UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DIRETORIA DE PESQUISA E PÓS GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

JÔNATAS TRABUCO BELOTTI

IMPLEMENTAÇÃO PROJETO PRÁTICO 5.9: MLP PARA CLASSIFICAÇÃO DE PADRÕES

RELATÓRIO

PONTA GROSSA
2017

JÔNATAS TRABUCO BELOTTI

IMPLEMENTAÇÃO PROJETO PRÁTICO 5.9: MLP PARA CLASSIFICAÇÃO DE PADRÕES

Relatório apresentado como requisito parcial à obtenção de nota na disciplina de Fundamentos de Redes Neurais Artificiais do Programa de Pós-Graduação em Engenharia Elétrica, da Universidade Tecnológica Federal do Paraná– Campus Ponta Grossa.

Professor: Prof. Dr. Sérgio Okida

PONTA GROSSA
2017

SUMÁRIO

1 INTRODUÇÃO

O *Perceptron* de Múltiplas Camadas (PMC, ou MLP do inglês *Multilayer Perceptron*) é constituído por um conjunto de neurônios artificiais dispostos em varias camadas, de modo que o sinal de entrada se propaga para frente através da rede, camada por camada. Dentre suas camadas existe a camada de entrada que recebe os sinais de entrada, a camada de saída que entrega o resultado obtido pela rede e no meio dessas podem existir quantas camadas forem necessárias. Essas camadas são chamadas intermediárias ou ocultas (HAYKIN, 2001). Note que uma MLP é constituida de pelo menos 2 camadas neurais, sendo uma camada de saída e pelo menos 1 camada escondida (SILVA; SPATTI; FLAUZINO, 2010).

As MLPs são consideradas uma das arquiteturas mais versáteis quanto a aplicabilidade, sendo utilizadas em diversas áreas do conhecimento. Dentre as utilizações da MLP estão: aproximação universal de funções, reconhecimento de padrões, identificação e controle de processos, previsão de series temporais e otimização de sistemas (SILVA; SPATTI; FLAUZINO, 2010).

Esse relatório tem como objetivo descrever o desenvolvimento do Projeto Prático 5.9 do livro Redes neurais artificias para engenharia e ciências aplicadas de Silva, Spatti e Flauzino (2010), o projeto consiste na implementação, treinamento e teste de uma rede neural do tipo *Perceptron* Multicamadas para ser usada como classificadora de padrões com o objetivo de auxiliar no processamento de bebidas.

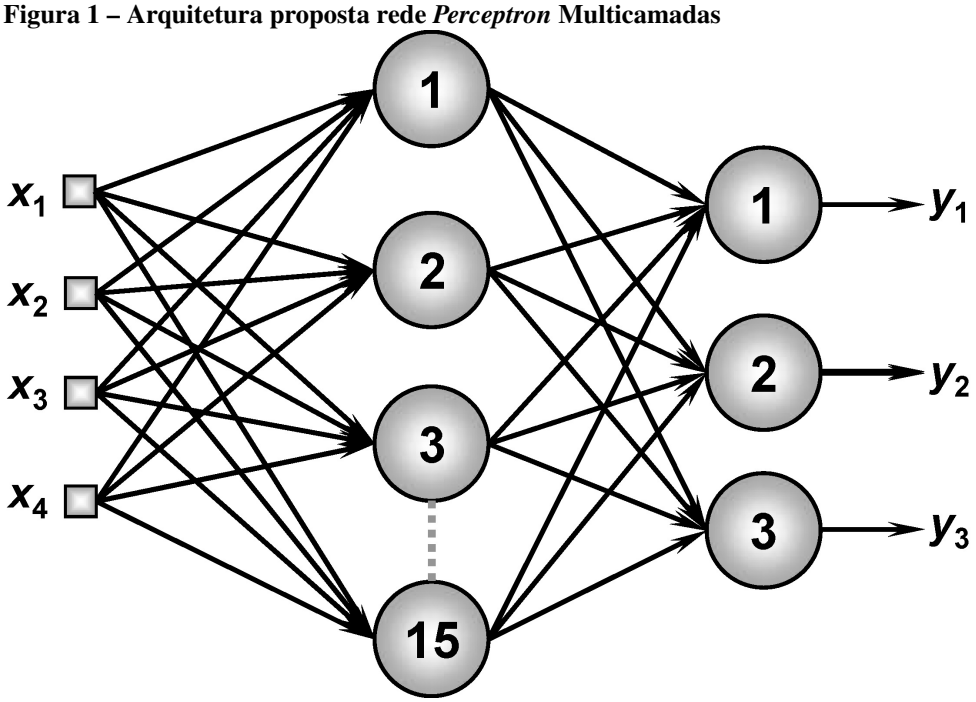
1.1 ESTUDO DE CASO

O projeto prático 5.9 do livro Redes neurais artificias para engenharia e ciências aplicadas de Silva, Spatti e Flauzino (2010), mostra que no processamento de bebidas, é aplicado um conservante à bebida em função da analise de 4 características medidas em cada lote de bebida, definidas por x_1 (teor de água), x_2 (grau de acidez), x_3 (temperatura) e x_4 (tensão interfacial). Existem apenas 3 possibilidades de conservantes para serem adicionados as bebidas, denotados por **A**, **B** e **C**.

O objetivo do projeto é o desenvolvimento, treinamento e teste de uma MLP que determine qual dos conservantes deve ser adicionado a cada lote de bebida a partir da análise das 4 características medidas. É determinada que a arquitetura da rede, que deve conter 4 sinais de entrada, 2 camadas neurais, sendo 15 neurônios na camada escondida e 3 neurônios na camada de saída. A Figura 1 mostra detalhadamente a arquitetura que a MLP deve ter para a realização do projeto.

Analisando a arquitetura proposta na Figura 1 nota-se que a MLP possuirá 3 sinais de saída, como a resposta da MLP deve ser o tipo de conservante (A, B ou C) se faz necessário padronizar quais combinações de saída representarão cada conservante. Essa padronização pode

Figura 1 – Arquitetura proposta rede *Perceptron* Multicamadas



Fonte: (SILVA; SPATTI; FLAUZINO, 2010).

ser vista na Tabela 1.

Tabela 1 – Padronização da saída da MLP

Tipo de conservante	y_1	y_2	y_3
Tipo A	1	0	0
Tipo B	0	1	0
Tipo C	0	0	1

Fonte: Autoria própria.

