**EXERCÍCIO COMPUTACIONAL #2**

*UFMG - EEE920 Sistemas Nebulosos*

**TAIGUARA MELO TUPINAMBÁS &**

**MARIA CLARA CHENG SHUEN ALBUQUERQUE**

*14 de setembro de 2017*

## Introdução

O EC#2 consiste em desenvolver um classificador nebuloso com graus de certeza para classificação de tipos de lírios, conforme descrito em artigo de Ishibuchi e Nakashima 2001 [1].

Os dados utilizados foram baixados do site https://archive.ics.uci.edu/ml/datasets/Iris no dia 03/09/2017.

A primeira parte do algoritmo é, então, responsável por carregar os dados, normalizá-los e adequá-los para a lógica a ser utilizada.

```
%% Carregamento os dados
file = readtable('iris.txt','Delimiter',',');
X=[file.Var1 file.Var2 file.Var3 file.Var4];
Y=[file.Var5];

%normalização
for i=1:size(X,2)
    X(:,i)=(X(:,i)-min(X(:,i)))/(max(X(:,i))-min(X(:,i)));
end

Class1='Iris-setosa';
Class2='Iris-versicolor';
Class3='Iris-virginica';

%adiciona coluna com número das classes conforme legenda acima
X=[X strcmp(Y,Class1)+2*strcmp(Y,Class2)+3*strcmp(Y,Class3)];
```

Os parâmetros também são definidos no início do algoritmo

```
%% Parametros
K=5; %número de regras
tnorma=@times; %define a tnorma (times ou min)
par=tpar(K); %parâmetros da função triangular
```

A função `tpar(K)` retorna os parâmetros das funções triangulares de pertinência, conforme a quantidade de regras definida pelo usuário

```
function [par] = tpar(K)

for i=0:K-1
    par(i+1,1)=(i-1)/(K-1);
    par(i+1,2)=(i+1)/(K-1);
end
```

## Parte 1 – Regiões de Classificação para 2 dimensões

Para a primeira parte do exercício, foram utilizadas apenas as duas primeiras colunas da base de dados, correspondentes ao comprimento e largura das sépalas, respectivamente.

```
%% Teste bidimensional
X=[X(:,1:2) X(:,end)];
```

### 1.1 Calculando o modelo do classificador

Foi desenvolvida uma função para encontrar o modelo do classificador, que recebe como argumento a quantidade de regras **K**, os parâmetros das funções triangulares **par**, a matriz **X** com os atributos das plantas (comprimento e largura das sépalas) e a classe correspondente (1, 2 ou 3) e a tnorma a ser utilizada.

A função retorna dois vetores: número correspondente à classe consequente (**Ccon**); e os graus de certeza (**CF**) – para cada uma das $K\text{^}n$ regras.

O código se inicia calculando a cardinalidade de cada ponto para cada uma das $K\text{^}n$ regras, baseado na tnorma passada como argumento. Ao final dessa lógica, a matriz R conterá m linhas (quantidade de amostras do treinamento) por $K\text{^}n$ colunas (quantidade de regras). Cada elemento dessa matriz representa a cardinalidade daquele ponto para aquela regra.

Em seguida, é definida uma matriz C, que contém 3 colunas, uma para cada classe, em que cada linha terá 1 na coluna representada por sua classe e 0 nas demais. Essa matriz é utilizada para calcular a cardinalidade de cada classe na lógica seguinte, com um *loop* externo rodando para cada classe, e o interno rodando para cada regra. Assim, para cada regra, calcula-se o somatório das cardinalidades da classe em questão. Ao final, a matriz RC terá 3 linhas, para cada regra e $K\text{^}n$ colunas.

Com essa matriz calculada, encontra-se o valor máximo de β para cada regra (**Bcon)** e em qual índice ele se encontra (**Ccon**), que é uma das saídas da função – classe consequente de cada regra.

