



**PTC3101 – Engenho e Arte do controle automático**  
**Implementação e comparação de classificadores**  
**08/12/2020**

Enrico Antonio Jehá Molento de Moraes

9838219

José Luiz Carvalho de Sousa

9017273

Victor Angelo Nunes Notário

9836822

Professor Felipe Pait

## Introdução

O trabalho da disciplina consiste na implementação de dois classificadores supervisionados em MatLab, sendo um linear e o outro não-linear. Tais classificadores utilizam redes neurais que tomam decisões a partir das entradas fornecidas. Para isso, arquivos de entrada são utilizados para treinar e testar a rede após o treinamento, respectivamente. Primeiramente, testa-se os classificadores com 4 entradas, e depois, testa-se com 9 entradas, visando dificultar o aprendizado e avaliando o desempenho dos classificadores para entradas mais densas e complexas.

## Dados de entrada

Trata-se de dois arquivos no formato “txt”, um para treinar a rede e outro para testá-la. As 3 primeiras colunas indicam valores de entrada do sistema, enquanto a última coluna representa a classe que a entrada pertence, sendo um valor inteiro de 1 a 3.

treina.txt - Bloco de Notas				
Arquivo	Editar	Formatar	Exibir	Ajuda
6.3	2.8	5.1	1.5	3
6.1	2.6	5.6	1.4	3
5.4	3.4	1.7	0.2	1
5.6	2.9	3.6	1.3	2
5.8	2.7	4.1	1.0	2
5.1	3.7	1.5	0.4	1
5.0	3.0	1.6	0.2	1
6.3	2.7	4.9	1.8	3
6.7	3.3	5.7	2.1	3
5.1	3.8	1.5	0.3	1
7.2	3.2	6.0	1.8	3
6.2	2.8	4.8	1.8	3
5.0	3.4	1.6	0.4	1
5.2	3.5	1.5	0.2	1
6.5	3.0	5.5	1.8	3
7.7	3.8	6.7	2.2	3
5.5	3.5	1.3	0.2	1
4.9	3.1	1.5	0.1	1
6.6	2.9	4.6	1.3	2
5.2	2.7	3.9	1.4	2
5.0	2.0	3.5	1.0	2
4.6	3.6	1.0	0.2	1
5.1	3.3	1.7	0.5	1
4.8	3.4	1.9	0.2	1
5.9	3.0	4.2	1.5	2
6.0	2.2	4.0	1.0	2

teste.txt - Bloco de Notas				
Arquivo	Editar	Formatar	Exibir	Ajuda
4.6	3.4	1.4	0.3	1
5.5	2.5	4.0	1.3	2
5.5	2.6	4.4	1.2	2
5.0	3.4	1.5	0.2	1
4.4	2.9	1.4	0.2	1
5.7	2.8	4.1	1.3	2
7.6	3.0	6.6	2.1	3
4.9	2.5	4.5	1.7	3
7.3	2.9	6.3	1.8	3
4.8	3.0	1.4	0.1	1
4.5	2.3	1.3	0.3	1
6.3	2.3	4.4	1.3	2
6.1	3.0	4.6	1.4	2
5.0	3.6	1.4	0.2	1
5.4	3.9	1.7	0.4	1
5.8	2.6	4.0	1.2	2
5.0	2.3	3.3	1.0	2
5.6	2.7	4.2	1.3	2
6.4	3.1	5.5	1.8	3
5.6	3.0	4.1	1.3	2
6.0	3.0	4.8	1.8	3
6.9	3.1	5.4	2.1	3
5.7	3.0	4.2	1.2	2
5.7	2.9	4.2	1.3	2
6.7	2.5	5.8	1.8	3

Também, há outros 2 arquivos de entrada no mesmo formato, “teste1” e “treino1”, os quais contém 9 dados de entrada, com a finalidade de avaliar os classificadores com entradas mais numerosas e confusas.

## Classificador Não-Linear

Nesta etapa do projeto, foi proposta a construção de um Perceptron multicamada, com algumas especificações:

- Deve ser possível alterar o número de neurônios nas camadas
- Possuir uma camada escondida
- Utilização do algoritmo de retropropagação, com as entradas normalizadas para média nula e variância unitária (saídas entre -1 e 1), além da inclusão de um termo de *momentum*, caso necessário.

Com o algoritmo construído, deve-se treinar a rede com os dados fornecidos nos arquivos textos “treina.txt” e “treina1.txt”. Por fim, testa-se a rede com entradas novas, dadas pelo “teste.txt” e “teste1.txt”, retirando-se a última coluna, referente às classes alvo de cada sequência de entrada.