2 DESENVOLVIMENTO DO PROJETO

A rede MLP proposta na Seção 2.4 foi desenvolvida na linguagem Java, a classe MLP é a responsável por implementar o funcionamento da rede, seu código fonte está disponível no Apêndice A. Com o intuito de facilitar o acesso a rede desenvolvida todo o código fonte, juntamente com o programa já compilado e os arquivos de treinamento e de teste estão disponíveis em um repositório do *GitHub* [1] que pode ser acessado pelo link <https://github.com/jonatastbelotti/MLPClassificacaoPadrao>.

As seções 2.1, 2.2, 2.3 e 2.4 apresentam as discussões a respeito dos treinamentos e testes realizados na MLP.

2.1 TREINAMENTO COM ALGORITMO *BACKPROPAGATION*

A rede neural Perceptron Multicamadas foi treinada utilizando o algoritmo de aprendizagem *backpropagation*. Para tanto os pesos sinápticos iniciais foram gerados de forma aleatória com valores entre 0 e 1. Os valores inciais dos pesos sinápticos são apresentados a seguir:

```
Pesos camada escondida:
N1 = 0,107536 0,287204 0,988448 0,269921 0,595165
N2 = 0,778432 0,468819 0,848750 0,809264 0,984074
N3 = 0,870440 0,929182 0,649449 0,641384 0,600356
N4 = 0,134505 0,718753 0,872210 0,824644 0,652206
N5 = 0,755379 0,247501 0,331459 0,701983 0,818383
N6 = 0,065348 0,447899 0,400095 0,730609 0,207458
N7 = 0,465652 0,559282 0,787082 0,451635 0,363438
N8 = 0,429271 0,148211 0,786839 0,353487 0,504985
N9 = 0,654516 0,102525 0,924824 0,307877 0,023243
N10 = 0,605466 0,607867 0,480391 0,517105 0,454927
N11 = 0,642867 0,624359 0,211529 0,400999 0,098983
N12 = 0,061225 0,929147 0,661139 0,761268 0,095299
N13 = 0,758948 0,876337 0,279626 0,420834 0,832334
N14 = 0,751103 0,656794 0,029495 0,689504 0,900353
N15 = 0,141663 0,346921 0,136945 0,501500 0,989760
Pesos camada de saída:
N1 = 0,355341 0,970452 0,581492 0,354469 0,143825 0,317984 0,156232
     0,780592 0,668531 0,642778 0,983764 0,386072 0,502407 0,639562
```

[1] No repositório do *GitHub* o caminho para acessar a classe MLP.java é 'MLPClassificaoPadrao/src/Modelo/MLP.java'

```
     0,392809 0,838261
N2 = 0,007120 0,355295 0,504662 0,657817 0,669543 0,870788 0,179184
     0,216368 0,310969 0,790655 0,184540 0,033314 0,499279 0,606461
     0,415555 0,622651
N3 = 0,772612 0,721538 0,466979 0,192461 0,217666 0,555839 0,505676
     0,957716 0,536568 0,685324 0,179671 0,691314 0,983315 0,412770
     0,252200 0,408747
```

Foram utilizadas uma taxa de aprendizagem $\eta = 0,1$ e uma precisão de $\varepsilon = 10^{-6}$. O conjunto de dados utilizado para o treinamento da MLP está disponível no Anexo A.

A rede foi iniciada com um Erro quadrático médio de $E_{qm} = 0,969800$ e ao final de $7,13$ segundos o treinamento foi finalizado, tendo levado 1167 épocas para obter $E_{qm} = 0,025987$. Os pesos sinápticos obtidos ao final do treinamento foram:

```
Pesos camada escondida:
N1 = -1,510861 0,088077 0,566373 -0,324799 0,183600
N2 = 9,523669 2,555060 3,787443 4,998849 3,577444
N3 = 7,589782 3,520399 5,458402 5,619990 3,920627
N4 = 4,907093 2,808587 4,240590 3,951884 2,922394
N5 = 4,963422 2,221356 4,043916 3,976752 3,265644
N6 = -0,226388 0,561020 0,857369 0,641873 0,383079
N7 = 6,813671 2,049162 2,741281 3,374751 2,467509
N8 = 0,066571 0,249918 1,001865 0,292513 0,644953
N9 = 0,548289 -0,229772 0,527223 -0,181457 -0,468084
N10 = -1,980162 -0,423890 -1,213491 -1,518960 -1,018689
N11 = 4,743890 1,693819 1,762479 2,292896 1,713066
N12 = 4,001849 1,457595 1,813445 2,198073 1,125105
N13 = 5,027010 1,727022 1,737807 2,292377 2,174540
N14 = 5,671584 1,716370 1,858657 2,760279 2,490358
N15 = -2,381570 -0,745227 -1,577350 -1,661411 -0,724918
Pesos camada de saída:
N1 = -2,935681 0,996104 -3,221576 -4,831441 -2,296207 -3,823373 -0,643523
     -1,742724 -0,262937 0,455740 2,399089 -1,341618 -1,948689 -1,811138
     -1,974467 2,733235
N2 = 2,840502 -1,615977 -7,153427 8,517873 5,696752 4,820299 -1,341960
     -4,811377 -1,422140 -0,161324 -0,825257 -3,281889 -2,147716 -3,096454
     -3,801467 -0,563837
N3 = 6,791618 -1,701707 5,681341 1,158218 -1,585635 1,801072 -0,655237
     4,039734 -0,323526 -0,677612 -3,162048 2,678222 2,728488 3,157097
     3,287382 -3,453343
```

2.2 TREINAMENTO COM ALGORITMO *BACKPROPAGATION* COM *MOMENTUM*

A MLP também foi treinada com a utilização do algoritmo de aprendizagem *backpropagation* com *momentum*, que é uma variação do algoritmo de aprendizagem *backpropagation* utilizada na Seção 2.1. Para esse treinamento foram utilizadas as mesmas matrizes de pesos sinápticos iniciais geradas no treinamento da Seção 2.1. Foram utilizadas uma taxa de aprendizagem $\eta = 0,1$, uma precisão de $\varepsilon = 10^{-6}$ e fator *momentum* de $\alpha = 0,9$.

