.5141 %
500	0.8	8.86 s	93.0307 %	16.27 s	87.3402 %	31.34 s	88.4271 %
	1	8.95 s	85.422 %	16.31 s	87.4041 %	31.86 s	90.601 %

Tabela 1 – Treinamentos das RNS para n=0.1 e n=0.3, n=0.8, n=1, Tempo de treino de 100, 300 e 500 e Camadas ocultas 5, 10 e 20.

Analisando os dados para o tempo de treino igual a 100, a RNA com melhor taxa de acerto foi a de n=0.3, 10 camadas. Entretanto esse valor é insignificantemente maior que os obtidos nos outros treinamentos. Analisando os resultados obtidos para o tempo de treino igual a 300, a RNA com melhor taxa de acerto foi a de n=0.1, 5 camadas. A RNA n=0.3, 10 camadas foi a

segunda melhor, e um pouco melhor que para n=0.1. Mas estas melhoras são pouco significantes.

Realizou-se outros experimentos com tempo de treino igual a 500 e n=0.8 e n=1. Observou-se que a melhor RNA foi a de n=0.8, 5 camadas. Essa melhora foi aproximadamente 3% a mais que a melhor dos treinos anteriores.

Se compararmos as melhoras entre os treinos anteriores que era de menos de 1%, poder-se-ia inferir que aumentando o tempo de treino e a taxa de aprendizado, obter-se-ia resultados melhores, entretanto não foi o que ocorreu para n=1.

Referências

[1] Stuart. J. Russel and Peter Norving. Artificial Intelligence: A Modern Approach. Pearson Education, 2003.

[1] http://www.cs.waikato.ac.nz/ml/weka/, acessado em 08/07/2015