Primeiramente, para normalizar as entradas, foi criada uma função extra, chamada “normaliza”, que retorna as entradas no intervalo -1 e 1. Além disso, foi usada a função de ativação da tangente hiperbólica, dado que a sigmóide usual varia apenas de 0 a 1.

```
function X_normalizado = normaliza(X,valor_minimo,valor_maximo)

%funcao que normaliza as entradas dadas

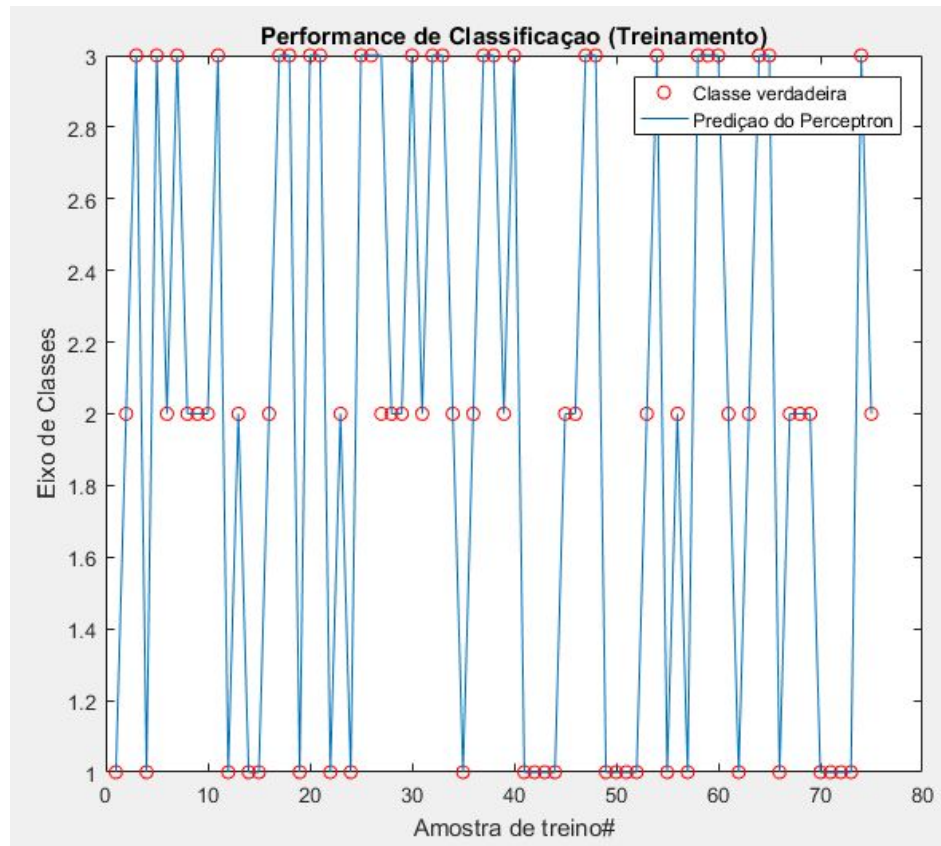
X_normalizado = X - min(X(:));
X_normalizado = (X_normalizado/range(X_normalizado(:)))*(valor_maximo-valor_minimo);
X_normalizado = X_normalizado + valor_minimo;
```

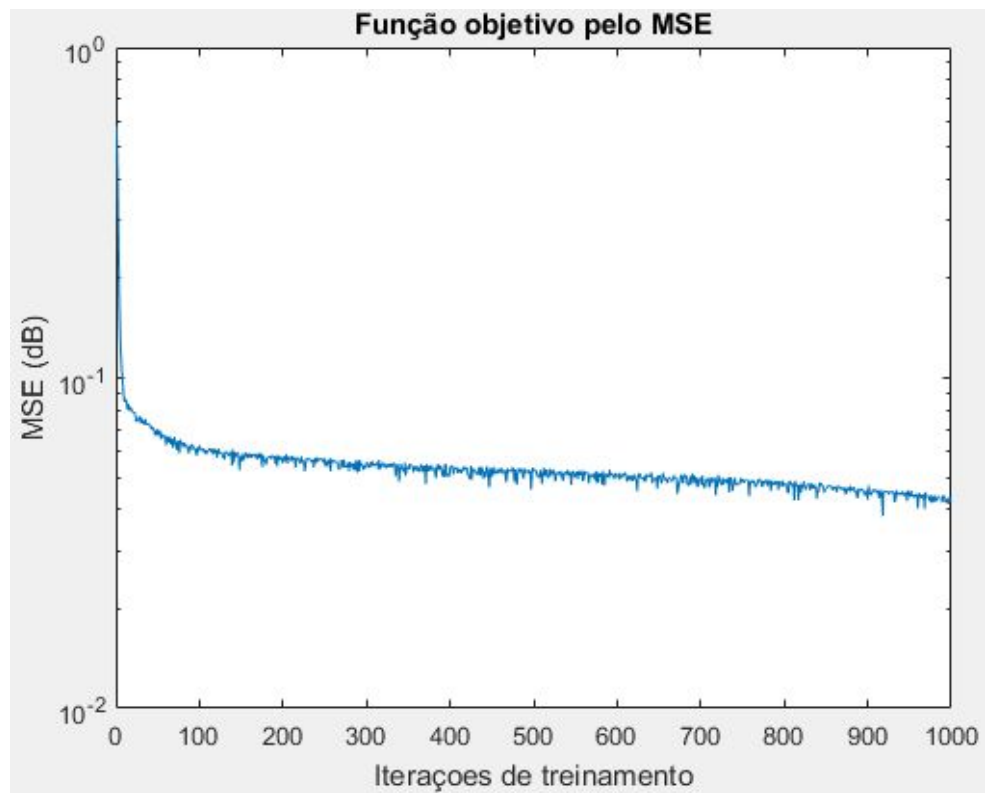
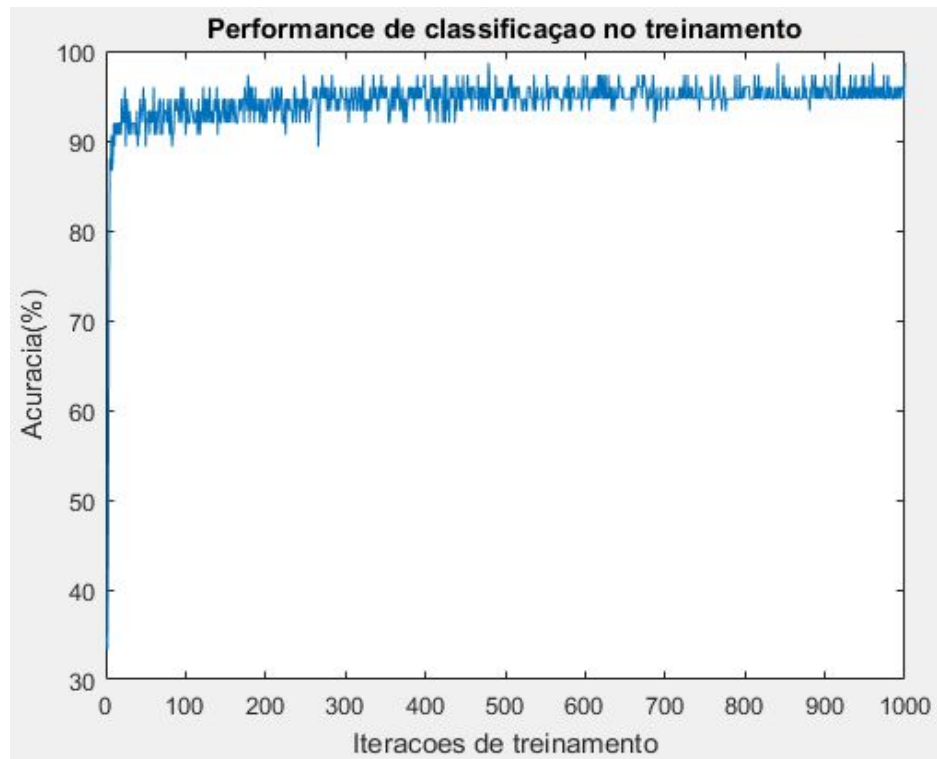
Inicialmente, os testes não convergiam, pois a taxa de aprendizado e o número de iterações (épocas) encontravam-se inadequados.

