# Faculdade de Engenharia da Universidade do Porto


# Sistema de classificação para diagnóstico

Diogo Silva up201809213

Fábio Morais up201504257

Relatório do Trabalho Prático realizado no âmbito da unidade curricular
Sistemas Baseados em Inteligência Computacional do
Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Data de entrega do Relatório 05/06/2020

# Índice

# 1 Introdução

Este problema tem como objectivo avaliar a qualidade da soldadura recorrendo a técnicas de radiografia. Os dados para este problema podem ser encontrados em “A "Soft"K-Nearest neighbour voting scheme”. Os dados apresentados no artigo seguem estrutura semelhante ao seguinte exemplo:

| # | $u_i$ | $v_i$ | $w_i$ | Class |
|---|---|---|---|---|
| 1 | 0.6471 | 0.0171 | 0.2039 | Weld |
| 2 | 0.6471 | 0.0156 | 0.3059 | Weld |
| 3 | 0.8824 | 0.0279 | 0.3020 | Weld |

Figura 1: Exemplo de formato dos dados

Nesta tabelas os valores de $u_i$, $v_i$ e $w_i$, vão ser usados para conseguir classificar a solda. Com este trabalho temos como objectivo analisar e implementar 4 diferentes métodos de classificação, e avaliar os seus resultados. Os métodos a analisar são os seguintes: "soft K-nearest neighbour", "fuzzy K-nearest neighbour", "sistema neuro-difuso"e "rede neuronal".

# 2 Implementação e analise dos diferentes algoritmos

## 2.a Soft K-nearest neighbour

K-nearest neighbour tem como principio de funcionamento avaliar os K vizinhos mais próximos, e com base nas classe destes decidir a que classe pertence o novo ponto. Cada vizinho próximo tem direito a um voto, no fim a classe com mais votos será a classe atribuída ao novo ponto.
Por forma a tentar explicar o algoritmo apresentamos agora uma explicação presente no artigo a cima citado. Tendo $Q_i$ ,$i$ $\varepsilon\{1,2,3,4,5\}$, em que $Q_2$e $Q_3$ pertencem á classe "X"e os restantes pertencem a classe "Y". A distancia dos $Q_i$ ao novo ponto ("P") é a seguinte: $d_1 < d_2 < d_3 < d_4 < d_5$. Matematicamente teremos o seguinte

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}, R = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad (1)$$

Em que a matriz A, é a classificação dos pontos conhecidos, coluna um classe "X"coluna 2 classe "Y". A matriz R é referente a distancia crescente entre o ponto de teste e todos os outros pontos com classes conhecidas. Seguindo o exemplo do artigo faremos $K = 3$, para isso $w = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \end{bmatrix}^T$
Fazendo:

$$i^* = argmax(w^T RA) \quad (2)$$

Temos que:

$$= argmax(\begin{bmatrix}2 & 1\end{bmatrix}) = 1 \tag{3}$$

Então a classificação final será "X".
Porem como se pode ver no artigo, a classificação final varia muito dependendo do K que se escolher. Por exemplo para K=4 teríamos uma indeterminação e para K=5 a classe do ponto de teste seria "Y". O Soft K-nearest neighbour tenta minimizar a variação causada pelo K. Para isso usa um método de votação diferente, que consiste na fuzificação da matriz R , mantendo a matriz A com a mesma estrutura. Posto isto temos agora a seguinte adaptação a formula (2):

$$i^* = argmax(w^T \widetilde{R} A) \tag{4}$$

Onde a matriz $\widetilde{R}$ é a fuzificação da matriz R. Analisando o ponto 2 (dois) do artigo dado obtemos a forma de implementar o que foi referido acima. Para implementar este algoritmo foram seguidos estas etapas Em primeiro lugar podemos começar por fruzificar as distancias crespas atribuindo um numero difuso $D_i$ a cada uma das distancias. Podemos para isso aplicar a seguinte expressão:

$$D_i(x) = \frac{N(|d_i - x|, \sigma_p)}{N(0, \sigma_p)} \tag{5}$$

Apesar de os valores fuzzy de $D_x$ poderem ter qualquer forma unimodal escolhemos usar a forma Gaussiana. O valor de $\sigma_p^2$ pode ser calculado da seguinte forma:

$$\sigma_p^2 = \frac{\sum_{i=1}^{K}(d_i - \overline{d})^2}{K} \tag{6}$$

O valor de $\overline{d}$ (media das distancias) segue a seguinte expressão:

$$\overline{d} = \sum_{i=1}^{K} \frac{d_i}{K} \tag{7}$$

O segundo passo seria calcular a matriz auxiliar $\alpha$ com a seguinte expressão:

$$\alpha_{ij} = S(D_i, D_{(j)}) \tag{8}$$

Em que S foi definido como sendo Max-Min para o cálculo da matriz auxiliar. $D_{(j)}$ é o valor difuso da j maior distancia.

Em terceiro lugar temos de normalizar as linhas da matriz $\alpha$ e depois as suas colunas. Podemos fazer isso da seguinte maneira:

$$\alpha_{ij}^{(r+1)} = \frac{\alpha_{ij}^{(r)}}{\sum_{l=1}^{M} \alpha_{ij}^{(r)}} \tag{9}$$

Segue agora a normalização das colunas:

$$\alpha_{ij}^{(r+2)} = \frac{\alpha_{ij}^{(r+1)}}{\sum_{l=1}^{M} \alpha_{ij}^{(r+1)}} \tag{10}$$

Por ultimo devemos repetir a terceira etapa ate convergir, sendo a nova matriz definida por:

$$R = \alpha_{ij}^{(\infty)} \tag{11}$$

### 2.b Fuzzy K-nearest neighbour

Este algoritmo ao contrario do Soft K-nearest neighbour, fuzifica a matriz A em vez da matriz R. a equação (4) fica agora com o seguinte formato:

$$i^* = argmax(w^T R\widetilde{A}) \tag{12}$$

Tal como no método anterior também existe muitas maneiras de fuzificar a matriz A. Iremos usar uma *membership* Gaussiana. Deste modo teremos de ter K conjuntos difusos para os K vizinhos mais próximos Denotar que como a classificação da soldadura é entre 0 e 1, o intervalo para as distancias basta ser entre 0 e $\sqrt{3}$ visto que a maior distancia é $\sqrt{3}$ (Hipotenusa é igual a raiz quadrada da soma dos catetos).
O mesmo se verifica no método anterior (Soft K-nearest neighbour).

### 2.c Sistema neuro-difuso

Para este método decidimos implementar um sistema adaptativo usando o método de inferência de Takagi-Sugeno. Este sistema reune conceitos de redes neuronais e de lógica difusa. Para a estrutura deste método será a seguinte. Na primeira camada teremos as três entradas. Essa entradas vão ligar a segunda camada onde será feita a sua fuzificação e determinadas as suas funções de pertença de cada uma. Depois disso a camada seguinte, onde estão as regras, ira determinar quais regras serão activadas, este resultado é depois normalizado com base na intensidade de todas as regras. Em ultimo lugar é feita a desfuzificação e obtido o resultado final. Este conceito pode ser visualizado na figura abaixo.
Para aplicar isto no *matlab* começamos por usar o ide *anfisedit* para ter uma ideia de como podíamos estruturar a rede. A estrutura da rede que usamos, para o artigo foi a seguinte:

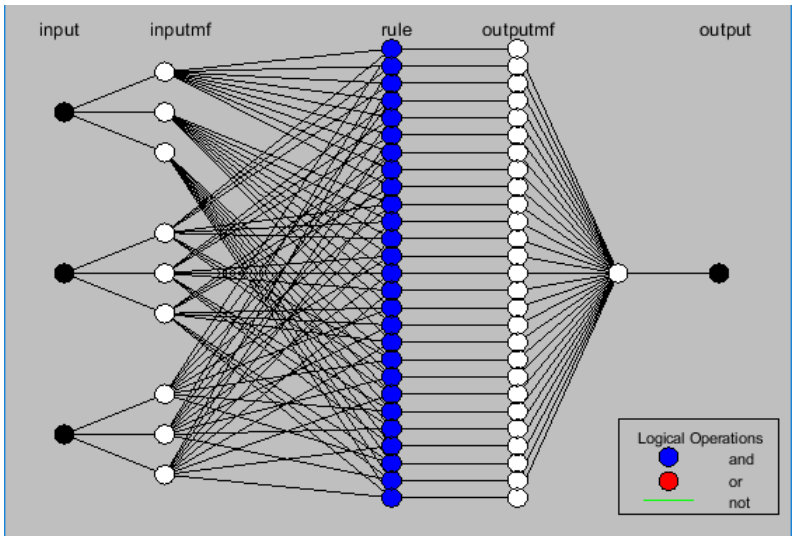


Figura 2: Estrutura MET3

Depois de testar e deferir alguns parâmetros, implementamos este algoritmo com a função do matlab anfis. Esta função recebe os valores de input e as suas respectivas saídas, e usa esses dados para dar *tune* no sistema de inferência baseado em Takagi-Sugeno criado pela própria. Após o treino é devolvido o respectivo FIS (*Fuzzy inference system*) para que possamos usar para avaliar os inputs cuja saída é desconhecida. Tendo agora o FIS podemos então calcular as saídas desconhecidas usando a função evalfis. Esta função recebe os inputs em que a saída é desconhecida e usando o sistema FIS que lhe passamos consegue prever uma saída. Como estes valores não tinham nenhuma classe atribuída( no caso do artigo soldado ou não soldado) criamos uma função que classifica como um os valores maiores que 0.5, e como zero os valores menores que 0.5. Podemos ver em baixo a classificação da nossa rede para os dados do artigo, em que obtemos 11 erradas

