# Redes Neurais Artificiais

## Exercício 6 - ELMs com bases de dados reais

Vítor Gabriel Reis Caitité - 2016111849

1/25/2021

# Função que calcula a saída de uma rede ELM

Abaixo está a função que calcula a saída de uma rede ELM.

```
YELM<-function(xin, Z, W, par){
  n<-dim(xin)[2]

  # Adiciona ou não termo de polarização
  if(par == 1) {
    xin<-cbind(1, xin)
  }
  H<-tanh(xin%*%Z)
  y_hat<-sign(H %*% W)
  return(y_hat)
}
```

# Treinamento da rede ELM

A função abaixo é uma implementação possível, em R, para o algoritmo de treinamento de uma rede ELM. Essa função é utilizada na resolução dos exercícios da lista.

```
library("corpcor")

trainELM <- function(xin, yin, p, par){
  n <- dim(xin) [2] # Dimensão da entrada

  #Adiciona ou não o termo de polarização
  if(par == 1){
    xin<-cbind(1,xin)
    Z<-replicate(p, runif(n+1, -0.5, 0.5))
  }
  else{
    Z<-replicate(p, runif(n, -0.5, 0.5))
  }
  H<-tanh(xin %*% Z)

  W<-pseudoinverse(H)%*%yin
  #W<-(solve(t(H) %*% H) %*% t(H)) %*% yin

  return(list(W,H,Z))
}
```

# Função que calcula a resposta de um perceptron simples

Abaixo está a função que calcula a resposta de um perceptron simples e é utilizada nas questões da lista.

```
#Função que calcula a resposta de um perceptron simples.
yperceptron <- function(xvec, w, par){
  # xvec: vetor de entrada
  # w: vetor de pesos
  # par: se adiciona ou não o vetor de 1s na entrada
  # yperceptron: resposta do perceptron
  if ( par==1){
    xvec<-cbind ( 1 , xvec )
  }
  u <- xvec %*% w
  y <- 1.0 * (u>=0)
  return(as.matrix(y))
}
```

# Treinamento do perceptron

A função abaixo é uma implementação possível, em R, para o algoritmo de treinamento do Perceptron. Essa função é utilizada na resolução dos exercícios da lista.

```
trainPerceptron <- function ( xin , yd , eta , tol , maxepocas , par )
{
  dimxin<-dim( xin )
  N <-dimxin[ 1 ]
  n<-dimxin[ 2 ]
  if ( par==1){
    wt<-as.matrix ( runif(n+1) - 0.5)
    xin<-cbind ( 1 , xin )
  } else {
    wt<-as.matrix ( runif ( n ) - 0.5)
  }
  nepocas<-0
  eepoca<-tol + 1

  evec<-matrix ( nrow =1 , ncol=maxepocas )
  while( ( nepocas < maxepocas ) && ( eepoca>tol ) )
  {
    ei2<-0
    xseq<-sample(N)
    for ( i in 1:N)
    {
      irand<-xseq [i]
      yhati<-1.0 * ( ( xin[
        irand , ] %*% wt ) >= 0 )
      ei<-yd[irand]- yhati
      dw<-as.vector(eta) * as.vector(ei) * xin[ irand  , ]
      wt<-wt+dw
      ei2<-ei2 + ei * ei
    }
    nepocas<-nepocas+1
    evec[ nepocas ]<-ei2/N

    eepoca<-evec[nepocas]
  }
  retlist<-list ( wt, evec[ 1:nepocas ] )
  return (retlist)
}
```

# Enunciado Exercício 6

O objetivo dos exercícios desta semana é utilizar as ELMs para resolver problemas multidimensionais, a partir de bases de dados reais. As bases de dados devem ser baixadas do repositório UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets.php (https://archive.ics.uci.edu/ml/datasets.php)). A primeira base de dados a ser estudada é a base Breast Cancer (diagnostic), disponível no link:

https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29 (https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29)

Para esta base, os alunos deverão dividir de forma aleatória os dados entre treinamento e teste e comparar as acurácias de treinamento e teste para diferentes valores do hiperparâmetro que controla o número de neurônios. Os valores de acurácia devem ser apresentados na forma de media ± desvio_padrao para, pelo menos, cinco execuções diferentes.

Algumas perguntas que devem ser respondidas são:

• Com quantos neurônios (aproximadamente) a acurácia de teste aparenta ser máxima?

• O que acontece com os valores de acurácia de treinamento e teste conforme aumentamos progressivamente o número de neurônios (por exemplo, para 5, 10, 30, 50, 100, 300 neurônios)?