A rede foi iniciada com um Erro quadrático médio de $E_{qm} = 0,969800$ e ao final de $1,63$ segundos o treinamento foi finalizado, tendo levado 285 épocas para obter $E_{qm} = 0,027476$. Os pesos sinápticos obtidos ao final do treinamento foram:

```
Pesos camada escondida:
N1 = 0,364231 -0,101421 0,478615 -0,515954 0,050560
N2 = 5,744250 1,212367 2,394739 3,021475 2,333066
N3 = 3,580229 1,993885 2,992756 3,151278 2,242548
N4 = 5,273199 2,790674 3,928003 4,560609 3,062594
N5 = 4,146302 0,870298 1,483484 2,253496 1,862166
N6 = 0,285704 0,087872 -0,111363 -0,103668 -0,338788
N7 = 2,588926 0,737709 1,262036 1,129330 0,830946
N8 = 1,320428 0,101101 0,805437 0,294167 0,504498
N9 = 1,388367 0,114231 1,003829 0,358607 0,047355
N10 = 1,612350 0,604587 0,545561 0,574071 0,535121
N11 = 1,439040 0,728675 0,331504 0,470565 0,230423
N12 = 0,867900 0,859955 0,680834 0,720724 0,019922
N13 = 4,175834 1,413048 1,391555 1,865061 1,814576
N14 = 4,111459 1,247607 1,159302 2,066459 1,907613
N15 = -0,005413 -0,131148 -0,702680 -0,600535 0,227005
Pesos camada de saída:
N1 = -9,858128 0,740417 -7,236521 -8,399917 -4,192569 -4,444404 1,294910
     -2,781377 -0,872503 -0,849482 -0,765183 -0,037756 -1,061603 -4,119547
     -3,645823 2,788173
N2 = 4,792244 -1,954569 -9,790357 9,205573 19,638707 -6,618097 -1,602220
     -4,128565 -2,710568 -2,269847 -2,980937 -2,647401 -0,663291 -6,989166
     -6,994032 -1,123327
N3 = 15,010818 -2,783439 9,925807 4,975526 -2,800946 7,104502 -2,854327
     3,528069 0,093214 0,188790 0,927912 0,314883 -0,053431 6,482948
     6,376826 -3,627838
```

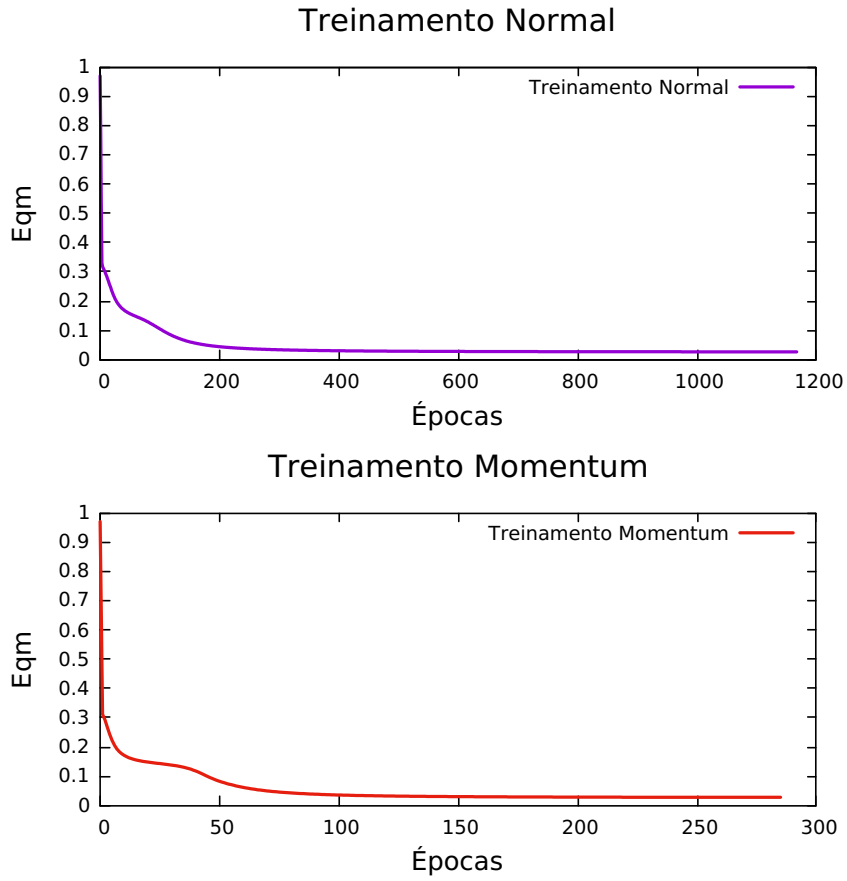
2.3 ANÁLISE DOS TREINAMENTOS

Perceba que apesar de os 2 treinamentos das seções 2.1 e 2.2 terem sido iniciados com as mesmas matrizes de pesos sinápticos, as matrizes de pesos sinápticos resultantes, o Erro quadrático médio resultante, o número de épocas necessárias para o treinamento e o tempo necessário para o treinamento foram diferentes.

A adição do termo de *momentum* no treinamento da MLP resulta no aceleramento da conversão dos pesos sinápticos para os valores finais. De forma que quanto mais longe os pesos sinapticos estiverem dos valores finais maior será o ajuste sofrido por eles (SILVA; SPATTI; FLAUZINO, 2010). Isso fica evidente ao compararmos o número de épocas do treinamento sem *momentum* (1167 épocas) com o número de épocas do treinamento com *momentum* (285 épocas), a aceleração do treinamento pelo termo de *momentum* gerou um treinamento com 882 épocas a menos. Consequentemente, menos épocas resultam em um menor tempo necessário para o treinamento.

Os gráficos da Figura 2 apresentam a relação dos valores de Erro quadrático médio em função de cada época de treinamento, tanto para o treinamento com *momentum* como sem.

Figura 2 – Comparação dos treinamentos



Fonte: Autoria própria.

Analisando os gráficos da Figura 2 nota-se que o comportamento das duas curvas são

similares, ambas tem um período onde a velocidade de decaimento do E_{qm} é maior, e ao decorrer das épocas essa velocidade vai diminuindo até ser menor que a precisão estabelecida e o treinamento ter então o seu fim. No treinamento com *momentum* o período com maior velocidade de decaimento do E_{qm} vai da época 0 até a época 50, por sua vez no treinamento sem *momentum* esse período é compreendido entre as épocas 0 e 200.

Pelos gráficos da Figura 2 mais uma vez é possível notar que a velocidade de decaimento do E_{qm} é maior no treinamento realizado pelo algoritmo *backpropagation* com *momentum*.

2.4 TESTE DA REDE

Após o treinamento da rede a mesma foi submetida a um teste, com o objetivo de validar o poder de abstração da rede obtido pelo processo de treinamento. O teste foi realizado mediante a comparação dos valores fornecidos pela rede com os valores desejadas para cada amostra de teste. A Tabela 2 apresenta o conjunto de teste utilizado. Esse mesmo conjunto de teste está disponível no Anexo B.