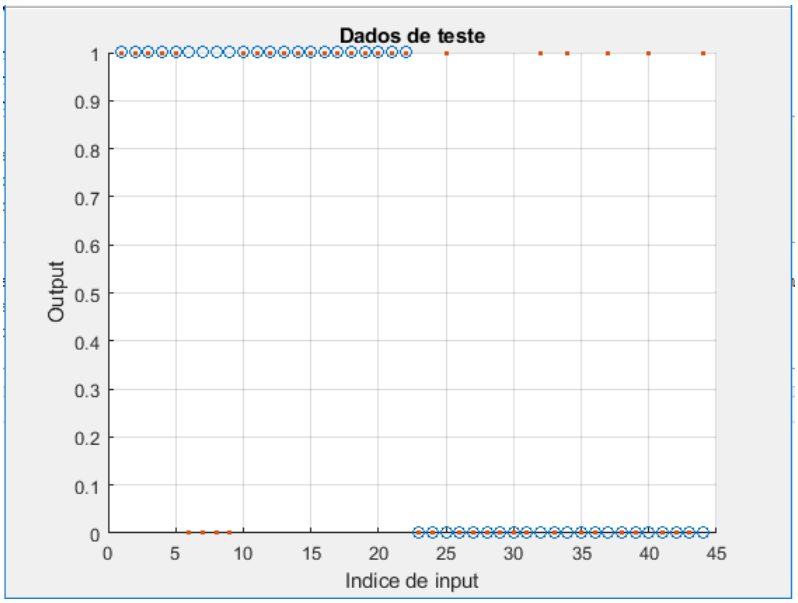


Figura 3: Exemplo de resultado obtido no MET3

## 2.d Rede neuronal

Em semelhança ao que já foi feito na ficha 7 desta mesma unidade curricular, iremos também implementar uma rede neuronal. A composição da rede será de 3 entradas, 5 neurónios na primeira camada e 3 neurónios na segunda camada. A função de activação escolhida foi a função hiperbólica e a taxa de aprendizagem a usar será de 0.2
Iremos treinar a nossa rede ao colocar na entrada valores em que conhecemos o valor da sua saída.

$$e_k(n) = d_k(n) - z_k(n) \tag{13}$$

O erro é a diferença entre o valor desejado e a saída. Se o neurónio $k$ for diferente da saída desejada ira existir um erro $e_k(n)$
O objetivo da aprendizagem é, normalmente, minimizar o custo da função baseada no erro.

Foi usado o método de *BackPropagation* para calcular os novos pesos. Este algoritmo calcula o erro ocorrido na saída e recalcula o valor dos pesos.

$$error = (expected - output) * transferDerivative(output) \tag{14}$$

Dessa forma iremos calcular os novos pesos como é representado a seguir:

$$w_i = w_i + \eta * error * x_i \tag{15}$$

O estrutura da rede neural é a seguinte:

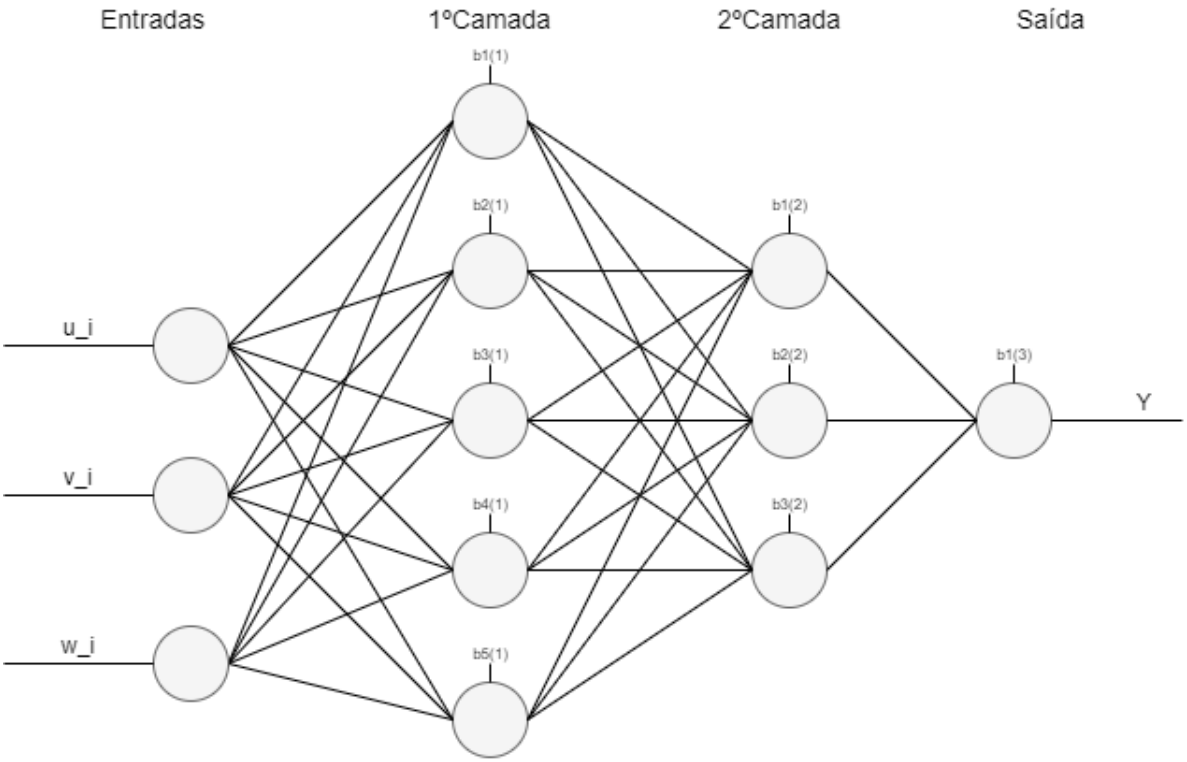


Figura 4: Rede neuronal

## 3 Discussão de resultados

Iremos agora apresentar os resultados obtidos, com a execução de cada um dos respectivos algoritmos, Denotar que no Soft KNN e no Fuzzy KNN os dados consistem em:
Erradas | Indeterminadas | Corretas.

| Dados | K=3 | K=4 | K=5 | K=7 | Nº de amostras |
|---|---|---|---|---|---|
| Artigo | 6 \| 0 \| 38 | 4 \| 6 \| 34 | 7 \| 0 \| 37 | 7 \| 0 \| 37 | 44 |
| TestInput11A.txt | 0 \| 0 \| 20 | 0 \| 0 \| 20 | 0 \| 0 \| 20 | 0 \| 0 \| 20 | 20 |
| TestInput11B.txt | 0 \| 1 \| 49 | 0 \| 0 \| 50 | 0 \| 0 \| 50 | 0 \| 0 \| 50 | 50 |
| TestInput11C.txt | 0 \| 0 \| 40 | 0 \| 0 \| 40 | 0 \| 0 \| 40 | 0 \| 0 \| 40 | 40 |

Tabela 1: Resultados Soft KNN

| Dados | K=3 | K=4 | K=5 | K=7 | Nº de amostras |
|---|---|---|---|---|---|
| Artigo | 9 \| 1 \| 34 | 9 \| 0 \| 35 | 9 \| 0 \| 35 | 8 \| 0 \| 36 | 44 |
| TestInput11A.txt | 0 \| 1 \| 19 | 0 \| 1 \| 19 | 0 \| 1 \| 19 | 0 \| 0 \| 20 | 20 |
| TestInput11B.txt | 0 \| 2 \| 48 | 0 \| 0 \| 50 | 0 \| 0 \| 50 | 0 \| 0 \| 50 | 50 |
| TestInput11C.txt | 0 \| 0 \| 40 | 0 \| 0 \| 40 | 0 \| 0 \| 40 | 0 \| 0 \| 40 | 40 |

Tabela 2: Resultados Fuzzy KNN

| Teste | Nº iterações | Nº de erros teste | Nº de amostras | Sucesso teste(%) |
|---|---|---|---|---|
| artigo | 10000 | 10 | 44 | 77.27% |
| testInput11A.txt | 1000 | 0 | 20 | 100% |
| testInput11B.txt | 1000 | 0 | 50 | 100% |
| testInput11C.txt | 1000 | 0 | 40 | 100% |

Tabela 3: Resultados Neuro-Fuzzy

| Teste | Nº iterações | Sucesso treino(%) | Nº de erros teste | Nº de amostras | Sucesso teste(%) |
|---|---|---|---|---|---|
| artigo | 1000 | 86.1% | 3 | 44 | 93.2% |
| testInput11A.txt | 1000 | 97.0% | 0 | 20 | 100% |
| testInput11B.txt | 1000 | 90.8% | 2 | 50 | 96.0% |
| testInput11C.txt | 1000 | 99.5% | 0 | 40 | 100% |

Tabela 4: Resultados da Rede Neuronal

Apresentamos agora a percentagem de acerto de cada um dos métodos em ordem decrescente de acerto:

| Algoritmo | Percentagem de acertos(%) |
|---|---|
| Rede Neuronal | 96.75% |
| Soft KNN k=3 | 95.46% |
| Soft KNN k=5 | 95.46% |
| Soft KNN k=7 | 95.46% |
| Fuzzy KNN k=7 | 94.16% |
| Neuro-Fuzzy | 93.5% |
| Soft KNN k=4 | 93.5% |
| Fuzzy KNN k=4 | 93.5% |
| Fuzzy KNN k=5 | 93.5% |
| Fuzzy KNN k=3 | 93.5% |