A última parte do código encarrega-se de calcular o grau de certeza, através das equações (14) e (15) do artigo.

```
function [Ccon, CF] = modelo2(K,par,X,tnorma)

%R -> Cardinalidade de cada ponto para cada regra
for i=1:K
    for j=1:K
        R(:,j+((i-1)*K))=tnorma(trimf(X(:,1),[par(j,1)
(par(j,1)+par(j,2))/2 par(j,2)]),trimf(X(:,2),[par(i,1)
(par(i,1)+par(i,2))/2 par(i,2)]));
    end
end


%matriz de 1 e 0, cuja coluna representa a classe
C(:,1)=X(:,end)==1;
C(:,2)=X(:,end)==2;
C(:,3)=X(:,end)==3;

%RC -> cardinalidade de cada classe para cada regra
for i=1:3 %quantidade de classes
    for j=i:K^2 %quantidade de regras
        RC(i,j)=sum(R(:,j)'*C(:,i));
    end
end



%encontra o valor do maior beta (Bcon)
%encontra o índice do maior beta (Ccon - classe consequente)
[Bcon, Ccon] = max(RC);
```

```
for i=1:K^2
    %calcula o Beta_barra da regra do artigo
    Bbarra=(sum(RC(:,i))-Bcon(i))/2; %soma todos Betas, exceto o refe-
rente à classe consequente

    %calcula o CF, conforme artigo
    CF(i)=(Bcon(i)-Bbarra)/sum(RC(:,i));
end
```

### 1.2 Definindo a função classificador

De posse do modelo, foi criada a função classificador, que recebe como argumento a quantidade de regras **K**, os parâmetros das funções triangulares **par**, a matriz **X** com os atributos das plantas (comprimento e largura das sépalas), as classes consequentes de cada regra **Ccon** e a tnorma a ser utilizada. A saída é um vetor com as classes de cada amostra passada como argumento.

O código cria uma matriz **R**, com **m** linhas e $K^n$ colunas, em que cada elemento representa a cardinalidade de cada amostra para cada regra, ponderada pelo grau de certeza daquela regra.

Em seguida é verificado qual o índice da regra de maior cardinalidade para cada amostra (**iclasse**), que é utilizado para definir as classes da matriz **X**, utilizando o argumento **Ccon** (classes correspondentes por regra).

```
function [classe] = classificador2(K,par,X,Ccon,CF,tnorma)

 for i=1:K
     for j=1:K
         R(:,j+((i-1)*K))=tnorma(trimf(X(:,1),[par(j,1)
(par(j,1)+par(j,2))/2 par(j,2)]),trimf(X(:,2),[par(i,1)
(par(i,1)+par(i,2))/2 par(i,2)]))... %vezes o mu de x2
         *CF(j+((i-1)*K)); %vezes o grau de certeza
     end
 end

 [~,iclasse]=max(R'); %encontra os índice das maiores pertinencias
 classe=Ccon(iclasse)'; %retorna um vetor com as classes de maior perti-
nencia
```

### 1.3 Código principal

Uma vez criadas as funções modelo e classificador para o problema bidimensional, a base de dados carregada foi modelada e classificada

```
%Ccon representa o a classe consequente de cada regra
%CF representa o grau de certeza
[Ccon, CF] = modelo2(K,par,X,tnorma);

%classe é um vetor com as classes de cada ponto
[classe] = classificador2(K,par,X,Ccon,CF,tnorma);
```

### 1.4 Código dos plots

Para encontrar as regiões de classificação, foi criado um grid de pontos (100x100) a ser classificado pelo modelo. Cada ponto foi plotado com uma cor diferente e, em seguida, os pontos da base de dados utilizada foram plotados em destaque, na mesma figura.

```
%% desenhando o grid
x1=0:0.01:1;
x2=0:0.01:1;
```

```
for i=1:size(x1,2)
    for j=1:size(x2,2)
        G(j+(i-1)*size(x1,2),1)=x1(i);
        G(j+(i-1)*size(x1,2),2)=x2(j);
    end
end

[classeG] = classificador2(K,par,G,Ccon,CF,tnorma);

G=[G classeG];

for k = 1 : length(classeG)
    if classeG(k)==1
        colorMap(k, :) = [1,0,0]; % Red
    elseif classeG(k)==2
        colorMap(k, :) = [0,0,1]; % Blue
    elseif classeG(k)==3
        colorMap(k, :) = [0,1,0]; % Green
    end
end

scatter(G(:,1),G(:,2),5,colorMap,'filled');
hold on;


%% desenhando os pontos da base de dados
for k = 1 : length(Y)
    if strcmp(Y(k),Class1);
        c1=plot(X(k,1),X(k,2),'ko-','MarkerFaceColor', 'r','Mark-
erSize',10);
        hold on;
    elseif strcmp(Y(k),Class2);
        c2=plot(X(k,1),X(k,2),'kd-','MarkerFaceColor', 'b','Mark-
erSize',10);
        hold on;
    elseif strcmp(Y(k),Class3);
        c3=plot(X(k,1),X(k,2),'ks-','MarkerFaceColor', 'g','Mark-
erSize',10);
        hold on;
    end
end

xlabel('Comprimento das sépalas (normalizado)');
ylabel('Largura das sépalas (normalizado)');
title('Regiões de Classificação para os Tipos de Lírios');
legend([c1 c2 c3], 'setosa', 'versicolor', 'virginica')
```

### 1.5 Resultados

O algoritmo foi processado para as tnorma “mínimo” e “produto” e para valores de K igual a 5, 10 e 25. O valor mais coerente de K seria 5, mas, por curiosidade, foram utilizados K igual a 10 e 25.