Portanto, alterando os parâmetros corretamente, após alguns testes, um dos melhores resultados possíveis de se obter com a rede ocorreu com a taxa de aprendizado em 0.01 e número de iterações 1000, sem aplicação de *momentum*, em uma rede com 4 neurônios na camada escondida.

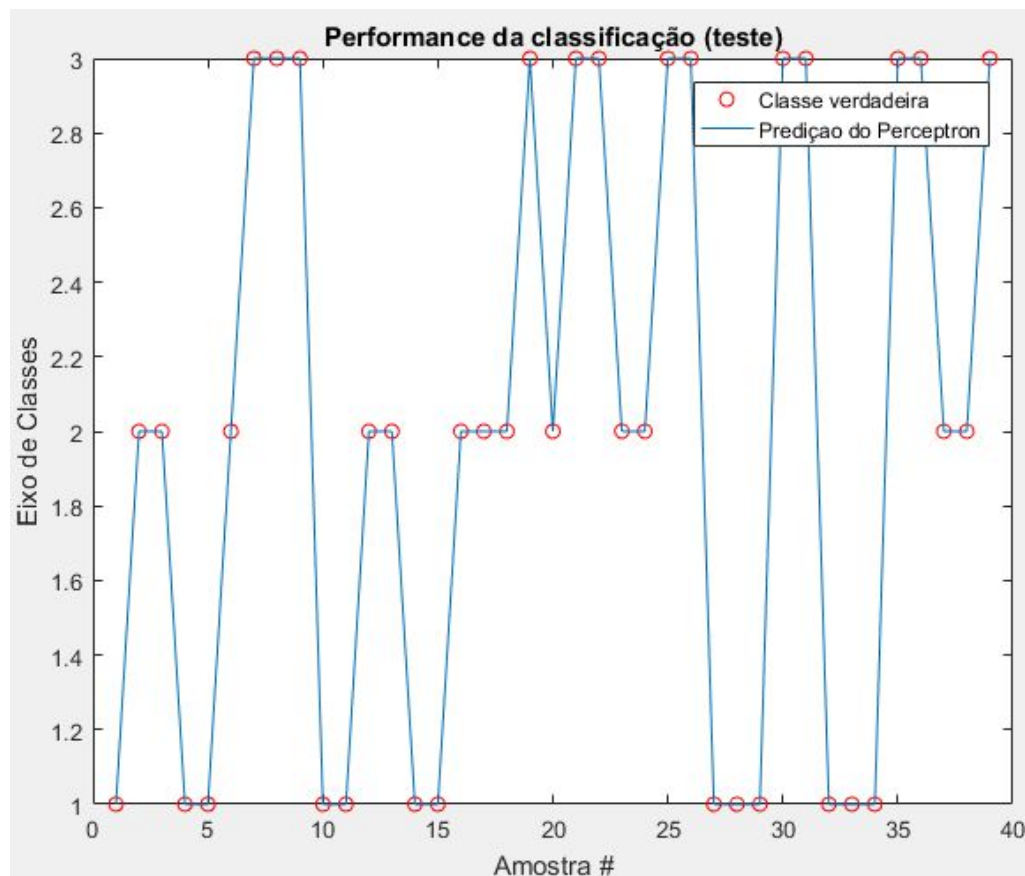
### Simulações com 4 valores de entrada:

Os resultados obtidos são apresentados abaixo:





Assim, o teste aplicado à rede foi resolvido com 100% de precisão:



Dado que a convergência ocorre em um número de iterações aceitável, o termo de *momentum* não se apresentou tão eficiente, sendo útil apenas para valores não tão adequados para a taxa de aprendizado e iterações. Por exemplo, foi testado para 10 iterações, com uma taxa de aprendizado de 0.2 e *momentum* igual a 0.05, obtendo-se uma acurácia de 66.7% no teste e treinamento. Isso ocorre porque o vetor auxiliar de pesos para *momentum* aumenta seus valores exponencialmente, o que não permite a classificação correta de todas as amostras.