Tabela 5: Percentagem de acerto dos algoritmos

Com a analise destas tabelas, reparamos que o "Soft"KNN tem uma taxa de acertos superior ao Fuzzy KNN. Porem não podemos deixar de realçar o facto que o Soft KNN, demorava muito mais tempo a executar que o Fuzzy KNN. Em termos práticos um sistema que precise de uma resposta rápida não poderá usar o Soft KNN, nesse caso seria melhor usar o Fuzzy KNN que é bastante mais rápido. Com estes resultados podemos concluir então que o Soft KNN tem mais precisão mas é também mais demorado, já o Fuzzy KNN tem menos precisão mas também é menos demorado.
O facto de termos escolhido K=7 foi para tentar mostrar que com K ímpar exite menos probabilidade de termos um valor indeterminado, visto que funciona por votação a probabilidade de dar indeterminado num numero ímpar é mais baixa que num par. Porem não ficou muito visível. Mas mesmo assim podemos ver no caso do Soft KNN para os dados do artigo em que tivemos seis indeterminações para K=4 e zero para os restantes.
Analisando agora o sistema Neuro Fuzzy, em comparação com os métodos anteriores podemos ver que este é um bocado menos preciso que o Soft KNN, e de precisão igual ao Fuzzy KNN, o seu tempo de processamento é bastante demorado também. Porem após obtermos o FIS o processo de obtenção do output é rápido, logo para sistemas em que a velocidade de resposta é necessária este método será uma boa opção, pois após o treino a sua execução é rápida.
Por ultimo temos a Rede neuronal, esta foi a que obtivemos o melhor resultado, porem dado a aleatório na inicialização dos pesos temos momentos em que temos uma maior existência de acertos, e momentos em que os erros são maiores. Tal como a Neuro Fuzzy, também aqui temos um grande tempo perdido no treino da rede mas a sua resposta ao dados de teste é também rápida.
Posto isto podemos concluir que os quer a Neuro Fuzzy, quer a rede neuronal depois de treinadas são as mais rápidas a dar a resposta. Para um sistema em que seja viável ter um tempo de arranque para treinar as redes este é um bom sistema. Porem este tipo de sistema precisa de muitos dados, como podemos ver nas tabelas para o artigo se existisse mais dados para treino a rede teria uma melhor resposta. Quantos mais dados existirem mais lentos serão os KNN.

## 4 Conclusões

Dado que conseguimos aplicar todos os métodos, com uma elevada taxa de acertos, concluímos que os objectivos do trabalho foram concluídos. De realçar novamente que se no artigo os dados fornecidos para treino fossem em maior numero teríamos um ainda melhor desempenho.

Concluímos este trabalho com um maior conhecimento sobre métodos de *machin learning* e inteligência computacional, bem como implementados na pratica e como os podemos usar na resolução de problemas. Ficamos assim com umas bases mais fortes sobre este tema.