O resultado é apresentado na Figura 1. Observa-se pequenas diferenças ao se alterar a tnorma, mas grandes diferenças aumentando a quantidade de regras. A quantidade de erros cometida pelo classificador para os dados de treinamento reduz bastante, mas provavelmente para novas classificações o modelo não seria tão bom.

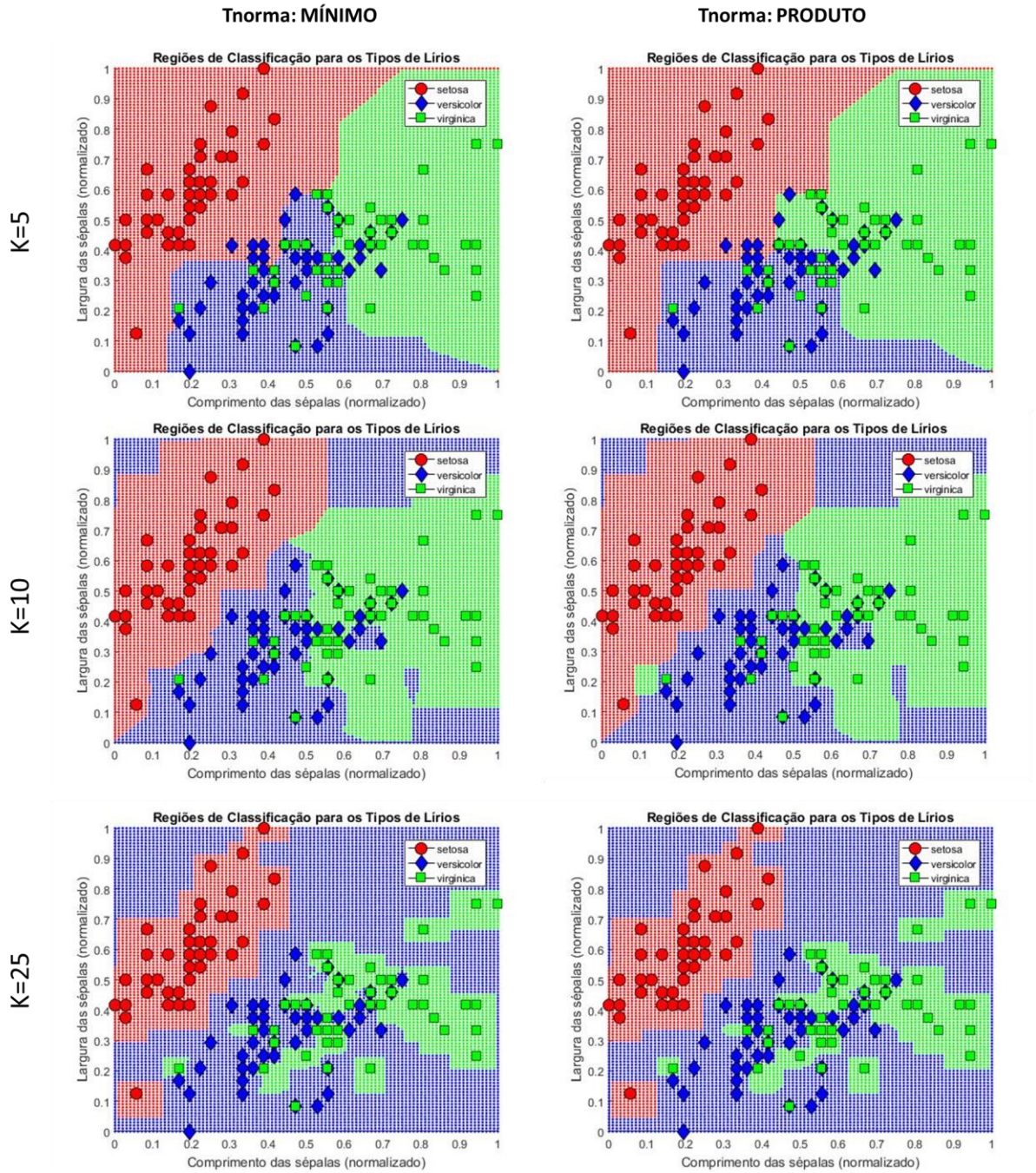


Figura 1 - Regiões de classificação para diferentes K e tnormas

## Parte 2 – Erro de Validação e Treinamento para 4 dimensões

Para o problema de quatro variáveis de entrada, foram feitas modificações nas funções modelo e classificador.