Tabela 2 – Conjunto de padrões de teste

Amostra	$\mathbf{x_1}$	$\mathbf{x_2}$	$\mathbf{x_3}$	$\mathbf{x_4}$	$\mathbf{d_1}$	$\mathbf{d_2}$	$\mathbf{d_3}$
1	$0,8622$	$0,7101$	$0,6236$	$0,7894$	0	0	1
2	$0,2741$	$0,1552$	$0,1333$	$0,1516$	1	0	0
3	$0,6772$	$0,8516$	$0,6543$	$0,7573$	0	0	1
4	$0,2178$	$0,5039$	$0,6415$	$0,5039$	0	1	0
5	$0,7260$	$0,7500$	$0,7007$	$0,4953$	0	0	1
6	$0,2473$	$0,2941$	$0,4248$	$0,3087$	1	0	0
7	$0,5682$	$0,5683$	$0,5054$	$0,4426$	0	1	0
8	$0,6566$	$0,6715$	$0,4952$	$0,3951$	0	1	0
9	$0,0705$	$0,4717$	$0,2921$	$0,2954$	1	0	0
10	$0,1187$	$0,2568$	$0,3140$	$0,3037$	1	0	0
11	$0,5673$	$0,7011$	$0,4083$	$0,5552$	0	1	0
12	$0,3164$	$0,2251$	$0,3526$	$0,2560$	1	0	0
13	$0,7884$	$0,9568$	$0,6825$	$0,6398$	0	0	1
14	$0,9633$	$0,7850$	$0,6777$	$0,6059$	0	0	1
15	$0,7739$	$0,8505$	$0,7934$	$0,6626$	0	0	1
16	$0,4219$	$0,4136$	$0,1408$	$0,0940$	1	0	0
17	$0,6616$	$0,4365$	$0,6597$	$0,8129$	0	0	1
18	$0,7325$	$0,4761$	$0,3888$	$0,5683$	0	1	0

Fonte: Adaptado de Silva, Spatti e Flauzino (2010).

A rede foi testada tanto com o treinamento realizado por meio do algoritmo de aprendizagem *backpropagation* (Seção 2.1) como com o treinamento realizado por meio do algoritmo de aprendizagem *backpropagation* com *momentum* (Seção 2.2). Os resultados obtidos pelos pesos de cada treinamento para cada uma das amostras de teste da Tabela 2 podem ser vistos na Tabela 3.

Tabela 3 – Conjunto de padrões de teste

Amostra	$\mathbf{d_1}$	$\mathbf{d_2}$	$\mathbf{d_3}$	***Backpropagation***			***Momentum***		
				$\mathbf{y_1^{pós}}$	$\mathbf{y_2^{pós}}$	$\mathbf{y_3^{pós}}$	$\mathbf{y_1^{pós}}$	$\mathbf{y_2^{pós}}$	$\mathbf{y_3^{pós}}$
1	0	0	1	0	0	1	0	0	1
2	1	0	0	1	0	0	1	0	0
3	0	0	1	0	0	1	0	0	1
4	0	1	0	0	1	0	0	1	0
5	0	0	1	0	0	1	0	0	1
6	1	0	0	1	0	0	1	0	0
7	0	1	0	0	1	0	0	1	0
8	0	1	0	0	1	0	0	1	0
9	1	0	0	1	0	0	1	0	0
10	1	0	0	1	0	0	1	0	0
11	0	1	0	0	1	0	0	1	0
12	1	0	0	1	0	0	1	0	0
13	0	0	1	0	0	1	0	0	1
14	0	0	1	0	0	1	0	0	1
15	0	0	1	0	0	1	0	0	1
16	1	0	0	1	0	0	1	0	0
17	0	0	1	0	0	1	0	0	1
18	0	1	0	0	1	0	0	1	0
Total de acertos (%)				100%			100%		

Fonte: Autoria própria.

Na Tabela 3 além da resposta fornecida pela MLP (após o pós-processamento) mediante cada treinamento é possível verificar o porcentual de acerto de cada treinamento. Apesar dos valores de Erro quadrático médio obtidos pelos treinamentos com o algoritmo *backpropagation* e *backpropagation* com *momentum* terem sido diferentes, os 2 treinamentos apresentaram o mesmo resultado no teste, ambos obtiveram porcentagem de acertado de 100% em todos os testes realizados.

3 CONCLUSÃO

Foi desenvolvida uma rede neural Perceptron multicamadas composta por 4 sinais de entrada, 1 camada escondida com 15 neurônios e 1 camada de saída com 3 neurônio para ser usada como classificadora de padrões no processo de produção de bebidas. A rede foi treinada utilizando o algoritmo de aprendizagem *backpropagation* obtendo $\mathrm{E_{qm}} = 0,025987$ ao final de 1167 épocas de treinamento. Também foi realizado um treinamento com o algoritmo de aprendizagem *backpropagation* com *momentum*, obtendo $\mathrm{E_{qm}} = 0,027476$ ao fim de 285 épocas.

Conclui-se que o melhor treinamento para a MLP proposta é o obtido através do algoritmo de aprendizagem *backpropagation* com *momentum*, pois apesar dos 2 treinamentos terem obtido porcentagem de acerto de 100% nos testes o treinamento com *momentum* foi executado com 882 épocas a menos.

REFERÊNCIAS

HAYKIN, Simon. **Redes Neurais: principios e prática**. 2. ed. Porto Alegre: Bookman, 2001. ISBN 978-85-7307-718-6.

SILVA, Ivan Nunes da; SPATTI, Danilo Hernane; FLAUZINO, Rogério Andrade. **Redes Neurais Artificiais Para Engenharia e Cincias Aplicadas - Curso Pratico**. 1. ed. São Paulo: ARTLIBER, 2010. ISBN 978-85-88098-53-4.