## 5 Referências

1. Mitchell HB, Schaefer PA. A “soft” K-nearest neighbor voting scheme. Int J Intelligent Systems 2001; 16:459-468.

# 6 Anexos

## 6.a Rede neuronal

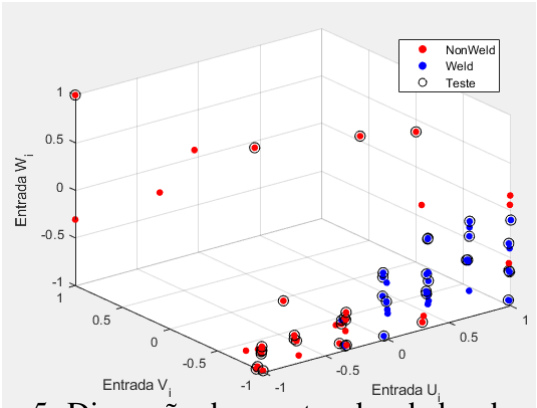

Figura 5: Dispersão dos pontos dos dados do artigo

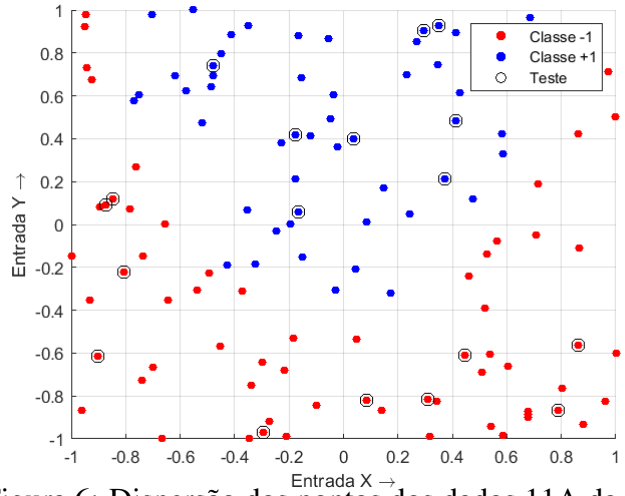

Figura 6: Dispersão dos pontos dos dados 11A da TP7

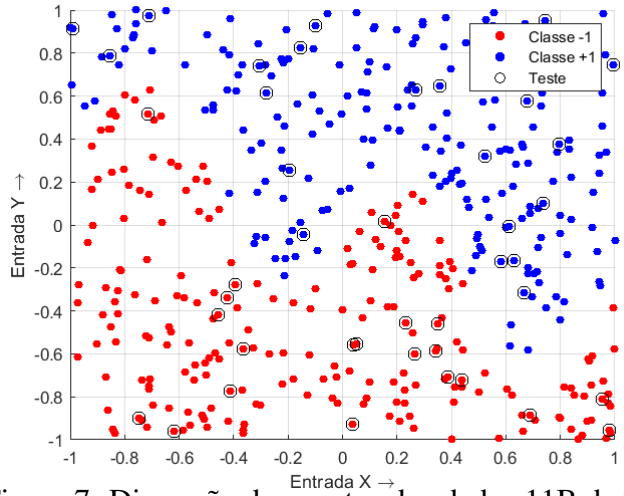

Figura 7: Dispersão dos pontos dos dados 11B da TP7

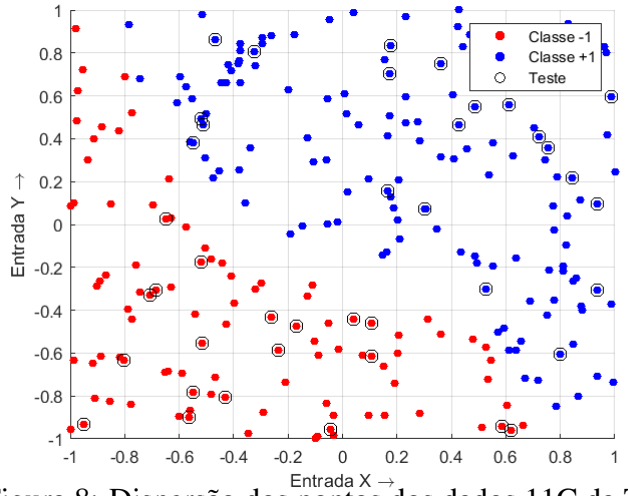

Figura 8: Dispersão dos pontos dos dados 11C da TP7

### 6.b Soft KNN Dados TP7

```
%##############################################
%                    SOFT KNN                  #
%Desenvolvido por :                            #
%Diogo Silva u201809213                        #
%Fabio Morais u201504257                       #
%                                              #
%##############################################

%% Limpar consola
close all;
clear all;
clc;
%% Ler dados
Input = readmatrix("Dados\P2\testInput11C.txt");
Output = readmatrix("Dados\P2\testOutput11C.txt");
%% Escolha de variaveis padrao
indeterminados=0;
erradas=0;
corretas=0;
K=7;
%% fun oes auxiliares
normalizar_out = @(y) (y+1)/2;
%% Calcular tamanho dos vetores de dados
tamanho_data_treino = find(ismember(Input,[0 0 0],'rows'))-1; %
    Encontrar [0 0 0] para calcular o tamanho dos dados de
    treino
tamanho_data_teste = length(Input)-tamanho_data_treino-1; %
    Tamanho dos dados de teste
%% Criar vetores de dados de input
Dados_Treino = Input(1:tamanho_data_treino,:);
Dados_Teste = Input(tamanho_data_treino+2:tamanho_data_treino+
```

```
    tamanho_data_teste+1,1:2);
%% Criar e normalizar dados de output para treino
Dados_Treino_Output_norm = normalizar_out(Dados_Treino(:,3));
Output= normalizar_out( Output);
%% Algoritmo
for i=1:tamanho_data_teste % Para cada ponto em que nao
    conhe a o seu valor de output vai tentar descobir o seu
    valor atravez dos seu vizinhos

    distancia = zeros(1,tamanho_data_treino);%vetor que contem
       as distancias entre os pontos conhecidos e o ponto
       desconhecido

    % 1 calcular distancias
    for j=1:tamanho_data_treino % Hip^2=cat1^2+cat2^2
        distancia(1,j) = sqrt((Dados_Teste(i,1)- Dados_Treino(j
           ,1))^2 +(Dados_Teste(i,2)- Dados_Treino(j,2))^2);
    end

    % 2 encontrar pontos mais proximos
    A = zeros(K,2);% Matriz A coluna 1 contem a classe "Nao
       soldado ->0" a coluna dois tem "Soldado ->1"
    menorDist = zeros(1,K); %guarda as menores distancias
    for j=1:K % seleciona as K menores distancias
        [menorDist(1,j),indexMinDist] = min(distancia);%Vai
           busacar a menor distancia e devolve o numero e a
           posicao no vetor
        distancia(1,indexMinDist) = 10000000000;%Para nao
           voltar a ir buscar a mesma distancia metemos essa
           como "infinita"
        if Dados_Treino_Output_norm(indexMinDist,1)== 1 %se a
           classe se for 1
            A(j,2)=1;%insere na coluna 2
        else % senao
            A(j,1)=1;  %insere na coluna 2
        end
    end

    media = mean(menorDist); %Calcular a media das menores
       distancias para calcualar o sigma
    sumatorio = 0; %Para calcular a primeira parte do sigma (
       formula 6 do relatorio)

    % 3 Calcular Sigma
    for j=1:K
        sumatorio= sumatorio + (menorDist(1,j)  - media)^2; %
           Sumatorio (d_i - d_medio)^2
    end
```

```
62      sigma=sqrt( sumatorio/K); %sigma
63
64      Di=[];%Numero difuso distancia crescente
65      Dj=[];%Numero difuso distancia decrescente
66      D0=[];%Numero difuso com [sigma, 0]    o denominador da
           formula (5)
67      aux=K;%para conseguir calcular para o numero difuso Dj, no
           for faremos calcular de forma decrescente decrementando
           esta variavel
68      % 4 calcular os numeros difusos para as distancias
69      for j=1:K %criar numerador e denuminador
70          Di=[Di;gaussmf(0:0.0001:sqrt(3),[sigma menorDist(1,j)])
               ]; %numerador eq (5)
71          Dj=[Dj;gaussmf(0:0.0001:sqrt(3),[sigma menorDist(1,aux)
               ])]; %Para termos tambem a ordem decrescente para
               fazer a matriz aauxiliar alpha
72          aux=aux-1;
73          D0=[D0;gaussmf(0:0.0001:sqrt(3),[sigma 0])];%
               denominador da formula (5)
74      end
75      Di=Di./(D0+1e-100);%calculo final do numero difuso eq(5)
           adicionamos um numero muito pequeno para nao dar erro a
           dividir por zero
76      Dj=Dj./(D0+1e-100);
77
78      alpha=zeros(K,K);%matriz auxiliar iremos calcular de acordo
           com a formula (8)
79      for j=1:K
80          for l=1:K
81              alpha(j,l)=max(min(Di(j,:),Dj(l,:))); % fizemos S
                   como sendo max min
82          end
83      end
84      teste_converge=zeros(K,K);%para verifica se a matriz alpha
           ja convergiu, ou seja se a alpha antiga    igual a nova
           entao terminamos
85      R=zeros(K,K);%Matriz R
86      while 1
87          %normalizar linhas de alpha
88          for j=1:K
89              SumatorioLinhas = sum(alpha(j,:))+1e-100;% sumar
                   numero muito pequeno para pervenir que tenhamos
                   uma divisao por zero no passo seguinte
90              for l=1:K
91                  alpha(j,l)=alpha(j,l)/SumatorioLinhas;%equa ao
                       (9)
92              end
93          end
```

```
94          %normalizar colunas de alpha
95          for j=1:K
96              Sumatoriocolunas = sum(alpha(:,j))+1e-100;% sumar
                    numero muito pequeno para pervenir que tenhamos
                    uma divisao por zero no passo seguinte
97              for l=1:K
98                  alpha(l,j)=alpha(l,j)/Sumatoriocolunas;%
                        equa ao (10)
99              end
100         end
101
102         %verificar se convergiu se nao faz isto ate convergir
103         if((round(teste_converge,10)==round(alpha,10))) %Quando
                as primeiras 10 casas decimais nao mudarem entao
               convergiu
104             R= alpha;%se convergir a matriz R esta determinada
                    e podemos sair do ciclo
105             break;
106         end
107         teste_converge=alpha;%guarda o antigo valor de alpha
               para comparar
108     end
109     W = ones(1,K);
110     resultado = W * R *A; %equa ao (4)
111     [Valor,index]= max(resultado);%equa ao (4)
112     if resultado(1,1)==resultado(1,2)%Se as duas colunas
           tiverem 1 o algoritmo escolheu as duas classes como
           valor final logo uma indetermina ao
113         index=0;
114         indeterminados=indeterminados+1;
115     elseif(Output(i)==index-1)
116         corretas=corretas+1;
117     else
118        erradas = erradas+1;
119     end
120 end
121 corretas
122 erradas
123 indeterminados
```

### 6.c Soft KNN Dados Artigo

```
1 %##############################################
2 %                    SOFT KNN                  #
3 %Desenvolvido por :                            #
4 %Diogo Silva u201809213                        #
5 %Fabio Morais u201504257                       #
6 %                                              #
7 %##############################################
```

```
8
9  %% Limpar consola
10 close all;
11 clear all;
12 clc;
13
14 %% Ler dados
15
16 TesteInput = readmatrix("Dados\Artigo\testInput.txt");
17 TesteOutput = readmatrix("Dados\Artigo\testOutput.txt");
18 Treino= readmatrix("Dados\Artigo\treino.txt");
19 %% Escolha de variaveis padrao
20 indeterminados=0;
21 erradas=0;
22 corretas=0;
23 K=4;
24 %% fun oes auxiliares
25 normalizar_out = @(y)(y);% (y+1)/2;
26 %% Calcular tamanho dos vetores de dados
27
28 tamanho_data_treino = length(Treino); %Tamanho dos dados de
       treino
29 tamanho_data_teste = length(TesteInput); %Tamanho dos dados de
       teste
30
31 %% Criar e normalizar dados de output para treino
32 Dados_Treino_Output_norm = normalizar_out(Treino(:,4));
33 TesteOutput= normalizar_out(TesteOutput);
34 %% Algoritmo
35
36 for i=1:tamanho_data_teste % Para cada ponto em que nao
       conhe a o seu valor de output vai tentar descobir o seu
       valor atravez dos seu vizinhos
37
38     distancia = zeros(1,tamanho_data_treino);%vetor que contem
          as distancias entre os pontos conhecidos e o ponto
          desconhecido
39
40     % 1 calcular distancias
41     for j=1:tamanho_data_treino % Hip^2=cat1^2+cat2^2
42         distancia(1,j) = sqrt((TesteInput(i,1)- Treino(j,1))^2
               + ...
43         (TesteInput(i,2)- Treino(j,2))^2+(TesteInput(i,3)-
               Treino(j,3))^2);
44     end
45
46     % 2 encontrar pontos mais proximos
47     A = zeros(K,2);% Matriz A coluna 1 contem a classe "Nao
```

```
        soldado ->0" a coluna dois tem "Soldado ->1"
48      menorDist = zeros(1,K); %guarda as menores distancias
49      for j=1:K % seleciona as K menores distancias
50          [menorDist(1,j),indexMinDist] = min(distancia);%Vai
                busacar a menor distancia e devolve o numero e a
                posicao no vetor
51          distancia(1,indexMinDist) = 10000000000;%Para nao
                voltar a ir buscar a mesma distancia metemos essa
                como "infinita"
52          if Dados_Treino_Output_norm(indexMinDist,1)== 0 %se a
                classe se for 1
53              A(j,1)=1;%insere na coluna 2
54          else % senao
55              A(j,2)=1;  %insere na coluna 2
56          end
57      end
58      media = mean(menorDist); %Calcular a media das menores
            distancias para calcualar o sigma
59      sumatorio = 0; %Para calcular a primeira parte do sigma (
            formula 6 do relatorio)
60      % 3 Calcular Sigma
61      for j=1:K
62          sumatorio= sumatorio + (menorDist(1,j)  - media)^2; %
                Sumatorio (d_i - d_medio)^2
63      end
64      sigma=sqrt( sumatorio/K); %sigma
65      Di=[];%Numero difuso distancia crescente
66      Dj=[];%Numero difuso distancia decrescente
67      D0=[];%Numero difuso com [sigma, 0]    o denominador da
            formula (5)
68      aux=K;%para conseguir calcular para o numero difuso Dj, no
            for faremos calcular de forma decrescente decrementando
            esta variavel
69      % 4 calcular os numeros difusos para as distancias
70      for j=1:K %criar numerador e denuminador
71          Di=[Di;gaussmf(0:0.0001:sqrt(3),[sigma menorDist(1,j)])
                ]; %numerador eq (5)
72          Dj=[Dj;gaussmf(0:0.0001:sqrt(3),[sigma menorDist(1,aux)
                ])]; %Para termos tambem a ordem decrescente para
                fazer a matriz auxiliar alpha
73          aux=aux-1;
74          D0=[D0;gaussmf(0:0.0001:sqrt(3),[sigma 0])];%
                denominador da formula (5)
75      end
76      Di=Di./(D0+1e-100);%calculo final do numero difuso eq(5)
            adicionamos um numero muito pequeno para nao dar erro a
            dividir por zero
77      Dj=Dj./(D0+1e-100);
```

```
78
79      alpha=zeros(K,K);%matriz auxiliar iremos calcular de acordo
            com a formula (8)
80      for j=1:K
81          for l=1:K
82              alpha(j,l)=max(min(Di(j,:),Dj(l,:)));% fizemos S
                    como sendo max min.
83                                                      %Com o min
                                                           iremos
                                                           criar um
84                                                      %array em que
                                                            em cada
                                                           posicao
85                                                      % sera o
                                                           valor
                                                           minimo
86                                                      % entre di e
                                                           dj, depois
87                                                      % disso
                                                           escolhemos
                                                            o
88                                                      % maior valor
                                                            desses
89          end
90      end
91      teste_converge=zeros(K,K);%para verifica se a matriz alpha
            ja convergiu, ou seja se a alpha antiga    igual a nova
            entao terminamos
92      R=zeros(K,K);%Matriz R
93      while 1
94          %normalizar linhas de alpha
95          for j=1:K
96              SumatorioLinhas = sum(alpha(j,:))+1e-100;% sumar
                    numero muito pequeno para pervenir que tenhamos
                    uma divisao por zero no passo seguinte
97              for l=1:K
98                  alpha(j,l)=alpha(j,l)/SumatorioLinhas;%equa ao
                        (9)
99              end
100         end
101         %normalizar colunas de alpha
102         for j=1:K
103             Sumatoriocolunas = sum(alpha(:,j))+1e-100;% sumar
                    numero muito pequeno para pervenir que tenhamos
                    uma divisao por zero no passo seguinte
104             for l=1:K
105                 alpha(l,j)=alpha(l,j)/Sumatoriocolunas;%
                        equa ao (10)
```

```
106             end
107         end
108
109         %verificar se convergiu se nao faz isto ate convergir
110         if((round(teste_converge,10)==round(alpha,10))) %Quando
                as primeiras 10 casas decimais nao mudarem entao
               convergiu
111             R= alpha;%se convergir a matriz R esta determinada
                   e podemos sair do ciclo
112             break;
113         end
114         teste_converge=alpha;%guarda o antigo valor de alpha
               para comparar
115     end
116     W = ones(1,K);
117     resultado = W * R *A; %equa ao (4)
118     [Valor,index]= max(resultado);%equa ao (4)
119     if resultado(1,1)==resultado(1,2)%Se as duas colunas
           tiverem 1 o algoritmo escolheu as duas classes como
           valor final logo uma indetermina ao
120         index=0;
121         indeterminados=indeterminados+1;
122     elseif(TesteOutput(i)==index-1)
123         corretas=corretas+1;
124     else
125         erradas = erradas+1;
126     end
127 end
128 corretas
129 erradas
130 indeterminados
```