### 2.1 Calculando o modelo do classificador

A lógica do classificador para 4 dimensões é a mesma definida no item 1.2, para o problema bidimensional. Os *loops* para se encontrar as matrizes de cardinalidade tiveram de ser adaptadas, e foi criada uma matriz auxiliar **P**, com a pertinência de cada variável de entrada das amostras com as funções triangulares, para simplificar o código.

Mais uma vez, como saída, obteve-se dois vetores, um com as classes consequentes e o outro com os graus de certeza para cada uma das $K\hat{}n$ regras.

```
function [Ccon, CF] = modelo4(K,par,X,tnorma)

C=zeros(size(X,1),3);

C(:,1)=X(:,end)==1;
C(:,2)=X(:,end)==2;
C(:,3)=X(:,end)==3;

%R -> Cardinalidade de cada ponto para cada regra

for i=1:K
    P1(:,i)=trimf(X(:,1),[par(i,1) (par(i,1)+par(i,2))/2 par(i,2)]);
    P2(:,i)=trimf(X(:,2),[par(i,1) (par(i,1)+par(i,2))/2 par(i,2)]);
    P3(:,i)=trimf(X(:,3),[par(i,1) (par(i,1)+par(i,2))/2 par(i,2)]);
    P4(:,i)=trimf(X(:,4),[par(i,1) (par(i,1)+par(i,2))/2 par(i,2)]);
end

n=size(X,2)-1;
R=zeros(size(X,1),K^n);

for d=1:K
    ind1=(d-1)*(K^3)+1;
    R(:,ind1:ind1+(K^3-1))=repmat(P1(:,d),[1,(K^3)]);
    for c=1:K
        ind2=ind1+(c-1)*(K^2);
        R(:,ind2:ind2+(K^2-1))=tnorma(R(:,ind2:ind2+(K^2-1)),rep-
mat(P2(:,c),[1,(K^2)]));
        for b=1:K
            ind3=ind2+(b-1)*K;
            R(:,ind3:ind3+K-1)=tnorma(R(:,ind3:ind3+K-1),rep-
mat(P3(:,b),[1,K]));
            for a=1:K
               ind4=ind3+a-1;
               R(:,ind4)=tnorma(R(:,ind4),P4(:,a));
            end
        end
    end
end

%RC -> cardinalidade de cada classe para cada regra
for i=1:3 %quantidade de classes
    for j=i:K^n %quantidade de regras
        RC(i,j)=sum(R(:,j)'*C(:,i));
    end
end


%encontra o valor do maior beta (Bcon)
%encontra o índice do maior beta (Ccon - classe consequente)
[Bcon, Ccon] = max(RC);

for i=1:K^n
    %calcula o Beta_barra da regra do artigo
    Bbarra=(sum(RC(:,i))-Bcon(i))/2; %soma todos Betas, exceto o refe-
rente à classe consequente

    %calcula o CF, conforme artigo
    CF(i)=(Bcon(i)-Bbarra)/sum(RC(:,i));
end
```