O mesmo deve ser feito para a base Statlog (Heart), disponível no link:

https://archive.ics.uci.edu/ml/datasets/Statlog+%28Heart%29 (https://archive.ics.uci.edu/ml/datasets/Statlog+%28Heart%29)

Além das Extreme Learning Machines, os alunos deverão, também, treinar, utilizando a rotina desenvolvida para as atividades anteriores, um perceptron, e avaliar seu desempenho na solução dos dois problemas, comparado às ELMs.

Por questões de convergência, pode ser necessário escalonar os valores dos atributos para que fiquem restritos entre 0 e 1. Para tanto, uma possibilidade é utilizar a forma abaixo:

$z_i = [x_i - \min(x)]/[\max(x) - \min(x)]$

# ELM com base de dados Breast Cancer (diagnostic)

Ulizando a base de dados Breast Cancer (diagnostic) foi desenvolvida uma ELM para classificar um tumor em maligno ou benigno. Essa base de dados é composta de 32 atributos, sendo eles: ID, diagnóstico (“M”-maligno, ou “B”-benigno) e 30 características de entrada com valores reais. Foram utilizados 70% dos dados para treino e 30% para teste. Além disso, variou-se progressivamente o número de neurônios da seguinte forma, 5, 10, 30, 50, 100, 200 e 300 neurônios. Para cada número de neurônios foram realizados 20 execuções diferentes de treinamento e teste, e os valores de acurácia, para cada caso, foram apresentados na forma de média ± desvio_padrão. Os resultados e o script desenvolvido podem ser vistos abaixo.

```
rm(list=ls())
source("~/Documents/UFMG/9/Redes Neurais/exemplos/trainELM.R")
source("~/Documents/UFMG/9/Redes Neurais/exemplos/YELM.R")
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
# Carregando base de dados:
path <- file.path("~/Documents/UFMG/9/Redes Neurais/listas/lista 6/cancer", "wdbc.cs
v")
data <- read.csv(path)

# Separando dados de entrada e saída:
x_all <- as.matrix(data[1:569, 3:32])
class <- as.matrix(data[1:569, 2])
y_all <- rep(0,569)
for (count in 1:length(class)) {
  if (class[count] == 'M' ){
    y_all[count] = -1
  }
  else if(class[count] == 'B'){
    y_all[count] = 1
  }
}

for (p in c(5,10,30,50,100,200,300)){
  # Realiza pelo 20 execuções diferentes
  accuracy_train <- rep(0, 20)
  accuracy_test <- rep(0, 20)
  for(execution in 1:20){
    # Separando dados entre treino e teste aleatoriamente:
    positions_train <- createDataPartition(1:569,p=.7)
    length_train <- length(positions_train$Resample1)
    length_test <- length(y_all) - length_train
    x_train <- matrix(rep(0, 30*length_train), ncol=30, nrow=length_train)
    y_train <- rep(0, length_train)
    x_test <- matrix(rep(0, (30*length_test)), ncol=30, nrow=(length(y_all) - length_
train))
    y_test <- rep(0, (length(y_all) - length_train))
    index_train <- 1
    index_test <- 1
    for (count in 1:length(y_all)) {
      if (index_train <= length_train && count == positions_train$Resample1[index_tra
in]){
        x_train[index_train, ] <- x_all[count, 1:30 ]
        y_train[index_train] <- y_all[count]
        index_train = index_train + 1
      } else {
        x_test[index_test, ] <- x_all[count, 1:30 ]
        y_test[index_test] <- y_all[count]
        index_test = index_test + 1
      }
    }

    # Treinando modelo:
    retlist<-trainELM(x_train, y_train, p, 1)
    W<-retlist[[1]]
    H<-retlist[[2]]
    Z<-retlist[[3]]

    # Calculando acurácia de treinamento
    y_hat_train <- as.matrix(YELM(x_train, Z, W, 1), nrow = length_train, ncol = 1)
    accuracy_train[execution]<-((sum(abs(y_hat_train + y_train)))/2)/length_train
    #print(paste("Acurácia de treinamento para execução", execution, "com", p, "nerôn
ios:", accuracy_train))

    # Calculando acurácia de Teste:
    y_hat_test <- as.matrix(YELM(x_test, Z, W, 1), nrow = length_test, ncol = 1)
    accuracy_test[execution]<-((sum(abs(y_hat_test + y_test)))/2)/length_test
    #print(paste("Acurácia de teste para execução", execution, "com", p, "nerônios:",
accuracy_test))
  }
```

```
  # Média das acurácias
  mean_accuracy_train <- mean(accuracy_train) * 100
  mean_accuracy_test <- mean(accuracy_test) * 100

  # Desvio Padrão das acurácias
  sd_accuracy_train <- sd(accuracy_train) * 100
  sd_accuracy_test <- sd(accuracy_test) * 100

  print(paste("Acurácia de treinamento do modelo com", p, "neurônios:", mean_accuracy
_train, "%", "±", sd_accuracy_train, "%"))
  print(paste("Acurácia de teste do modelo com", p, "neurônios:", mean_accuracy_test,
"%", "±", sd_accuracy_test, "%"))
}
```

```
## [1] "Acurácia de treinamento do modelo com 5 neurônios: 62.8678304239401 % ± 3.081
85758019101 %"
## [1] "Acurácia de teste do modelo com 5 neurônios: 60.3273809523809 % ± 5.268604898
57386 %"
## [1] "Acurácia de treinamento do modelo com 10 neurônios: 66.3092269326683 % ± 3.64
79748258944 %"
## [1] "Acurácia de teste do modelo com 10 neurônios: 64.1666666666667 % ± 5.30150726
056268 %"
## [1] "Acurácia de treinamento do modelo com 30 neurônios: 73.7905236907731 % ± 3.81
292556166834 %"
## [1] "Acurácia de teste do modelo com 30 neurônios: 70.2678571428572 % ± 6.82094960
855727 %"
## [1] "Acurácia de treinamento do modelo com 50 neurônios: 77.4563591022444 % ± 3.13
398816775674 %"
## [1] "Acurácia de teste do modelo com 50 neurônios: 72.6488095238095 % ± 4.54336805
473881 %"
## [1] "Acurácia de treinamento do modelo com 100 neurônios: 83.3416458852868 % ± 2.4
5087141837332 %"
## [1] "Acurácia de teste do modelo com 100 neurônios: 74.9404761904762 % ± 3.2135894
5396903 %"
## [1] "Acurácia de treinamento do modelo com 200 neurônios: 90.1496259351621 % ± 1.7
3434935281317 %"
## [1] "Acurácia de teste do modelo com 200 neurônios: 73.1845238095238 % ± 3.5529755
1079485 %"
## [1] "Acurácia de treinamento do modelo com 300 neurônios: 93.2668329177057 % ± 1.3
6349325772102 %"
## [1] "Acurácia de teste do modelo com 300 neurônios: 70.327380952381 % ± 4.30652753
813173 %"
```