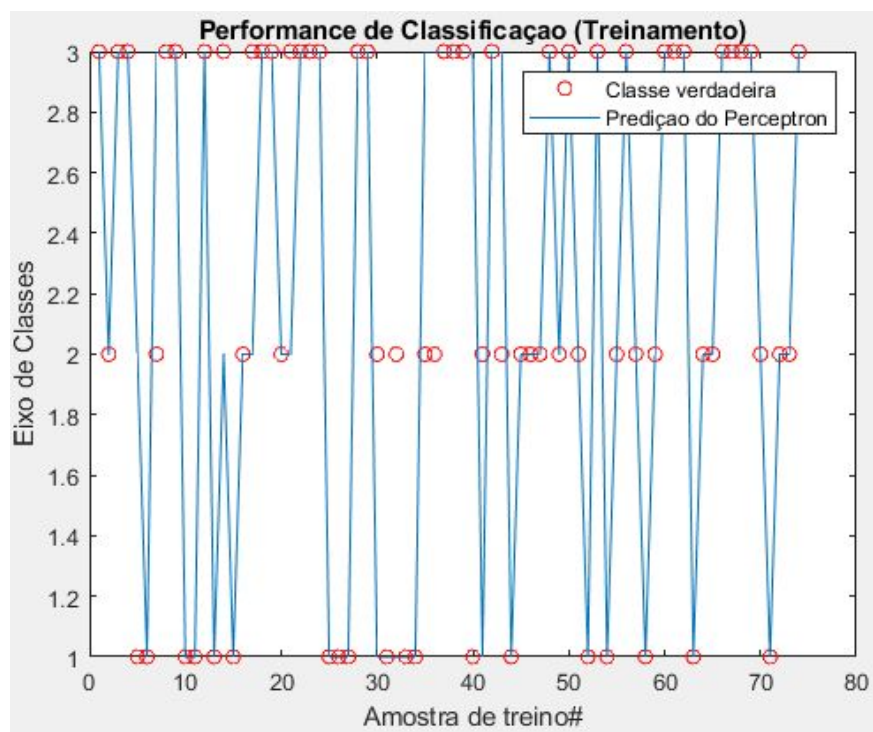
Os valores finais são registrados no *Workspace* do MatLab, evidenciando o formato dos vetores de dados, iterações, dentre outros. Além disso, nessa execução, observa-se uma acurácia de treinamento de 98.667% (97% na média das execuções, aproximadamente) e uma acurácia de teste de 100% nas execuções do código:

Workspace	
Name ▲	Value
a1	[0.9968;0.2293;0.8549;-0.9401]
a2	0.8984
Acuracia	1x1000 double
Acuracia_teste	100
Acuracia_treino	98.6667
Alvos_teste	1x39 double
Alvos_treino	1x75 double
aux1	[3.2203;0.2335;1.2742;-1.7390]
aux2	1.4638
Dados_teste	39x4 double
Dados_treino	75x4 double
Data_teste	39x5 double
Data_treino	75x5 double
e	0.3465
indicealeatorio	1000x75 double
Input	[1 0.5000 -0.3158 0.3158 -0.5263]
iteracoes	1000
j	1000
k	39
LR	0.0100
Max_alvos_teste	3
Max_alvos_treino	3
Min_alvos_teste	1
Min_alvos_treino	1
momentum	0
MSE	1x1000 double
N0	5
N1	4
N2	1
Output	1x39 double
Output_treino	1x75 double
SE	1000x75 double
w1	4x5 double
w2	[0.5938 -2.6453 -0.2298 -1.7817]
Y1	[0.0022;-0.4766;-0.0334;-0.3273]
Y2	0.6098