APÊNDICE A - IMPLEMENTAÇÃO DA CLASSE MLP EM JAVA

```
1  package Modelo;
2
   import Controle.Comunicador;
4  import Recursos.Arquivo;
   import java.io.BufferedReader;
6  import java.io.FileNotFoundException;
   import java.io.FileReader;
8  import java.io.IOException;
   import java.util.Random;
10
   /**
12  *
    * @author Jônatas Trabuco Belotti [jonatas.t.belotti@hotmail.com]
14  */
   public class MLP {
16
     public static final int NUM_ENTRADAS = 4;
18   private final int NUM_NEU_CAMADA_ESCONDIDA = 15;
     public static final int NUM_NEU_CAMADA_SAIDA = 3;
20   private final double TAXA_APRENDIZAGEM = 0.1;
     private final double PRECISAO = 0.000001;
22   private final double FATOR_MOMENTUM = 0.9;
     private final double BETA = 1.0;
24
     private int numEpocas;
26   private double fatorMomentum;
     private double[] entradas;
28   private double[][] pesosCamadaEscondidaInicial;
     private double[][] pesosCamadaEscondida;
30   private double[][] pesosCamadaEscondidaProximo;
     private double[][] pesosCamadaEscondidaAnterior;
32   private double[][] pesosCamadaSaidaInicial;
     private double[][] pesosCamadaSaida;
34   private double[][] pesosCamadaSaidaProximo;
     private double[][] pesosCamadaSaidaAnterior;
36   private double[] potencialCamadaEscondida;
     private double[] saidaCamadaEscondida;
38   private double[] potencialCamadaSaida;
     private double[] saidaCamadaSaida;
```

```
40  private double[] saidaEsperada;
    private double[] gradienteCamadaSaida;
42  private double[] gradienteCamadaEscondida;

44  public MLP() {
      Random random;
46
      entradas = new double[NUM_ENTRADAS + 1];
48    pesosCamadaEscondidaInicial = new double[
         NUM_NEU_CAMADA_ESCONDIDA][NUM_ENTRADAS + 1];
      pesosCamadaEscondida = new double[NUM_NEU_CAMADA_ESCONDIDA][
         NUM_ENTRADAS + 1];
50    pesosCamadaEscondidaAnterior = new double[
         NUM_NEU_CAMADA_ESCONDIDA][NUM_ENTRADAS + 1];
      pesosCamadaEscondidaProximo = new double[
         NUM_NEU_CAMADA_ESCONDIDA][NUM_ENTRADAS + 1];
52    pesosCamadaSaidaInicial = new double[NUM_NEU_CAMADA_SAIDA][
         NUM_NEU_CAMADA_ESCONDIDA + 1];
      pesosCamadaSaida = new double[NUM_NEU_CAMADA_SAIDA][
         NUM_NEU_CAMADA_ESCONDIDA + 1];
54    pesosCamadaSaidaAnterior = new double[NUM_NEU_CAMADA_SAIDA][
         NUM_NEU_CAMADA_ESCONDIDA + 1];
      pesosCamadaSaidaProximo = new double[NUM_NEU_CAMADA_SAIDA][
         NUM_NEU_CAMADA_ESCONDIDA + 1];
56    potencialCamadaEscondida = new double[NUM_NEU_CAMADA_ESCONDIDA
         + 1];
      saidaCamadaEscondida = new double[NUM_NEU_CAMADA_ESCONDIDA +
         1];
58    potencialCamadaSaida = new double[NUM_NEU_CAMADA_SAIDA];
      saidaCamadaSaida = new double[NUM_NEU_CAMADA_SAIDA];
60    saidaEsperada = new double[NUM_NEU_CAMADA_SAIDA];
      gradienteCamadaSaida = new double[NUM_NEU_CAMADA_SAIDA];
62    gradienteCamadaEscondida = new double[NUM_NEU_CAMADA_ESCONDIDA
         ];

64    //Iniciando pesos sinapticos
      Comunicador.iniciarLog("Iniciando os pesos sinapticos");
66    random = new Random();

68    for (int i = 0; i < NUM_NEU_CAMADA_ESCONDIDA; i++) {
        for (int j = 0; j < NUM_ENTRADAS + 1; j++) {
70        pesosCamadaEscondidaInicial[i][j] = random.nextDouble();
```

```
        }
72    }

74    for (int i = 0; i < NUM_NEU_CAMADA_SAIDA; i++) {
        for (int j = 0; j < NUM_NEU_CAMADA_ESCONDIDA + 1; j++) {
76        pesosCamadaSaidaInicial[i][j] = random.nextDouble();
        }
78    }

80    copiarMatriz(pesosCamadaEscondidaInicial, pesosCamadaEscondida)
        ;
      copiarMatriz(pesosCamadaSaidaInicial, pesosCamadaSaida);
82    imprimirPesos();
  }
84
  public boolean treinar(Arquivo arquivoTreinamento) {
86    Comunicador.limparLog();
      return treinar(arquivoTreinamento, false);
88  }

90  public boolean treinar(Arquivo arquivoTreinamento, boolean
      momentum) {
      FileReader arq;
92    BufferedReader lerArq;
      String linha;
94    double erroAtual;
      double erroAnterior;
96    long tempInicial;

98    copiarMatriz(pesosCamadaEscondidaInicial, pesosCamadaEscondida)
        ;
      copiarMatriz(pesosCamadaSaidaInicial, pesosCamadaSaida);
100   copiarMatriz(pesosCamadaEscondidaInicial,
        pesosCamadaEscondidaAnterior);
      copiarMatriz(pesosCamadaSaidaInicial, pesosCamadaSaidaAnterior)
        ;
102
      tempInicial = System.currentTimeMillis();
104   numEpocas = 0;
      erroAtual = erroQuadraticoMedio(arquivoTreinamento);
106
      if (!momentum) {
```

```
108         fatorMomentum = 0D;
            Comunicador.iniciarLog("Início treinamento da MLP");
110       } else {
            fatorMomentum = FATOR_MOMENTUM;
112         Comunicador.iniciarLog("Início treinamento com MOMENTUM da
               MLP");
          }
114
          Comunicador.addLog(String.format("Erro inicial: %.6f",
             erroAtual).replace(".", ","));
116       imprimirPesos();
          Comunicador.addLog("Época Eqm");
118
          try {
            do {
              this.numEpocas++;
122           erroAnterior = erroAtual;
              arq = new FileReader(arquivoTreinamento.getCaminhoCompleto
                 ());
124           lerArq = new BufferedReader(arq);

126           linha = lerArq.readLine();
              if (linha.contains("x1")) {
128             linha = lerArq.readLine();
              }
130
              while (linha != null) {
132             separarEntradas(linha);

134             calcularSaidas();

136             ajustarPesos();

138             linha = lerArq.readLine();
              }
140
              arq.close();
142           erroAtual = erroQuadraticoMedio(arquivoTreinamento);
              Comunicador.addLog(String.format("%d   %.6f", numEpocas,
                 erroAtual).replace(".", ","));
144         } while (Math.abs(erroAtual - erroAnterior) > PRECISAO &&
               numEpocas < 10000);
```

```
146         Comunicador.addLog(String.format("Fim do treinamento. (%.2fs)
               ", (double) (System.currentTimeMillis() - tempInicial) /
               1000D));
            imprimirPesos();
148       } catch (FileNotFoundException ex) {
            return false;
150       } catch (IOException ex) {
            return false;
152       }

154       return true;
      }
156
      public boolean treinarMomentum(Arquivo arquivoTreinamento) {
158       return treinar(arquivoTreinamento, true);
      }
160
      public void testar(Arquivo arquivoTreinamento) {
162       FileReader arq;
          BufferedReader lerArq;
164       String linha;
          String esperada;
166       String resposta;
          boolean errou;
168       int amostrasErradas;
          int numAmostras;
170       double porcAcerto;

172       numAmostras = 0;
          amostrasErradas = 0;
174
          Comunicador.iniciarLog("Início teste da MLP");
176       Comunicador.addLog("d1 d2 d3 -- y1 y2 y3");

178       try {
            arq = new FileReader(arquivoTreinamento.getCaminhoCompleto())
               ;
180         lerArq = new BufferedReader(arq);

182         linha = lerArq.readLine();
            if (linha.contains("x1")) {
```

```
184           linha = lerArq.readLine();
          }
186
          while (linha != null) {
188           numAmostras++;
              separarEntradas(linha);
190
              calcularSaidas();
192
              esperada = "";
194           resposta = "";
              errou = false;
196           for (int i = 0; i < NUM_NEU_CAMADA_SAIDA; i++) {