### 6.d Fuzzy KNN Dados TP7

```
1  %##############################################
2  %                   FUZZY KNN                  #
3  %Desenvolvido por :                           #
4  %Diogo Silva u201809213                       #
5  %Fabio Morais u201504257                      #
6  %                                             #
7  %##############################################
8
9  %% Limpar consola
10 close all;
11 clear all;
12 clc;
13
14 %% Ler dados
15
```

```
16 Input = readmatrix("Dados\P2\testInput11A.txt");
17 Output = readmatrix("Dados\P2\testOutput11A.txt");
18
19 %% Escolha de variaveis padrao
20 indeterminados=0;
21 erradas=0;
22 corretas=0;
23 K=7;
24
25 %% fun oes auxiliares
26 normalizar_out = @(y) (y+1)/2;
27
28 %% Calcular tamanho dos vetores de dados
29
30 tamanho_data_treino = find(ismember(Input,[0 0 0],'rows'))-1; %
    Encontrar [0 0 0] para calcular o tamanho dos dados de
    treino
31 tamanho_data_teste = length(Input)-tamanho_data_treino-1; %
    Tamanho dos dados de teste
32
33 %% Criar vetores de dados de input
34
35 Dados_Treino = Input(1:tamanho_data_treino,:);
36 Dados_Teste = Input(tamanho_data_treino+2:tamanho_data_treino+
    tamanho_data_teste+1,1:2);
37
38 %% Criar e normalizar dados de output
39 Dados_Treino_Output_norm = normalizar_out(Dados_Treino(:,3));
40 Output= normalizar_out( Output);
41
42 %% Algoritmo
43
44 for i=1:tamanho_data_teste % Para cada ponto em que nao
    conhe a o seu valor de output vai tentar descobir o seu
    valor atravez dos seu vizinhos
45
46     distancia = zeros(1,tamanho_data_treino);%vetor que contem
       as distancias entre os pontos conhecidos e o ponto
       desconhecido
47
48     % 1 calcular distancias
49     for j=1:tamanho_data_treino % Hip^2=cat1^2+cat2^2
50         distancia(1,j) = sqrt((Dados_Teste(i,1)- Dados_Treino(j
              ,1))^2 +(Dados_Teste(i,2)- Dados_Treino(j,2))^2);
51     end
52
53     menorDist = zeros(1,K); %guarda as menores distancias
54     indexMinDist = zeros(1,K);  %guarda os index das distancias
```

```
55      for j=1:K % seleciona as K menores distancias
56          [menorDist(1,j),indexMinDist(1,j)] = min(distancia);%
                Vai busacar a menor distancia e devolve o numero e a
                 posicao no vetor
57          distancia(1,indexMinDist(1,j)) = 10000000000;%Para nao
                voltar a ir buscar a mesma distancia metemos essa
                como "infinita"
58      end
59
60      media = mean(menorDist); %Calcular a media das menores
            distancias para calcualar o sigma
61      sumatorio = 0; %Para calcular a primeira parte do sigma (
            formula 6 do relatorio)
62
63      % 3 Calcular Sigma
64      for j=1:K
65          sumatorio= sumatorio + (menorDist(1,j)  - media)^2; %
                Sumatorio (d_i - d_medio)^2
66      end
67      sigma=sqrt( sumatorio/K); %sigma
68
69      Di=[];%Numero difuso distancia crescente
70      D0=[];%Numero difuso com [sigma, 0]    o denominador da
            formula (5)
71      for j=1:K %criar numerador e denuminador
72          Di=[Di;gaussmf(0:0.0001:sqrt(3),[sigma menorDist(1,j)])
                ]; %numerador eq (5)
73          D0=[D0;gaussmf(0:0.0001:sqrt(3),[sigma 0])];%
                denominador da formula (5)
74      end
75      Di=Di./(D0+1e-100);%calculo final do numero difuso eq(5)
            adicionamos um numero muito pequeno para nao dar erro a
            dividir por zero
76
77      A = zeros(K,2);% Matriz A coluna 1 contem a classe "Nao
            soldado ->0" a coluna dois tem "Soldado ->1"
78
79      for j=1:K
80          %Fuzificar matriz A
81          if(Dados_Treino_Output_norm(indexMinDist(1,j))==1) %se
                tiver soldado(=1) entao adiciona na segundo coluna
82              A(j,2)=Di(j,1);
83          else
84              A(j,1)=Di(j,1);% Senao adiciona na primeira
85          end
86      end
87      W = ones(1,K);
88      R=eye(K);%Matriz identidade
```

```
89      resultado = W * R *A; %equa ao (12) Agora     a matriz A
            fuzificada e nao a R
90      [Valor,index]= max(resultado);%equa ao (12)
91      if resultado(1,1)==resultado(1,2)%Se as duas colunas
            tiverem 1 o algoritmo escolheu as duas classes como
            valor final logo uma indetermina ao
92          index=0;
93          indeterminados=indeterminados+1;
94      elseif(Output(i)==index-1)
95          corretas=corretas+1;
96      else
97          erradas = erradas+1;
98      end
99  end
100 corretas
101 erradas
102 indeterminados
```