### **Simulações com 9 valores de entrada:**

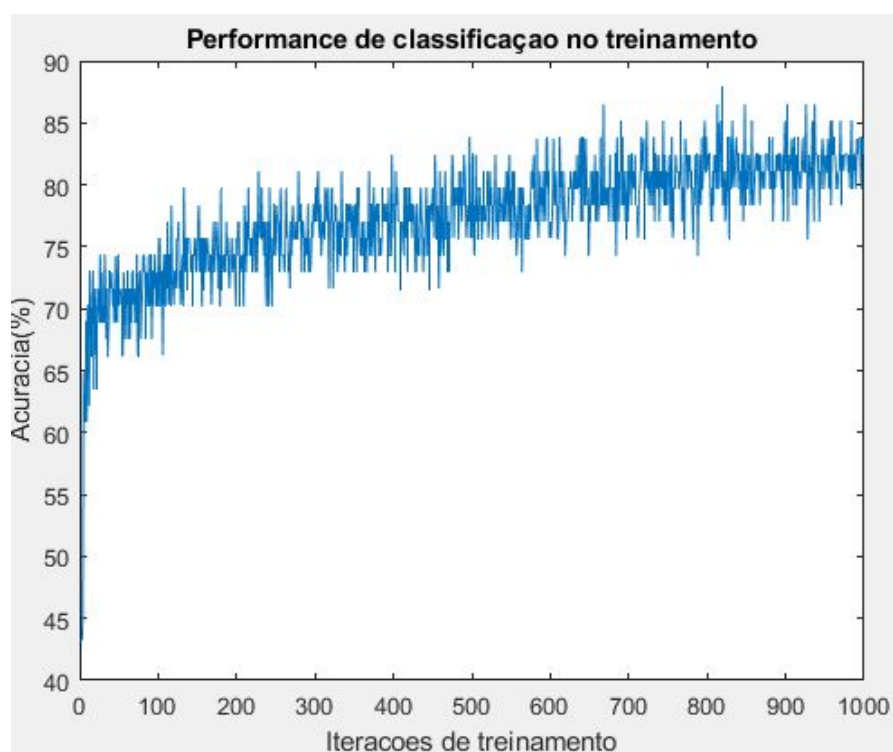
Repetindo o procedimento de maior eficiência, obtemos os seguintes resultados, primeiramente para o treinamento, e depois para o teste:



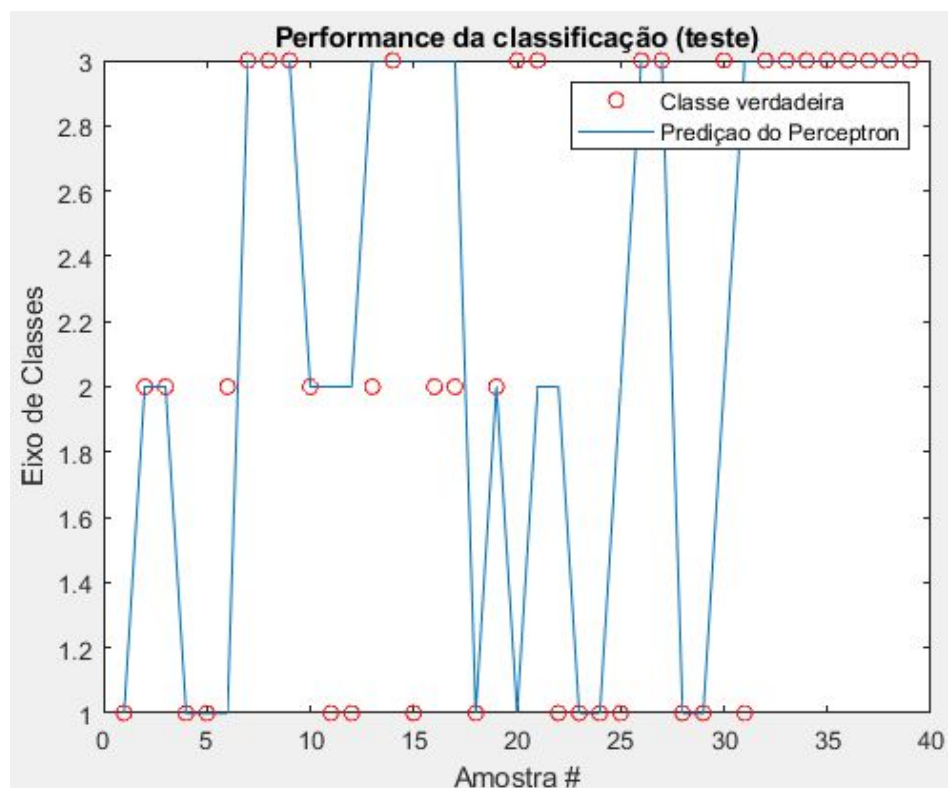
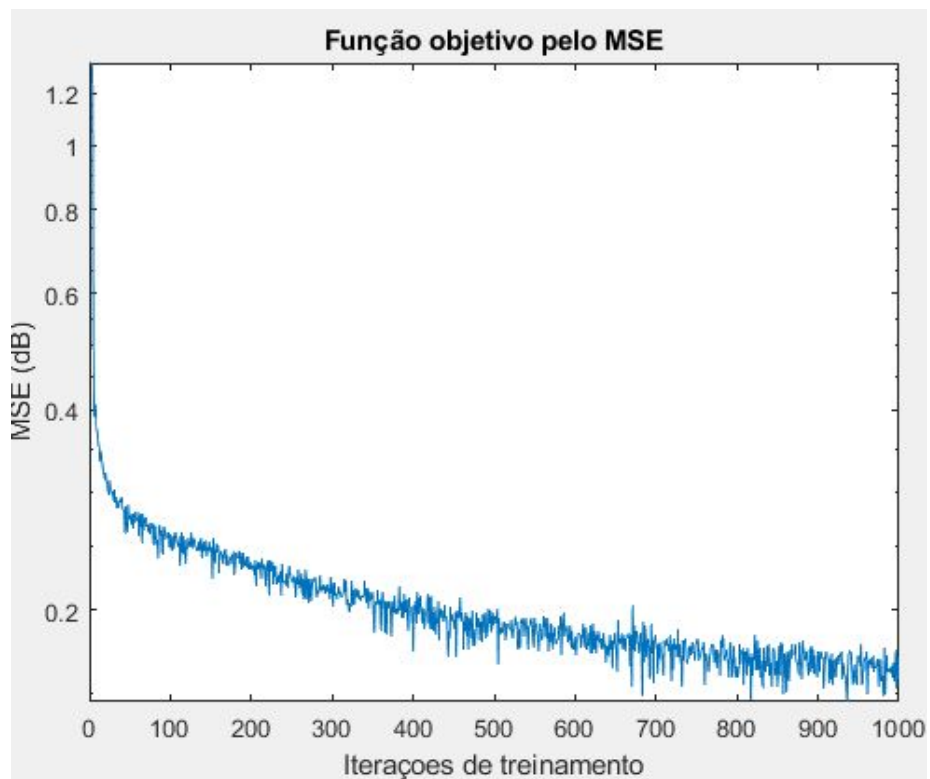