                esperada += String.format("%.0f   ", saidaEsperada[i]);
198             resposta += String.format("%d   ", posProcessamento(
                   saidaCamadaSaida[i]));

200             if (saidaEsperada[i] != posProcessamento(saidaCamadaSaida
                   [i])) {
                  errou = true;
202             }
              }
204
              if (errou) {
206             amostrasErradas++;
              }
208
              Comunicador.addLog(esperada + " -- " + resposta);
210
              linha = lerArq.readLine();
212       }

214       arq.close();
          porcAcerto = (100D / numAmostras) * ((numAmostras -
             amostrasErradas));
216       Comunicador.addLog(String.format("Total de acertos: %d/%d
             (%.2f%%)", (numAmostras - amostrasErradas), numAmostras,
             porcAcerto));
        } catch (FileNotFoundException ex) {
218     } catch (IOException ex) {
        }
220   }
```

```
222   private double erroQuadraticoMedio(Arquivo arquivo) {
        FileReader arq;
224     BufferedReader lerArq;
        String linha;
226     int numAmostras;
        double erroMedio;
228     double valorParcial;

230     erroMedio = 0D;
        numAmostras = 0;
232
        try {
234       arq = new FileReader(arquivo.getCaminhoCompleto());
          lerArq = new BufferedReader(arq);
236
          linha = lerArq.readLine();
238       if (linha.contains("x1")) {
            linha = lerArq.readLine();
240       }

242       while (linha != null) {
            numAmostras++;
244         separarEntradas(linha);

246         calcularSaidas();

248         //Calculando erro
            valorParcial = 0D;
250         for (int i = 0; i < saidaCamadaSaida.length; i++) {
              valorParcial = valorParcial + Math.pow((double) (
                  saidaEsperada[i] - saidaCamadaSaida[i]), 2D);
252         }
            erroMedio = erroMedio + (valorParcial / 2D);
254
            linha = lerArq.readLine();
256       }

258       arq.close();
          erroMedio = erroMedio / (double) numAmostras;
260
        } catch (FileNotFoundException ex) {
```

```
262       } catch (IOException ex) {
          }
264
          return erroMedio;
266       }

268       private void separarEntradas(String linha) {
            String[] vetor;
270         int i;

272         vetor = linha.split("\\s+");
            i = 0;
274
            if (vetor[0].equals("")) {
276           i = 1;
            }
278
            entradas[0] = -1.0;
280         for (int j = 1; j <= NUM_ENTRADAS; j++) {
              entradas[j] = Double.parseDouble(vetor[i++].replace(",", ".")
                  );
282         }
            for (int j = 0; j < NUM_NEU_CAMADA_SAIDA; j++) {
284           saidaEsperada[j] = Double.parseDouble(vetor[i++].replace(",",
                  "."));
            }
286       }

288       private void calcularSaidas() {
            double valorParcial;
290
            //Calculando saidas da camada escondida
292         saidaCamadaEscondida[0] = -1D;
            potencialCamadaEscondida[0] = -1D;
294
            for (int i = 1; i < saidaCamadaEscondida.length; i++) {
296           valorParcial = 0D;

298           for (int j = 0; j < entradas.length; j++) {
                valorParcial += entradas[j] * pesosCamadaEscondida[i - 1][j
                    ];
300           }
```

```
302         potencialCamadaEscondida[i] = valorParcial;
            saidaCamadaEscondida[i] = funcaoLogistica(valorParcial);
304       }

306       //Calculando saida da camada de saída
          for (int i = 0; i < saidaCamadaSaida.length; i++) {
308         valorParcial = 0D;

310         for (int j = 0; j < saidaCamadaEscondida.length; j++) {
              valorParcial += saidaCamadaEscondida[j] * pesosCamadaSaida[
                 i][j];
312         }

314         potencialCamadaSaida[i] = valorParcial;
            saidaCamadaSaida[i] = funcaoLogistica(valorParcial);
316       }
        }
318
        private void ajustarPesos() {
320       //Ajustando pesos sinapticos da camada de saida
          for (int i = 0; i < gradienteCamadaSaida.length; i++) {
322         gradienteCamadaSaida[i] = (saidaEsperada[i] -
               saidaCamadaSaida[i]) * funcaoLogisticaDerivada(
               potencialCamadaSaida[i]);

324         for (int j = 0; j < NUM_NEU_CAMADA_ESCONDIDA + 1; j++) {
              pesosCamadaSaidaProximo[i][j] = pesosCamadaSaida[i][j] +
326                   (fatorMomentum * (pesosCamadaSaida[i][j] -
                        pesosCamadaSaidaAnterior[i][j])) +
                     (TAXA_APRENDIZAGEM * gradienteCamadaSaida[i] *
                        saidaCamadaEscondida[j]);
328         }
          }
330
          //Ajustando pesos sinapticos da camada escondida
332       for (int i = 0; i < gradienteCamadaEscondida.length; i++) {
            gradienteCamadaEscondida[i] = 0D;
334         for (int j = 0; j < NUM_NEU_CAMADA_SAIDA; j++) {
              gradienteCamadaEscondida[i] += gradienteCamadaSaida[j] *
                 pesosCamadaSaida[j][i + 1];
336         }
```

```
        gradienteCamadaEscondida[i] *= funcaoLogisticaDerivada(
            potencialCamadaEscondida[i + 1]);

        for (int j = 0; j < NUM_ENTRADAS + 1; j++) {
          pesosCamadaEscondidaProximo[i][j] = pesosCamadaEscondida[i
              ][j] +
                  (fatorMomentum * (pesosCamadaEscondida[i][j] -
                      pesosCamadaEscondidaAnterior[i][j])) +
                  (TAXA_APRENDIZAGEM * gradienteCamadaEscondida[i] *
                      entradas[j]);
        }
      }

      //Copiando pesos
      copiarMatriz(pesosCamadaEscondida, pesosCamadaEscondidaAnterior
          );
      copiarMatriz(pesosCamadaEscondidaProximo, pesosCamadaEscondida)
          ;
      copiarMatriz(pesosCamadaSaida, pesosCamadaSaidaAnterior);
      copiarMatriz(pesosCamadaSaidaProximo, pesosCamadaSaida);
    }

    private void copiarMatriz(double[][] origem, double[][] destino)
        {
      for (int i = 0; i < origem.length; i++) {
        for (int j = 0; j < origem[i].length; j++) {
          if (destino.length > i) {
            if (destino[i].length > j) {
              destino[i][j] = origem[i][j];
            }
          }
        }
      }
    }

    private double funcaoLogistica(double valor) {
      return 1D / (1D + Math.pow(Math.E, -1D * BETA * valor));
    }

    private double funcaoLogisticaDerivada(double valor) {
      return (BETA * Math.pow(Math.E, -1D * BETA * valor)) / Math.pow
          ((Math.pow(Math.E, -1D * BETA * valor) + 1D), 2D);
```

```
    }
372
    private int posProcessamento(double valor) {
374   int resposta = 0;

376   if (valor >= 0.5) {
        resposta = 1;
378   }

380   return resposta;
    }
382
    private void imprimirPesos() {
384   String log;

386   Comunicador.addLog("Pesos camada escondida:");

388   for (int i = 0; i < NUM_NEU_CAMADA_ESCONDIDA; i++) {
        log = "N" + (i + 1) + " =";
390
        for (int j = 0; j < NUM_ENTRADAS + 1; j++) {
392       log += String.format(" %f", pesosCamadaEscondida[i][j]);
        }
394
        Comunicador.addLog(log);
396   }

398   Comunicador.addLog("Pesos camada de saída:");
      for (int i = 0; i < NUM_NEU_CAMADA_SAIDA; i++) {
400     log = "N" + (i + 1) + " =";

402     for (int j = 0; j < NUM_NEU_CAMADA_ESCONDIDA + 1; j++) {
          log += String.format(" %f", pesosCamadaSaida[i][j]);
404     }

406     Comunicador.addLog(log);
      }
408 }

410 }
```