### 2.2 Definindo a função classificador

As mesmas modificações da função de modelo foram aplicadas à função classificador para 4 variáveis de entrada.

```
function [classe] = classificador4(K,par,X,Ccon,CF,tnorma)

for i=1:K
    P1(:,i)=trimf(X(:,1),[par(i,1)  (par(i,1)+par(i,2))/2 par(i,2)]);
    P2(:,i)=trimf(X(:,2),[par(i,1)  (par(i,1)+par(i,2))/2 par(i,2)]);
    P3(:,i)=trimf(X(:,3),[par(i,1)  (par(i,1)+par(i,2))/2 par(i,2)]);
    P4(:,i)=trimf(X(:,4),[par(i,1)  (par(i,1)+par(i,2))/2 par(i,2)]);
end

n=size(X,2)-1;
R=zeros(size(X,1),K^n);

for d=1:K
    ind1=(d-1)*(K^3)+1;
    R(:,ind1:ind1+(K^3-1))=repmat(P1(:,d),[1,(K^3)]);
    for c=1:K
        ind2=ind1+(c-1)*(K^2);
        R(:,ind2:ind2+(K^2-1))=tnorma(R(:,ind2:ind2+(K^2-1)),rep-
mat(P2(:,c),[1,(K^2)]));
        for b=1:K
            ind3=ind2+(b-1)*K;
            R(:,ind3:ind3+K-1)=tnorma(R(:,ind3:ind3+K-1),rep-
mat(P3(:,b),[1,K]));
            for a=1:K
               ind4=ind3+a-1;
               R(:,ind4)=tnorma(R(:,ind4),P4(:,a))*CF(ind4);
            end
        end
    end
end

 [~,iclasse]=max(R'); %encontra os índice das maiores pertinencias
 classe=Ccon(iclasse)'; %retorna um vetor com as classes de maior perti-
nencia
```

### 1.3 Código principal

Desta vez, o código principal foi realizado utilizando-se um *loop* para registrar os erros de validação e treinamento para cada iteração.

Em cada iteração, a matriz X era aleatorizada e a mesma matriz era utilizada para todos as quantidades de regras K. O modelo foi testado utilizando-se o método de validação cruzada, i.e. treinamento com as primeiras 105 linhas (70%) da matriz aleatória e depois utilizado para classificar as 45 (30%) linhas restantes, como validação. Os dados utilizados no treinamento também foram classificados de acordo com o modelo obtido, para efeito de comparação.

As quantidades de acertos de validação e treinamento, alcançadas pelo algoritmo, eram então registradas em duas matrizes (uma para validação e outra para treinamento) que, ao final, eram compostas por **i** linhas (quantidade de iterações) e 9 colunas (quantidade de regras iteradas, no caso de 2 a 10).

Finalmente, as médias de acerto de validação e treinamento foram calculadas para cada valor de K iterado.

```
%% loop para verificar performance em função de K
tic();

for i=1:100

    %randomiza as linhas
    ind=randperm(size(X,1));
    for j=1:size(X,1);
        Xrand(j,:)=X(ind(j),:);
    end

    for K=2:10 %variações de K
        par=tpar(K);

        %Bcon representa o Beta da classe consequente (máximo)
        %Ccon representa a classe consequente
        [Ccon, CF] = modelo4(K,par,Xrand(1:105,:),tnorma); %treina com
70% dos dados

        %Ccon representa o a classe consequente de cada regra
        [classe_valid] = classifica-
dor4(K,par,Xrand(106:end,:),Ccon,CF,tnorma); %testa com os 30% restantes
        [classe_trein] = classifica-
dor4(K,par,Xrand(1:105,:),Ccon,CF,tnorma); %testa com os 30% restantes

        acertos_valid=classe_valid==Xrand(106:end,end);
        desemp_valid(i,K-1)=sum(acertos_valid)/45;

        acertos_trein=classe_trein==Xrand(1:105,end);
        desemp_trein(i,K-1)=sum(acertos_trein)/105;
    end
end

%calcula as médias de acertos (desempenho)
desemp_valid_=num2cell(desemp_valid, 1);
desemp_valid_avg=cellfun(@mean,desemp_valid_);
desemp_trein_=num2cell(desemp_trein, 1);
desemp_trein_avg=cellfun(@mean,desemp_trein_);


disp('Execution Time')
disp(toc())
```

**1.4 Código dos plots**

Para se obter o gráfico de erro, foi utilizado o seguinte código.

```
%% plots

k=2:10;

plot(k,1-desemp_valid_avg,'r',k,1-desemp_trein_avg,'b');
xlabel('K - quantidade de regras');
ylabel('Erro de classificação');
legend('Validação','Treinamento');
title('Desempenho do classificador VS quantidade de regras');
```

### 1.5 Resultados

Para 100 iterações por quantidade de regras, e com a quantidade de regras variando de 2 a 10, o resultado obtido pode ser conferido na Figura 2. Como esperado, o erro de treinamento tende a reduzir continuamente com o aumento de **K**, enquanto que para os dados de validação, o erro parece se estabilizar e crescer levemente para valores altos de **K**.