Acuracia\_treino =

83.7838







Acuracia\_teste =

66.6667

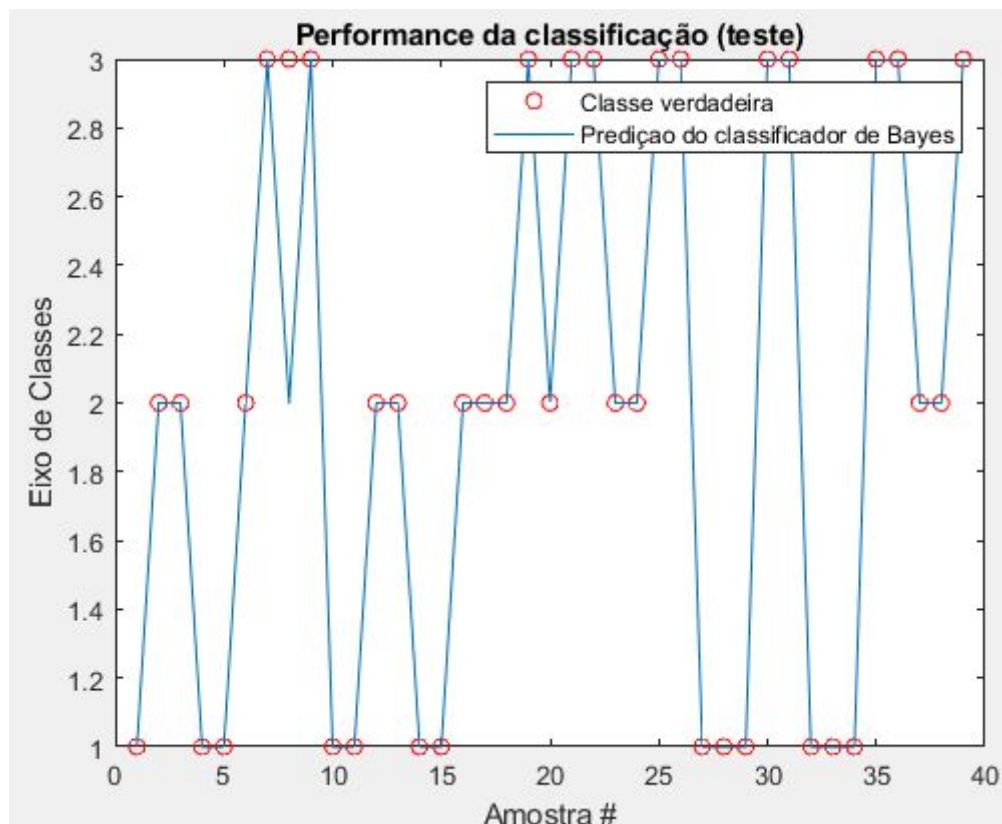
Testando a rede com o mesmo algoritmo, nota-se que o classificador perde uma parte da acurácia, visto que há mais entradas no sistema, que dificultam o aprendizado do classificador. Mesmo assim, ainda há uma classificação funcional em evidência.

O código é mostrado a seguir. Para uma melhor análise, recomenda-se a abertura do programa enviado em conjunto deste relatório.

## **Classificador Linear**

Agora, deseja-se criar treinar um algoritmo supervisionado que classifica dados de entrada a partir de um classificador linear, de modo a evitar regiões de indeterminação. Nesse contexto, foi desenvolvido um classificador por meio da utilização do comando “*fitcecoc*”, que parte do princípio LDA (*Linear Discriminant Analysis*), com o auxílio do “*Statistics and Machine learning toolbox*” do próprio Matlab.

Com esta rede, obteve-se 35 acertos dos 36 dados de teste, e, portanto, uma acurácia de 97.436%, como mostra a imagem abaixo:



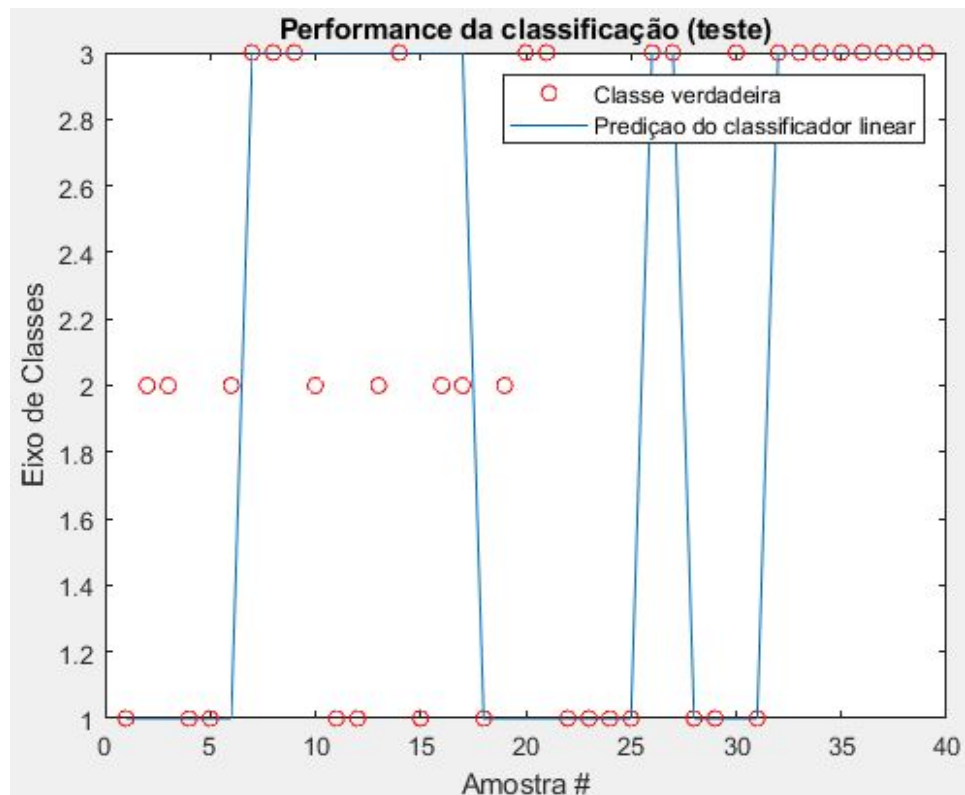
Os valores finais são registrados no *Workspace* do MatLab, evidenciando o formato dos vetores de dados, iterações, dentre outros:

Workspace	
Name ▲	Value
Acuracia_teste	97.4359
Alvos_teste	1x39 double
Alvos_treino	1x75 double
Classes_obtidas_teste	39x1 double
Classes_obtidas_teste_Bayes	1x39 double
Classificador_treinado_Bayes	1x1 ClassificationNai...
Dados	75x5 double
Dados_teste	39x4 double
Dados_treino	75x4 double
Data_teste	39x5 double
Data_treino	75x5 double
Erros	1x39 double

Dadas as condições do projeto, foi possível explorar boa parte do vasto campo de pesquisa de *Deep Learning*, principalmente pela ferramenta do Matlab, o que motivou o grupo a explicitar tais processos e resultados do trabalho.

### **Simulações com 9 valores de entrada:**

Repetindo o mesmo procedimento, obtemos os seguintes resultados:



Acuracia\_teste =

64.1026

A partir dos resultados obtidos, concluímos que o classificador também perdeu eficiência para mais entrada de dados, apresentando uma acurácia de 64.1%.

## Código Fonte do Classificador Linear

```
%%% Enrico Antonio Jehá Molento de Moraes    9838219
%%% Classificador linear - PTC3101 - Professor Felipe Pait

clc
clear all
close all

>Data_treino = load('treina1.txt') %Utilizado para 9 entradas
>Data_teste = load('teste1.txt')
Data_treino = load ('treina.txt');
Data_teste = load ('teste.txt');

Dados_treino = Data_treino(:,1:4); %leitura dos dados de treinamento
Alvos_treino = Data_treino(:,5)';
Dados_teste = Data_teste(:,1:4); %leitura dos dados de teste e
exclusão da linha 5 no arquivo de teste
Alvos_teste = Data_teste(:,5)'; %classes verdadeiras do teste
Dados = [Dados_treino Alvos_treino];

Classificador_Linear_treinado = fitcecoc(Dados_treino, Alvos_treino);
%utilização da função do deep Learning Toolbox
Classes_obtidas_teste =
predict(Classificador_Linear_treinado, Dados_teste); %predição dos
dados de teste pelo classificador treinado

Classificador_Linear_treinado = Classes_obtidas_teste';
Erros = [Alvos_teste - Classes_obtidas_teste'] % Vetor de Erros

%% plotagem das amostras e classes obtidas %%
figure
plot(Alvos_teste, 'or')
hold on
plot(round(Classes_obtidas_teste))
legend('Classe verdadeira', 'Predição do classificador linear')
xlabel('Amostra #')
ylabel('Eixo de Classes')
title('Performance da classificação (teste)')
Acuracia_teste =
length(find((round(Classes_obtidas_teste') - Alvos_teste) == 0)) * 100 / length(Classes_obtidas_teste);
Acuracia_teste
```