### 6.e Fuzzy KNN Dados Artigo

```
1  %##############################################
2  %                   FUZZY KNN                  #
3  %Desenvolvido por :                           #
4  %Diogo Silva u201809213                       #
5  %Fabio Morais u201504257                      #
6  %                                             #
7  %##############################################
8
9  %% Limpar consola
10 close all;
11 clear all;
12 clc;
13
14 %% Ler dados
15 TesteInput = readmatrix("Dados\Artigo\testInput.txt");
16 TesteOutput = readmatrix("Dados\Artigo\testOutput.txt");
17 Treino= readmatrix("Dados\Artigo\treino.txt");
18
19 %% Escolha de variaveis padrao
20 indeterminados=0;
21 erradas=0;
22 corretas=0;
23 K=3;
24
25 %% Calcular tamanho dos vetores de dados
26 tamanho_data_treino = length(Treino); %Tamanho dos dados de
       treino
27 tamanho_data_teste = length(TesteInput); %Tamanho dos dados de
       teste
```

```
28
29 %% Criar dados de output
30 Dados_Treino_Output_norm = Treino(:,4);
31
32
33 %% Algoritmo
34
35 for i=1:tamanho_data_teste % Para cada ponto em que nao
       conhe a o seu valor de output vai tentar descobir o seu
       valor atravez dos seu vizinhos
36
37     distancia = zeros(1,tamanho_data_treino);%vetor que contem
          as distancias entre os pontos conhecidos e o ponto
          desconhecido
38
39     % 1 calcular distancias
40     for j=1:tamanho_data_treino % Hip^2=cat1^2+cat2^2
41         distancia(1,j) = sqrt((TesteInput(i,1)- Treino(j,1))^2
              +(TesteInput(i,2)- Treino(j,2))^2+(TesteInput(i,3)-
              Treino(j,3))^2);
42     end
43
44     menorDist = zeros(1,K); %guarda as menores distancias
45     indexMinDist = zeros(1,K);  %guarda os index das distancias
46     for j=1:K % seleciona as K menores distancias
47         [menorDist(1,j),indexMinDist(1,j)] = min(distancia);%
              Vai busacar a menor distancia e devolve o numero e a
               posicao no vetor
48         distancia(1,indexMinDist(1,j)) = 10000000000;%Para nao
              voltar a ir buscar a mesma distancia metemos essa
              como "infinita"
49     end
50
51     media = mean(menorDist); %Calcular a media das menores
          distancias para calcualar o sigma
52     sumatorio = 0; %Para calcular a primeira parte do sigma (
          formula 6 do relatorio)
53
54     % 3 Calcular Sigma
55     for j=1:K
56         sumatorio= sumatorio + (menorDist(1,j)  - media)^2; %
              Sumatorio (d_i - d_medio)^2
57     end
58     sigma=sqrt( sumatorio/K); %sigma
59
60     Di=[];%Numero difuso distancia crescente
61     D0=[];%Numero difuso com [sigma, 0]    o denominador da
          formula (5)
```

```
62      for j=1:K %criar numerador e denuminador
63          Di=[Di;gaussmf(0:0.0001:sqrt(3),[sigma menorDist(1,j)])
                ]; %numerador eq (5)
64          D0=[D0;gaussmf(0:0.0001:sqrt(3),[sigma 0])];%
                denominador da formula (5)
65      end
66      Di=Di./(D0+1e-100);%calculo final do numero difuso eq(5)
            adicionamos um numero muito pequeno para nao dar erro a
            dividir por zero
67
68      A = zeros(K,2);% Matriz A coluna 1 contem a classe "Nao
            soldado ->0" a coluna dois tem "Soldado ->1"
69
70      for j=1:K
71          %Fuzificar matriz A
72          if(Dados_Treino_Output_norm(indexMinDist(1,j))==1) %se
                tiver soldado(=1) entao adiciona na segundo coluna
73              A(j,2)=Di(j,1);% Vai buscar os valores    primeira
                    coluna da matriz Di
74          else
75              A(j,1)=Di(j,1);% Senao adiciona na primeira
76          end
77      end
78      W = ones(1,K);
79      R=eye(K);%Matriz identidade
80      resultado = W * R *A; %equa ao (12) Agora    a matriz A
            fuzificada e nao a R
81      [Valor,index]= max(resultado);%equa ao (12)
82      if resultado(1,1)==resultado(1,2)%Se as duas colunas
            tiverem 1 o algoritmo escolheu as duas classes como
            valor final logo uma indetermina ao
83          index=0;
84          indeterminados=indeterminados+1;
85      elseif(TesteOutput(i)==index-1)
86          corretas=corretas+1;
87      else
88          erradas = erradas+1;
89      end
90  end
91  corretas
92  erradas
93  indeterminados
```

### 6.f Neuro Fuzzy Dados TP7

```
1  %##############################################
2  %             Sistema Neuro-Difuso             #
3  %Desenvolvido por :                            #
4  %Diogo Silva u201809213                        #
```

```
5 %Fabio Morais u201504257                                    #
6 %                                                          #
7 %############################################################
8
9 %% Limpar consola
10 close all;
11 clear all;
12 clc;
13
14 %% Ler dados
15 Input = readmatrix("Dados\P2\testInput11C.txt");
16 Output = readmatrix("Dados\P2\testOutput11C.txt");
17
18 %% Funcao auxiliar
19 Classificar =  @(y) (y>0)*2-1;%% Classificar a saida
20
21 %% Calcular tamanho dos vetores de dados
22
23 tamanho_data_treino = find(ismember(Input,[0 0 0],'rows'))-1; %
    Encontrar [0 0 0] para calcular o tamanho dos dados de
    treino
24 tamanho_data_teste = length(Input)-tamanho_data_treino-1; %
    Tamanho dos dados de teste
25 %% Criar vetores de dados de input
26 Dados_Treino = Input(1:tamanho_data_treino,:);
27 Dados_Teste = Input(tamanho_data_treino+2:tamanho_data_treino+
    tamanho_data_teste+1,1:2);
28 %% Valores para o anfis
29 iteracoes = 1000;% N mero de iteracoes maximas no treino
30 taxa_de_aprendizagem = 0.01; % taxa de aprendizagem
31 Nmemberships = 3;% N mero de memberships por input
32
33 %% Normalizar entrada
34 Dados_Treino(:,1) = ((Dados_Treino(:,1) - min( Dados_Treino
    (:,1)))/(max(Dados_Treino(:,1))-min( Dados_Treino(:,1))))
    *2-1;
35 Dados_Treino(:,2) = ((Dados_Treino(:,2) - min( Dados_Treino
    (:,2)))/(max(Dados_Treino(:,2))-min( Dados_Treino(:,2))))
    *2-1;
36
37 Dados_Teste(:,1) = ((Dados_Teste(:,1) - min( Dados_Teste(:,1)))
    /(max(Dados_Teste(:,1))-min( Dados_Teste(:,1))))*2-1;
38 Dados_Teste(:,2) = ((Dados_Teste(:,2) - min( Dados_Teste(:,2)))
    /(max(Dados_Teste(:,2))-min( Dados_Teste(:,2))))*2-1;
39 %% DEFINI  O DA ANFIS
40 [fis,error,stepsize] = anfis([Dados_Treino(:,1:2) Dados_Treino
    (:,3)],Nmemberships,[iteracoes 0 taxa_de_aprendizagem 0.9
    1.1],[1 0 0 1]);
```

```
41 %Anfis vai ser usada para devolver a FIS (Fuzzy inference
       system) isto sera
42 %usado para calcular as saidas dos inputs inseridos, para isso
       iremos usar
43 %a evalfis que vai avaliar a fis a uma dada entrada e retorna a
        saida esperada
44
45 %% dados de treino
46 Output_Rede_Treino = evalfis(fis,Dados_Treino(:,1:2));
47 Output_Rede_Treino_norm = Classificar(Output_Rede_Treino);
48 figure(1)
49 hold on
50 scatter(1:tamanho_data_treino,Dados_Treino(:,3))
51 scatter(1:tamanho_data_treino, Output_Rede_Treino_norm, '.')
52 grid on
53 title('Dados de treino')
54 xlabel('Indice de input')
55 ylabel('Output')
56
57 num_erros_treino = sum( Output_Rede_Treino_norm ~= Dados_Treino
       (:,3) );
58
59
60 %% Calcular outputs finais do teste usando a evalfis
61 Output_rede_teste      = evalfis(fis,Dados_Teste);
62 Output_rede_teste_norm = Classificar(Output_rede_teste);
63
64 %% Visualizar os dados de teste
65   figure(2)
66   hold on
67   scatter(1:tamanho_data_teste,  Output)
68   scatter(1:tamanho_data_teste, Output_rede_teste_norm, '.')
69   grid on
70   title('Dados de teste')
71   xlabel('Indice de input')
72   ylabel('Output')
73   num_erros_teste = sum( Output_rede_teste_norm ~=Output) ;
74
75 %% erros
76 num_erros_treino
77 num_erros_teste
```

### 6.g Neuro Fuzzy Dados Artigo

```
1 %##############################################
2 %            Sistema Neuro-Difuso             #
3 %Desenvolvido por :                           #
4 %Diogo Silva u201809213                       #
5 %Fabio Morais u201504257                      #
```

```
6 %                                                    #
7 %#################################################
8
9 %% Limpar consola
10 close all;
11 clear all;
12 clc;
13
14 %% Ler dados
15 TesteInput = readmatrix("Dados\Artigo\testInput.txt");
16 TesteOutput = readmatrix("Dados\Artigo\testOutput.txt");
17 Treino= readmatrix("Dados\Artigo\treino.txt");
18
19
20 %% Funcao auxiliar
21
22 Classificar =  @(y) (y>0.5);
23
24 %% Calcular tamanho dos vetores de dados
25 tamanho_data_treino = length(Treino); %Tamanho dos dados de
       treino
26 tamanho_data_teste = length(TesteInput); %Tamanho dos dados de
       teste
27
28
29 %% Normalizar entradas
30 TesteInput(:,1) = ((TesteInput(:,1) - min( TesteInput(:,1)))/(
       max(TesteInput(:,1))-min( TesteInput(:,1))))*2-1;
31 TesteInput(:,2) = ((TesteInput(:,2) - min( TesteInput(:,2)))/(
       max(TesteInput(:,2))-min( TesteInput(:,2))))*2-1;
32 TesteInput(:,3) = ((TesteInput(:,3) - min( TesteInput(:,3)))/(
       max(TesteInput(:,3))-min( TesteInput(:,3))))*2-1;
33
34 Treino(:,1) = ((Treino(:,1) - min( Treino(:,1)))/(max(Treino
       (:,1))-min( Treino(:,1))))*2-1;
35 Treino(:,2) = ((Treino(:,2) - min( Treino(:,2)))/(max(Treino
       (:,2))-min( Treino(:,2))))*2-1;
36 Treino(:,3) = ((Treino(:,3) - min( Treino(:,3)))/(max(Treino
       (:,3))-min( Treino(:,3))))*2-1;
37 %% Valores para o anfis
38 iteracoes = 10000;% N mero de iteracoes maximas no treino
39 taxa_de_aprendizagem = 0.008; % taxa de aprendizagem
40 Nmemberships = 3;% N mero de memberships por input
41
42 %% DEFINI  O DA ANFIS
43 [fis,error,stepsize] = anfis([Treino(:,1:3) Treino(:,4)],
       Nmemberships,[iteracoes 0 taxa_de_aprendizagem 0.9 1.1],[1 0
        0 1]);
```

```
%Anfis vai ser usada para devolver a FIS (Fuzzy inference system) isto sera
%usado para calcular as saidas dos inputs inseridos, para isso iremos usar
%a evalfis que vai avaliar a fis a uma dada entrada e retorna a saida esperada
%%
Output_Rede_Treino = evalfis(fis,Treino(:,1:3));
Output_Rede_Treino_norm = Classificar(Output_Rede_Treino);
figure(1)
hold on
scatter(1:tamanho_data_treino,  Treino(:,4))
scatter(1:tamanho_data_treino, Output_Rede_Treino_norm, '.')
grid on
title('Dados de treino')
xlabel('Indice de input')
ylabel('Output')

num_erros_treino = sum( Output_Rede_Treino_norm ~= Treino(:,4));

%% Calcular outputs finais do teste usando a evalfis
Output_rede_teste      = evalfis(TesteInput, fis);
Output_rede_teste_norm = Classificar(Output_rede_teste);

%% Visualizar os dados de teste
figure(2)
hold on
scatter(1:tamanho_data_teste,  TesteOutput)
scatter(1:tamanho_data_teste, Output_rede_teste_norm, '.')
grid on
title('Dados de teste')
xlabel('Indice de input')
ylabel('Output')
num_erros_teste = sum( Output_rede_teste_norm ~=TesteOutput) ;
%% erros
num_erros_treino
num_erros_teste
```