### *Discussão:*

Pôde-se perceber que a acurácia de treinamento também vai aumentando progressivamente a medida que aumenta-se o número de neurônios. Contudo, o mesmo não ocorre para a acurácia de teste. Uma explicação plausível para isso é que um modelo com um número muito elevado de neurônios pode se tornar super ajustado aos dados de treinamento e não funcionar da maneira esperada com dados de teste. Aparentemente, a acurácia média máxima de teste foi obtida para o número de 100 neurônios.

# Perceptron simples com base de dados Breast Cancer (diagnostic)

Nessa etapa foi treinado um perceptron simples, utilizando a rotina de treinamento de perceptron mostrada no início desse documento e a base de dados da questão anterior, para também classificar um tumor em maligno ou benigno. Foram utilizados 70% dos dados para treino e 30% para teste. Além disso, foram realizados 20 execuções diferentes de treinamento e teste, e os valores de acurácia (de treinamento e teste), foram apresentados na forma de média ± desvio_padrão. Os resultados e o script desenvolvido podem ser vistos abaixo.

```
rm(list=ls())
source("~/Documents/UFMG/9/Redes Neurais/listas/lista 4/trainPerceptron.R")
source("~/Documents/UFMG/9/Redes Neurais/listas/lista 4/yperceptron.R")
library(caret)

# Carregando base de dados:
path <- file.path("~/Documents/UFMG/9/Redes Neurais/listas/lista 6/cancer", "wdbc.cs
v")
data <- read.csv(path)

# Separando dados de entrada e saída:
x_all <- as.matrix(data[1:569, 3:32])
class <- as.matrix(data[1:569, 2])
y_all <- rep(0,569)
for (count in 1:length(class)) {
  if (class[count] == 'M' ){
    y_all[count] = 0
  }
  else if(class[count] == 'B'){
    y_all[count] = 1
  }
}

# Realiza pelo 20 execuções diferentes
accuracy_train <- rep(0, 20)
accuracy_test <- rep(0, 20)
for(execution in 1:20){
  # Separando dados entre treino e teste aleatoriamente:
  positions_train <- createDataPartition(1:569,p=.7)
  length_train <- length(positions_train$Resample1)
  length_test <- length(y_all) - length_train
  x_train <- matrix(rep(0, 30*length_train), ncol=30, nrow=length_train)
  y_train <- rep(0, length_train)
  x_test <- matrix(rep(0, (30*length_test)), ncol=30, nrow=(length(y_all) - length_tr
ain))
  y_test <- rep(0, (length(y_all) - length_train))
  index_train <- 1
  index_test <- 1
  for (count in 1:length(y_all)) {
    if (index_train <= length_train && count == positions_train$Resample1[index_trai
n]){
      x_train[index_train, ] <- x_all[count, 1:30 ]
      y_train[index_train] <- y_all[count]
      index_train = index_train + 1
    } else {
      x_test[index_test, ] <- x_all[count, 1:30 ]
      y_test[index_test] <- y_all[count]
      index_test = index_test + 1
    }
  }

  # Treinando modelo:
  retlist<-trainPerceptron(x_train, y_train, 0.1, 0.01, 1000, 1)
  W<-retlist[[1]]

  # Calculando acurácia de treinamento
  y_hat_train <- as.matrix(yperceptron(x_train, W, 1), nrow = length_train, ncol = 1)
  accuracy_train[execution]<-1-((t(y_hat_train-y_train) %*% (y_hat_train-y_train))/le
ngth_train)