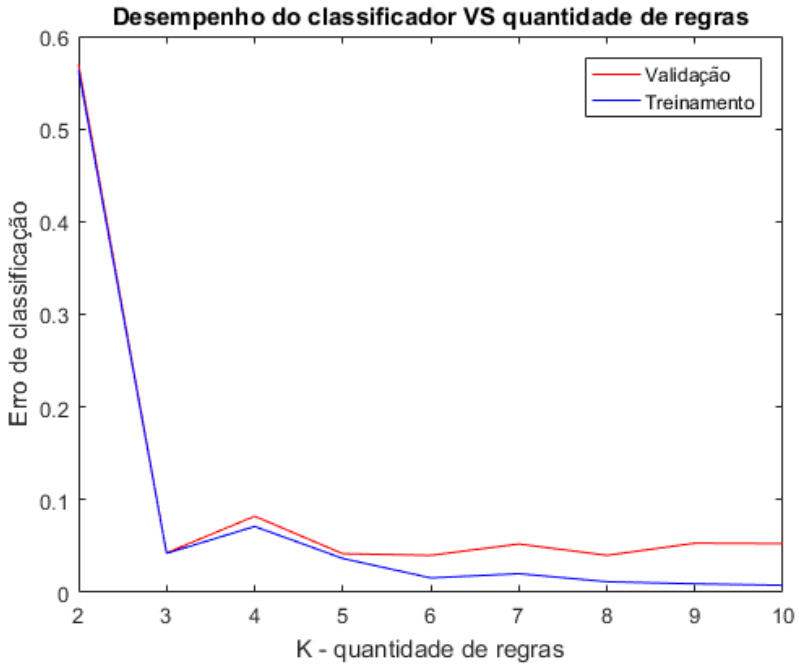


Figura 2 – Erro do classificador por validação cruzada, de validação (vermelho) e treinamento (azul)

Para se observar melhor o efeito de crescimento do erro de validação para altos valores de K, o algoritmo foi rodado para um K variando de 2 a 15, com a mesma quantidade de iteração por K (100), produzindo o resultado apresentado na Figura 3. Desta vez, observa-se melhor a tendência crescente do erro de validação em função do aumento de K, enquanto que o erro de treinamento chega a 0, para um K maior ou igual a 11 – como era de se esperar.

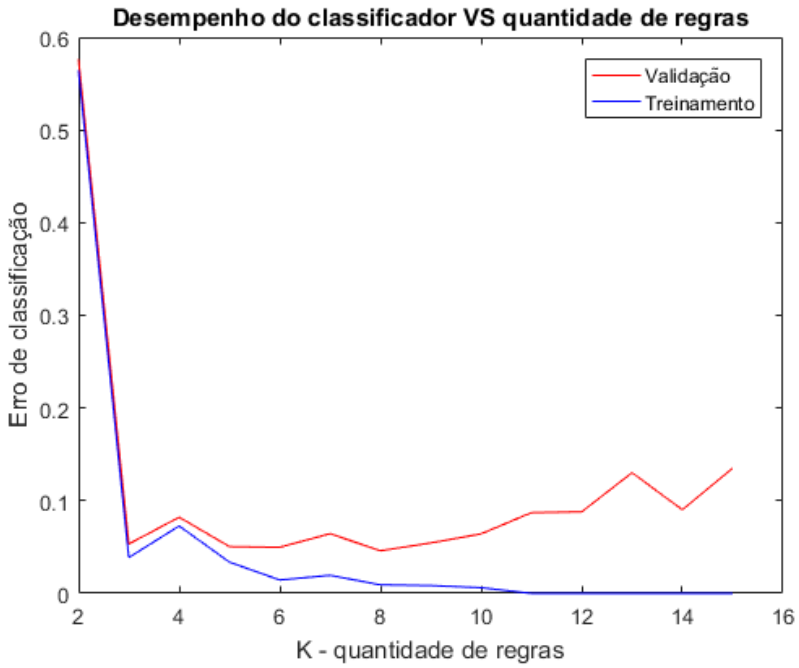


Figura 3 - Erro do classificador por validação cruzada, de validação (vermelho) e treinamento (azul)

## Bibliografia

[1] H. Ishibuchi and T. Nakashima, “Effect of Rule Weights in Fuzzy Rule-Based,” *IEEE Trans. FUZZY Syst.*, vol. 9, no. 4, pp. 506–515, 2001.

[2] K. Nozaki, H. Ishibuchi, and H. Tanaka, “Adaptive fuzzy rule-based classification systems,” *IEEE Trans. Fuzzy Syst.*, vol. 4, no. 3, pp. 238–250, 1996.