### 6.h Rede Neuronal Dados TP7

```
close all
clear all
clc
index_funcActivation=1; % escolhe fun ao de ativa ao
%%
switch index_funcActivation
  case 1  % FUN  O DE ATIVA  O TANGENTE HIPERB LICA
    inMin = -1;
```

```
9       inMax = +1;
10      normaliza = @(x,min,max) ((x-min)/(max-min)*2-1);
11      normalizaOut    = @(z) z;
12      activationFunction = @(x) tanh(x);
13      activationFunction_derivative = @(x) (sech(x)).^2;
14      normalizaSaida = @(z) (z > 0)*2-1; % entre -1 e 1
15    case 2  % FUN  O DE ATIVA  O SIGMOID
16      inMin =  0;
17      inMax = +1;
18      normaliza = @(x,min,max) ((x-min)/(max-min));
19      normalizaOut    = @(z) (z>0);
20      activationFunction = @(x) 1./(1+exp(-x));
21      activationFunction_derivative = @(x) exp(-x)./(exp(-x) + 1)
           .^2;
22      normalizaSaida = @(z) (z > 0.5); % entre 0 e 1
23  end
24
25  %%
26
27  inputFile = 'Dados/P2/testInput11B.txt';
28  outputFile = 'Dados/P2/testOutput11B.txt';
29  inputData = readmatrix(inputFile); % Leitura input
30  outputData = readmatrix(outputFile); % Leitura output
31
32  trainSize = find(ismember( inputData, [0 0 0], 'rows' )) - 1; %
       tamanho do treino
33  testSize = length(inputData) - find(ismember( inputData, [0 0
       0], 'rows' )) ; %tamanho do teste
34
35  if length(outputData) ~= testSize
36      disp('tamanho diferente');
37      return;
38  end
39
40  trainData =  inputData( 1:trainSize , : );
41  testData =  inputData(  trainSize + 2 : trainSize + testSize +
       1 , 1:2);
42
43
44  %% normaliza dados
45
46  trainDataNormalizado(:,1) = normaliza( trainData(:,1), min(
       trainData(:,1)), max(trainData(:,1)) ); % Normaliza  o dos
        dados de entrada para treino da rede
47  trainDataNormalizado(:,2) = normaliza( trainData(:,2), min(
       trainData(:,2)), max(trainData(:,2)) ); % Normaliza  o dos
        dados de entrada para treino da rede
48  trainDataNormalizado(:,3) = normalizaOut(trainData(:,3));
```

```
49
50 testeDataNormalizado(:,1) = normaliza( testData(:,1), min(
       trainData(:,1)), max(trainData(:,1)) ); % Normaliza  o dos
       dados de entrada para treino da rede
51 testeDataNormalizado(:,2) = normaliza( testData(:,2), min(
       trainData(:,2)), max(trainData(:,2)) ); % Normaliza  o dos
       dados de entrada para treino da rede
52
53
54 %% fun oes
55 random = @(m,n) rand(m,n)/2;              % gera os pesos entre 0
       a 0.5
56
57 sumNeuro = @(x, w, b) sum(x * w) + b;
58 saidaNeuro = @(x, w, b) activationFunction(sumNeuro(x,w,b));
59
60 %% Treinar
61 %exemplo de configura ao 5:2
62 %         o \
63 %x1---/ o-o\
64 %         o   o--
65 %x2---\ o-o/
66 %         o/
67
68 % Configurar a rede
69 nosCamada1=5; % nos da camada 1
70 nosCamada2=3; % nos da camada 2
71
72 w0 = random(nosCamada1,2);
73 w1 = random(nosCamada1,nosCamada2);
74 w2 = random(1,nosCamada2);
75
76 b1 = random(1,nosCamada1); % para o H1, H2 e Z
77 b2 = random(1,nosCamada2);
78 b3 = random(1,1);
79 eh1= zeros(nosCamada1,1);
80 eh2= zeros(nosCamada2,1);
81 ez= zeros(1,1);
82
83 erros = 1;
84 maxNumIteracao = 1000;
85 numIteracao = 0;
86 taxaAprendizagem = 0.2;
87 z = zeros(1,trainSize);
88
89 h1=zeros(nosCamada1,1); % hidden layer com 4 n s
90 h2=zeros(nosCamada2,1); % hidden layer com 2 n s
91
```

```
92 while erros > 0 && numIteracao < maxNumIteracao
93     for i=1 : trainSize
94         h1 = sum((w0.*trainDataNormalizado(i,1:2)).') + b1;
95         h2 = activationFunction(h1) * w1 + b2;
96         uk = sum(w2 .* activationFunction(h2)) + b3;
97         z(i) = activationFunction(uk);
98 
99         % Erros
100        ez= (trainDataNormalizado(i,3) - z(i)) *
               activationFunction_derivative(uk); % diferen a
               entre o desejado e saida
101        eh2=((w2*ez.*activationFunction_derivative(h2)).' );
102        eh1=(w1*eh2.*activationFunction_derivative(h1.'));
103 
104        w0= w0+ eh1*trainDataNormalizado(i,1:2)*
               taxaAprendizagem;
105        w1 = w1 + (eh2.*activationFunction(h1)*taxaAprendizagem
               ).';
106        w2 = w2 + ez*activationFunction(h2)*taxaAprendizagem ;
107 
108        b1 = b1 + (eh1*taxaAprendizagem).'; % pertence ao h1
109        b2 = b2 + (eh2*taxaAprendizagem).'; % pertence ao h2
110        b3 = b3 + (ez*taxaAprendizagem).'; % pertence ao z
111 
112 
113    end
114    saidaZ = normalizaSaida(z);
115    erros=0;
116    for i=1 : trainSize
117        if saidaZ(i) ~= trainDataNormalizado(i,3)
118            erros= erros + 1;
119        end
120    end
121    if mod(numIteracao, 100) == 0
122        fprintf('[%d] - erros %d\n', numIteracao, erros);
123    end
124 
125 numIteracao = numIteracao + 1;
126 end
127 
128 
129 %% testar com os valores que foram dados de teste
130 
131 erros=0;
132 for i=1 : testSize
133     h1 = sum((w0.*testeDataNormalizado(i,1:2)).') + b1;
134     h2 = activationFunction(h1) * w1 + b2;
135     uk = sum(w2 .* activationFunction(h2)) + b3;
```

```
136         z(i) = activationFunction(uk);
137         saidaTest = normalizaSaida(z(i));
138         saidaReal = normalizaOut(outputData(i));
139         if saidaTest ~= saidaReal
140             erros= erros+1;
141             fprintf('[%i]-Saida teste: %d saida real: %d \n',i,
                    saidaTest, saidaReal);
142
143         end
144     end
145     fprintf('[%s]  numero de itera ao= %d\t numero de erros: %d\t
        n   amostras: %d\t taxa de sucesso: %.1f%% \n',inputFile,
        numIteracao,erros, testSize,(testSize-erros)/testSize *100);
146
147     %%
148     red  = [];  % Classe -1
149     blue = [];  % Classe +1
150
151     % Pontos relativos ao treino da rede
152     for i=1:trainSize
153       if trainData(i,3) == -1
154         red  = [ red  ; trainDataNormalizado(i, 1:2) ];  %
              Adicionar valores da entrada quando o exemplo    da
              classe RED
155       else
156         blue = [ blue ; trainDataNormalizado(i, 1:2) ];  %
              Adicionar valores da entrada quando o exemplo    da
              classe BLUE
157       end
158     end
159     for i=1:testSize
160       if outputData(i,1) == -1
161         red  = [ red  ; testeDataNormalizado(i, :) ];     %
              Adicionar valores da entrada quando os dados de teste
              s o da classe RED
162       else
163         blue = [ blue ; testeDataNormalizado(i, :) ];     %
              Adicionar valores da entrada quando os dados de teste
              s o da classe BLUE
164       end
165     end
166
167     x = inMin:0.001:inMax;
168     figure(1)
169     hold on
170     xlim([ inMin inMax ])
171     ylim([ inMin inMax ])
172     scatter( red(:,1) , red(:,2) , 25, 'red' , 'filled')% Pontos
```

```
    pertencentes     classe -1
scatter( blue(:,1), blue(:,2), 25, 'blue', 'filled')% Pontos
    pertencentes     classe +1
scatter( testeDataNormalizado(:,1), testeDataNormalizado(:,2),
    60, 'black') % Pontos de teste da rede neuronal

%plot(x, (-b(1,1)-w(1,1)*x)/w(1,2), 'LineWidth', 1, 'Color', [0
    0 0]) % Linha de decis o das classes
legend('Classe -1', 'Classe +1', 'Teste')
xlabel('Entrada X \rightarrow')
ylabel('Entrada Y \rightarrow')
grid on
```