ANEXO A - CONJUNTO DE TREINAMENTO

x1	x2	x3	x4	d1	d2	d3
0.3841	0.2021	0	0.2438	1.0000	0	0
0.1765	0.1613	0.3401	0.0843	1.0000	0	0
0.3170	0.5786	0.3387	0.4192	0	1.0000	0
0.2467	0.0337	0.2699	0.3454	1.0000	0	0
0.6102	0.8192	0.4679	0.4762	0	1.0000	0
0.7030	0.7784	0.7482	0.6562	0	0	1.0000
0.4767	0.4348	0.4852	0.3640	0	1.0000	0
0.7589	0.8256	0.6514	0.6143	0	0	1.0000
0.1579	0.3641	0.2551	0.2919	1.0000	0	0
0.5561	0.5602	0.5605	0.2105	0	1.0000	0
0.3267	0.2974	0.0343	0.1466	1.0000	0	0
0.2303	0.0942	0.3889	0.1713	1.0000	0	0
0.2953	0.2963	0.2600	0.3039	1.0000	0	0
0.5797	0.4789	0.5780	0.3048	0	1.0000	0
0.5860	0.5250	0.4792	0.4021	0	1.0000	0
0.7045	0.6933	0.6449	0.6623	0	0	1.0000
0.9134	0.9412	0.6078	0.5934	0	0	1.0000
0.2333	0.4943	0.2525	0.2567	1.0000	0	0
0.2676	0.4172	0.2775	0.2721	1.0000	0	0
0.4850	0.5506	0.5269	0.6036	0	1.0000	0
0.2434	0.2567	0.2312	0.2624	1.0000	0	0
0.1250	0.3023	0.1826	0.3168	1.0000	0	0
0.5598	0.4253	0.4258	0.3192	0	1.0000	0
0.5738	0.7674	0.6154	0.4447	0	0	1.0000
0.5692	0.8368	0.5832	0.4585	0	0	1.0000
0.4655	0.7682	0.3221	0.2940	0	1.0000	0
0.5568	0.7592	0.6293	0.5453	0	1.0000	0
0.8842	0.7509	0.5723	0.5814	0	0	1.0000
0.7959	0.9243	0.7339	0.7334	0	0	1.0000
0.7124	0.7128	0.6065	0.6668	0	0	1.0000
0.6749	0.8767	0.6543	0.7461	0	0	1.0000
0.3674	0.4359	0.4230	0.2965	1.0000	0	0
0.3473	0.0754	0.2183	0.1905	1.0000	0	0
0.6931	0.5188	0.5386	0.5794	0	1.0000	0
0.6439	0.4959	0.4322	0.4582	0	1.0000	0
0.5627	0.4893	0.6831	0.5120	0	1.0000	0

0.5182	0.7553	0.6368	0.4538	0	1.0000	0
0.6046	0.7479	0.6542	0.4375	0	1.0000	0
0.6328	0.6786	0.7751	0.6183	0	0	1.0000
0.3429	0.4694	0.2855	0.2977	1.0000	0	0
0.6371	0.5069	0.5316	0.4520	0	1.0000	0
0.6388	0.6970	0.6407	0.7677	0	0	1.0000
0.3529	0.5504	0.3706	0.4828	0	1.0000	0
0.4302	0.3237	0.6397	0.4319	0	1.0000	0
0.7078	0.9604	0.7470	0.6399	0	0	1.0000
0.7350	0.8170	0.7227	0.6279	0	0	1.0000
0.7011	0.2946	0.6625	0.4312	0	1.0000	0
0.5961	0.3817	0.6363	0.3663	0	1.0000	0
0	0.2563	0.2603	0.3027	1.0000	0	0
0.5996	0.5704	0.6965	0.6548	0	0	1.0000
0.4289	0.3709	0.3994	0.3656	0	1.0000	0
0.2093	0.3655	0.3334	0.1802	1.0000	0	0
0.2335	0.2856	0.3912	0.1601	1.0000	0	0
0.3266	0.7751	0.4356	0.3448	0	1.0000	0
0.2457	0.1203	0.1228	0.2206	1.0000	0	0
0.4656	0.4815	0.4211	0.4862	0	1.0000	0
0.7511	0.8868	0.5408	0.6253	0	0	1.0000
0.7825	0.9386	0.6510	0.6996	0	0	1.0000
0.3463	0.4118	0.2507	0.0454	1.0000	0	0
0.5172	0.1482	0.3172	0.2323	1.0000	0	0
0.6942	0.4516	0.5387	0.5983	0	1.0000	0
0.7586	0.7017	0.7120	0.7509	0	0	1.0000
0.6880	0.6004	0.6602	0.4320	0	1.0000	0
0.4742	0.5079	0.4135	0.4161	0	1.0000	0
0.4419	0.5761	0.4515	0.4497	0	1.0000	0
0.3367	0.4333	0.2336	0.1678	1.0000	0	0
0.4744	0.4604	0.1507	0.4873	1.0000	0	0
0.7510	0.4350	0.5453	0.4831	0	1.0000	0
0.4045	0.5636	0.2534	0.5573	0	1.0000	0
0.1449	0.1539	0.2446	0.0559	1.0000	0	0
0.3460	0.2722	0.1866	0.5049	1.0000	0	0
0.2241	0.2046	0.3575	0.2891	1.0000	0	0
0.1412	0.2264	0.4025	0.2661	1.0000	0	0
0.5782	0.6418	0.7212	0.6396	0	0	1.0000
0.9153	0.6571	0.8229	0.6689	0	0	1.0000
0.6014	0.7664	0.6385	0.5513	0	0	1.0000