## Código Fonte do Classificador Não-Linear

```
%%% Enrico Antonio Jehá Molento de Moraes    9838219
%%% Classificador nao linear - PTC3101 - Professor Felipe Pait
```

```

clc
clear all
close all

%Data_treino = load ('treinal.txt'); txt com 9 entradas
%Data_teste = load ('teste1.txt');
Data_treino = load ('treina.txt');
Data_teste = load ('teste.txt');

Dados_treino = Data_treino(:,1:4); %leitura dos dados de treinamento
Alvos_treino = Data_treino(:,5)';
Dados_teste = Data_teste(:,1:4); %leitura dos dados de teste e
exclusão da linha 5 no arquivo de teste
Alvos_teste = Data_teste(:,5)'; %classes verdadeiras do teste

Max_alvos_treino=max(max(Alvos_treino)); % salva os pontos para
desnormalizar depois
Min_alvos_treino=min(min(Alvos_treino));
Max_alvos_teste=max(max(Alvos_teste));
Min_alvos_teste=min(min(Alvos_teste));

Dados_treino=normaliza(Dados_treino,-1,1);% Normalizando dados para
media nula e variancia unitaria
Alvos_treino=normaliza(Alvos_treino,-1,1);
Dados_teste = normaliza(Dados_teste,-1,1);
Alvos_teste = normaliza(Alvos_teste,-1,1);
%% Parâmetros da rede neural
% Mudar N1 e N0 para 9, em caso de 9 entradas
N1=4; % Neuronios da camada escondida
N2=1; % Neuronio de saida
N0=4+1; % Neuronios da camada de entrada + bias

% Parametros de treinamento
LR = 0.01; % Taxa de aprendizado (Learning rate)
iteracoes = 1000; % Iterações
momentum = 0; % termo de momento, se desejado

% Inicialização dos pesos aleatorios
w1=randn(N1,N0); % Conexao entre a camada escondida e de entrada
w2=randn(N2,N1); % Conexao entre a camada escondida e de saida

%%%%Treinamento da rede%%%%%%%%
for j=1:iteracoes

    indicealeatorio(j,:)=randperm(length(Alvos_treino));
    % randomizar os dados de treinamento para melhorar o aprendizado
    w1_M = w1; %cria vetor que armazena os pesos para usar
momentum
    w2_M = w2;

    for k=1:size(Dados_treino,1)

        Input=[1 Dados_treino(indicealeatorio(j,k),:)]'; % {1} is for
bias

        % Camada de entrada
        aux1 = w1*Input';
        a1=tansig(aux1); %aplicação da função de ativação tanh(x) nas
camadas

```

```

    % Camada escondida
    aux2 = w2*a1;
    a2=tansig(aux2);

    % Camada de saida
    Output_treino(k)=a2;
    e = Alvos_treino(indicealeatorio(j,k)) - Output_treino(k);

    % BackPropagation
    Y2 = 2*dtansig(aux2,a2)*e; % Gradiente da camada de saida
    Y1 = diag(dtansig(aux1,a1),0)*w2'*Y2; % Gradiente da camada
    escondida na diagonal

    % principal para
    multiplicar em seguida
    w1 = w1 + LR*Y1*Input + momentum*w1_M; % Pesos da camada de
    entrada atualizados
    w2 = w2 + LR*Y2*a1' + momentum*w2_M; % Pesos da camada
    escondida atualizados
    % com termo de
    momentum
    SE(j,k)= e*e'; % Erro quadratico
end

MSE(j)=mean(SE(j,:)); % Erro quadratico médio (gradient descent)

% Taxa de acertos
Acuracia(j)=length(find((round(Output_treino)-Alvos_treino(indicealeat
orio(j,:))')==0))*100/length(Output_treino);
end

Output_treino =
normaliza(Output_treino,Min_alvos_treino,Max_alvos_treino);
%%Desnormalizar a saída
Alvos_treino =
normaliza(Alvos_treino,Min_alvos_treino,Max_alvos_treino);
% caso não haja a desnormalização, as amostras estarão em -1, 0 ou 1

%%%plotagem do treinamento%%%%%%%%
figure
semilogy(MSE)
xlabel('Iterações de treinamento')
ylabel('MSE (dB)')
title('Função objetivo pelo MSE')

figure
plot(Acuracia)
xlabel('Iteracoes de treinamento')
ylabel('Acuracia(%)')
title('Performance de classificação no treinamento')

figure
plot(Alvos_treino(indicealeatorio(j,:)),'or')
hold on
plot(round(Output_treino))

legend('Classe verdadeira','Predição do Perceptron')
xlabel('Amostra de treino#')
ylabel('Eixo de Classes')
title('Performance de Classificação (Treinamento)')

```

```

Acuracia_treino=length(find((round(Output_treino)-Alvos_treino(indicea
leatorio(j,:))')==0))*100/length(Output_treino);

%%%%%%%%%% Teste da rede com novas entradas %%%%%%%%%%
Output=[];

for k=1:size(Dados_teste,1)

    Input=[1 Dados_teste(k,:)];

    aux1 = w1*Input';
    a1=tansig(aux1);    % ativação: tansig para [-1,+1] e logsig
para[0,1]
%
    aux2 = w2*a1;
    a2=tansig(aux2);    % ativação da camada de saída

    Output(k)=a2;

end
Output = normaliza(Output,Min_alvos_teste,Max_alvos_teste);
%%Desnormalizar a saída
Alvos_teste = normaliza(Alvos_teste,Min_alvos_teste,Max_alvos_teste);
% caso não haja a desnormalização, as amostras estarão em -1, 0 ou 1

%plotagem do resultado do teste
figure
plot(Alvos_teste,'or')
hold on
plot(round(Output))
legend('Classe verdadeira','Predição do Perceptron')
xlabel('Amostra #')
ylabel('Eixo de Classes')
title('Performance da classificação (teste)')

Acuracia_teste=length(find((round(Output)-Alvos_teste')==0))*100/length
(Output);

Acuracia_treino
Acuracia_teste

```

### Referências interessantes:

1 - Artigo “Deep learning” por Yann LeCun, Yoshua Bengio & Geoffrey Hinton.

2- [https://www.youtube.com/watch?v=aircAruvnKk&ab\\_channel=3Blue1Brown](https://www.youtube.com/watch?v=aircAruvnKk&ab_channel=3Blue1Brown)