### 6.i Rede Neuronal Dados Artigo

```
close all
clear all
clc
index_funcActivation=1; % escolhe fun ao de ativa ao
%%
switch index_funcActivation
  case 1  % FUN  O DE ATIVA  O TANGENTE HIPERB LICA
    inMin = -1;
    inMax = +1;
    normaliza = @(x,min,max) ((x-min)/(max-min)*2-1);
    normalizaOut    = @(z) (z > 0)*2-1;
    activationFunction = @(x) tanh(x);
    activationFunction_derivative = @(x) (sech(x)).^2;
    normalizaSaida = @(z) (z > 0)*2-1; % entre -1 e 1
  case 2  % FUN  O DE ATIVA  O SIGMOID
    inMin =  0;
    inMax = +1;
    normaliza = @(x,min,max) ((x-min)/(max-min));
    normalizaOut    = @(z) (z>0);
    activationFunction = @(x) 1./(1+exp(-x));
    activationFunction_derivative = @(x) exp(-x)./(exp(-x) + 1)
        .^2;
    normalizaSaida = @(z) (z > 0.5); % entre 0 e 1
end

%%

inputFile = 'Dados/artigo/testInput.txt';
outputFile = 'Dados/artigo/testOutput.txt';
trainFile = 'Dados/artigo/treino.txt';

testData = readmatrix(inputFile); % Leitura input
outputData = readmatrix(outputFile); % Leitura output
trainData = readmatrix(trainFile);
```

```
34
35 trainSize =length(trainData); % tamanho do treino
36 testSize = length(testData); %tamanho do teste
37
38 if length(outputData) ~= testSize
39     disp('tamanho diferente');
40     return;
41 end
42
43
44
45 % normaliza dados
46
47 trainDataNormalizado(:,1) = normaliza( trainData(:,1), min(
       trainData(:,1)), max(trainData(:,1)) ); % Normaliza o dos
       dados de entrada para treino da rede
48 trainDataNormalizado(:,2) = normaliza( trainData(:,2), min(
       trainData(:,2)), max(trainData(:,2)) ); % Normaliza o dos
       dados de entrada para treino da rede
49 trainDataNormalizado(:,3) = normaliza( trainData(:,3), min(
       trainData(:,3)), max(trainData(:,3)) ); % Normaliza o dos
       dados de entrada para treino da rede
50 trainDataNormalizado(:,4) = normalizaOut(trainData(:,4));
51
52 testeDataNormalizado(:,1) = normaliza( testData(:,1), min(
       trainData(:,1)), max(trainData(:,1)) ); % Normaliza o dos
       dados de entrada para treino da rede
53 testeDataNormalizado(:,2) = normaliza( testData(:,2), min(
       trainData(:,2)), max(trainData(:,2)) ); % Normaliza o dos
       dados de entrada para treino da rede
54 testeDataNormalizado(:,3) = normaliza( testData(:,3), min(
       trainData(:,3)), max(trainData(:,3)) ); % Normaliza o dos
       dados de entrada para treino da rede
55
56
57 %% fun oes
58 random = @(m,n) rand(m,n)/2;              % gera os pesos entre 0
       a 0.5
59
60 sumNeuro = @(x, w, b) sum(x * w) + b;
61 saidaNeuro = @(x, w, b) activationFunction(sumNeuro(x,w,b));
62
63 %% Treinar
64 %exemplo de configura ao 5:3
65 %         o \
66 %x1---/ o-o\
67 %         o-o-o--
68 %x2---\ o-o/
```

```
69 %         o/
70
71 % Configurar a rede
72 nosCamada1=5; % nos da camada 1
73 nosCamada2=3; % nos da camada 2
74
75 w0 = random(nosCamada1,3);
76 w1 = random(nosCamada1,nosCamada2);
77 w2 = random(1,nosCamada2);
78
79 b1 = random(1,nosCamada1); % para o H1, H2 e Z
80 b2 = random(1,nosCamada2);
81 b3 = random(1,1);
82 eh1= zeros(nosCamada1,1);
83 eh2= zeros(nosCamada2,1);
84 ez= zeros(1,1);
85
86 erros = 1;
87 maxNumIteracao = 1000;
88 numIteracao = 0;
89 taxaAprendizagem = 0.2;
90 z = zeros(1,trainSize);
91
92 h1=zeros(nosCamada1,1); % hidden layer com 4 n s
93 h2=zeros(nosCamada2,1); % hidden layer com 2 n s
94
95 while erros > 0 && numIteracao < maxNumIteracao
96     for i=1 : trainSize
97         h1 = sum((w0.*trainDataNormalizado(i,1:3)).') + b1;
98         h2 = activationFunction(h1) * w1 + b2;
99         uk =  sum(w2 .* activationFunction(h2)) + b3;
100        z(i) = activationFunction(uk);
101
102        % Erros
103        ez= (trainDataNormalizado(i,4) - z(i)) *
               activationFunction_derivative(uk); % diferen a
               entre o desejado e saida
104        eh2=((w2*ez.*activationFunction_derivative(h2)).' );
105        eh1=(w1*eh2.*activationFunction_derivative(h1.'));
106
107        w0= w0+ eh1*trainDataNormalizado(i,1:3)*
               taxaAprendizagem;
108        w1 = w1 + (eh2.*activationFunction(h1)*taxaAprendizagem
               ).';
109        w2 = w2 + ez*activationFunction(h2)*taxaAprendizagem ;
110
111        b1 = b1 +  (eh1*taxaAprendizagem).'; % pertence ao h1
112        b2 = b2 +  (eh2*taxaAprendizagem).'; % pertence ao h2
```

```
113         b3 = b3 +  (ez*taxaAprendizagem).'; % pertence ao z
114
115     end
116     saidaZ = normalizaSaida(z);
117     erros=0;
118     for i=1 : trainSize
119         if saidaZ(i) ~= trainDataNormalizado(i,4)
120             erros= erros + 1;
121         end
122     end
123     if mod(numIteracao, 100) == 0
124         fprintf('[%d] - erros %d\n', numIteracao, erros);
125     end
126
127 numIteracao = numIteracao + 1;
128 end
129
130
131 %% testar com os valores que foram dados de teste
132 fprintf("%d-sucesso treino : %f\n", erros, ((trainSize-erros)/
    trainSize) * 100);
133 erros=0;
134 for i=1 : testSize
135      h1 = sum((w0.*testeDataNormalizado(i,1:3)).')  + b1;
136      h2 = activationFunction(h1) * w1 + b2;
137      uk =  sum(w2 .* activationFunction(h2)) + b3;
138      z(i) = activationFunction(uk);
139     saidaTest = normalizaSaida(z(i));
140     saidaReal = normalizaOut(outputData(i));
141     if saidaTest ~= saidaReal
142         erros= erros+1;
143         fprintf('[%i]-Saida teste: %d saida real: %d\t%f\t%f \n
            ',i, saidaTest, saidaReal, z(i), outputData(i));
144
145     end
146 end
147 fprintf('[%s]  numero de itera ao= %d\t numero de erros: %d\t
    n  amostras: %d\t taxa de sucesso: %.1f%% \n',inputFile,
    numIteracao,erros, testSize,(testSize-erros)/testSize *100);
148
149 %
150 red  = [];  % Classe -1
151 blue = [];  % Classe +1
152
153 % Pontos relativos ao treino da rede
154 for i=1:trainSize
155   if trainData(i,4) == 0
156     red  = [ red  ; trainDataNormalizado(i, 1:3) ];  %
```

```
        Adicionar valores da entrada quando o exemplo    da
        classe RED
157   else
158     blue = [ blue ; trainDataNormalizado(i, 1:3) ];  %
        Adicionar valores da entrada quando o exemplo    da
        classe BLUE
159   end
160 end
161 for i=1:testSize
162   if outputData(i,1) == 0
163     red  = [ red  ; testeDataNormalizado(i, :) ];     %
        Adicionar valores da entrada quando os dados de teste
        s o da classe RED
164   else
165     blue = [ blue ; testeDataNormalizado(i, :) ];     %
        Adicionar valores da entrada quando os dados de teste
        s o da classe BLUE
166   end
167 end
168
169 figure(1)
170 hold on
171 xlim([ inMin inMax ])
172 ylim([ inMin inMax ])
173 zlim([ inMin inMax ])
174 scatter3( red(:,1) , red(:,2) ,red(:,3), 25, 'red' , 'filled')%
    Pontos pertencentes    classe -1
175 scatter3( blue(:,1), blue(:,2),blue(:,3), 25, 'blue', 'filled')
    % Pontos pertencentes    classe +1
176 scatter3( testeDataNormalizado(:,1), testeDataNormalizado(:,2),
    testeDataNormalizado(:,3), 60, 'black') % Pontos de teste
    da rede neuronal
177
178 %plot(x, (-b(1,1)-w(1,1)*x)/w(1,2), 'LineWidth', 1, 'Color', [0
    0 0]) % Linha de decis o das classes
179 legend('NonWeld', 'Weld', 'Teste')
180 xlabel('Entrada U_i')
181 ylabel('Entrada V_i')
182 zlabel('Entrada W_i')
183 grid on
184 view(3)
```