0.7328	0.8708	0.8812	0.7060	0	0	1.0000
0.4270	0.6352	0.6811	0.3884	0	1.0000	0
0.6189	0.1652	0.4016	0.3042	1.0000	0	0
0.2143	0.3868	0.1926	0	1.0000	0	0
0.5696	0.7238	0.7199	0.6677	0	0	1.0000
0.8656	0.6700	0.6570	0.6065	0	0	1.0000
0.9002	0.6858	0.7409	0.7047	0	0	1.0000
0.4167	0.5255	0.5506	0.4093	0	1.0000	0
0.8325	0.4804	0.7990	0.7471	0	0	1.0000
0.4124	0.1191	0.4720	0.3184	1.0000	0	0
1.0000	1.0000	0.7924	0.7074	0	0	1.0000
0.5685	0.6924	0.6180	0.5792	0	1.0000	0
0.6505	0.4864	0.2972	0.4599	0	1.0000	0
0.8124	0.7690	0.9720	1.0000	0	0	1.0000
0.9013	0.7160	1.0000	0.8046	0	0	1.0000
0.8872	0.7556	0.9307	0.6791	0	0	1.0000
0.3708	0.2139	0.2136	0.4295	1.0000	0	0
0.5159	0.4349	0.3715	0.4086	0	1.0000	0
0.6768	0.6304	0.8044	0.4885	0	0	1.0000
0.1664	0.2404	0.2000	0.3425	1.0000	0	0
0.2495	0.2807	0.4679	0.2200	1.0000	0	0
0.2487	0.2348	0.0913	0.1281	1.0000	0	0
0.5748	0.8552	0.5973	0.7317	0	0	1.0000
0.3858	0.7585	0.3239	0.3565	0	1.0000	0
0.3329	0.4946	0.5614	0.3152	0	1.0000	0
0.3891	0.4805	0.7598	0.4231	0	1.0000	0
0.2888	0.4888	0.1930	0.0177	1.0000	0	0
0.3827	0.4900	0.2272	0.3599	0	1.0000	0
0.6047	0.4224	0.6274	0.5809	0	1.0000	0
0.9840	0.7031	0.6469	0.4701	0	0	1.0000
0.6554	0.6785	0.9279	0.7723	0	0	1.0000
0.0466	0.3388	0.0840	0.0762	1.0000	0	0
0.6154	0.8196	0.6339	0.7729	0	0	1.0000
0.8452	0.8897	0.8383	0.6961	0	0	1.0000
0.6927	0.7870	0.7689	0.7213	0	0	1.0000
0.4032	0.6188	0.4930	0.5380	0	1.0000	0
0.4006	0.3094	0.3868	0.0811	1.0000	0	0
0.7416	0.7138	0.6823	0.6067	0	0	1.0000
0.7404	0.6764	0.8293	0.4694	0	0	1.0000
0.7736	0.7097	0.6826	0.8142	0	0	1.0000

0.5823	0.9635	0.3706	0.5636	0	1.0000	0
0.2081	0.3738	0.3119	0.3552	1.0000	0	0
0.5616	0.8972	0.5186	0.6650	0	0	1.0000
0.6594	0.8907	0.6000	0.7157	0	0	1.0000
0.3979	0.3070	0.3637	0.1220	1.0000	0	0
0.2644	0	0.3572	0.1931	1.0000	0	0
0.4816	0.4791	0.4213	0.5889	0	1.0000	0
0.0848	0.0749	0.4349	0.3328	1.0000	0	0
0.4608	0.6775	0.3533	0.3016	0	1.0000	0
0.4155	0.6589	0.5310	0.5404	0	1.0000	0
0.3934	0.6244	0.4817	0.4324	0	1.0000	0
0.5843	0.8517	0.8576	0.7133	0	0	1.0000
0.1995	0.3690	0.3537	0.3462	1.0000	0	0
0.3832	0.2321	0.0341	0.2450	1.0000	0	0

ANEXO B - CONJUNTO DE TESTE

x1	x2	x3	x4	d1	d2	d3
0.8622	0.7101	0.6236	0.7894	0	0	1.0000
0.2741	0.1552	0.1333	0.1516	1.0000	0	0
0.6772	0.8516	0.6543	0.7573	0	0	1.0000
0.2178	0.5039	0.6415	0.5039	0	1.0000	0
0.7260	0.7500	0.7007	0.4953	0	0	1.0000
0.2473	0.2941	0.4248	0.3087	1.0000	0	0
0.5682	0.5683	0.5054	0.4426	0	1.0000	0
0.6566	0.6715	0.4952	0.3951	0	1.0000	0
0.0705	0.4717	0.2921	0.2954	1.0000	0	0
0.1187	0.2568	0.3140	0.3037	1.0000	0	0
0.5673	0.7011	0.4083	0.5552	0	1.0000	0
0.3164	0.2251	0.3526	0.2560	1.0000	0	0
0.7884	0.9568	0.6825	0.6398	0	0	1.0000
0.9633	0.7850	0.6777	0.6059	0	0	1.0000
0.7739	0.8505	0.7934	0.6626	0	0	1.0000
0.4219	0.4136	0.1408	0.0940	1.0000	0	0
0.6616	0.4365	0.6597	0.8129	0	0	1.0000
0.7325	0.4761	0.3888	0.5683	0	1.0000	0