  #print(paste("Acurácia de treinamento para execução", execution, "com", p, "nerônio
s:", accuracy_train))

  # Calculando acurácia de Teste:
  y_hat_test <- as.matrix(yperceptron(x_test, W, 1), nrow = length_test, ncol = 1)
  accuracy_test[execution]<-1-((t(y_hat_test-y_test) %*% (y_hat_test-y_test))/length_
```

```
test)
  #print(paste("Acurácia de teste para execução", execution, "com", p, "nerônios:", a
ccuracy_test))
}
# Média das acurácias
mean_accuracy_train <- mean(accuracy_train) * 100
mean_accuracy_test <- mean(accuracy_test) * 100

# Desvio Padrão das acurácias
sd_accuracy_train <- sd(accuracy_train) * 100
sd_accuracy_test <- sd(accuracy_test) * 100

print(paste("Acurácia de treinamento do modelo com perceptron simples", mean_accuracy
_train, "%", "±", sd_accuracy_train, "%"))
```

```
## [1] "Acurácia de treinamento do modelo com perceptron simples 90.3491271820449 % ±
3.87356283541417 %"
```

```
print(paste("Acurácia de teste do modelo com perceptron simples", mean_accuracy_test,
"%", "±", sd_accuracy_test, "%"))
```

```
## [1] "Acurácia de teste do modelo com perceptron simples 88.7202380952381 % ± 4.057
71904406327 %"
```

### *Discussão:*

Pôde-se perceber que obteve-se uma acurácia de treinamento de cerca de 90%, assim como para a ELM com 100 neurônios. Contudo, obteve-se uma acurácia de teste também próxima dos 90%, o que é significativamente melhor do que a obtida para as ELMs.

# ELM com base de dados Statlog (Heart)

Ulizando a base de dados Statlog (Heart), foi desenvolvida uma ELM para indicar a presença ou ausência de problemas de coração, com base em uma série de caracteísticas. Essa base de dados é composta de 13 caracteristicas (features) e a variável de predição (que assume os valores: 1 - ausência ou 2 - presença de doença no coração). Foram utilizados 70% dos dados para treino e 30% para teste. Além disso, variou-se progressivamente o número de neurônios da seguinte forma, 5, 10, 30, 50, 100, 200 e 300 neurônios. Para cada número de neurônios foram realizados 20 execuções diferentes de treinamento e teste, e os valores de acurácia, para cada caso, foram apresentados na forma de média ± desvio_padrão. Os resultados e o script desenvolvido podem ser vistos abaixo.

```
rm(list=ls())
source("~/Documents/UFMG/9/Redes Neurais/exemplos/trainELM.R")
source("~/Documents/UFMG/9/Redes Neurais/exemplos/YELM.R")
library(caret)

# Carregando base de dados:
path <- file.path("~/Documents/UFMG/9/Redes Neurais/listas/lista 6/heart", "heart.cs
v")
data <- read.csv(path)

# Separando dados de entrada e saída:
x_all <- as.matrix(data[1:270, 1:13])
class <- as.matrix(data[1:270, 14])
y_all <- rep(0,270)
for (count in 1:length(class)) {
  if (class[count] == 2 ){
    y_all[count] = -1
  }
  else if(class[count] == 1){
    y_all[count] = 1
  }
}

for (p in c(5,10,30,50,100, 200, 300)){
  # Realiza pelo 20 execuções diferentes
  accuracy_train <- rep(0, 20)
  accuracy_test <- rep(0, 20)
  for(execution in 1:20){
    # Separando dados entre treino e teste aleatoriamente:
    positions_train <- createDataPartition(1:270,p=.7)
    length_train <- length(positions_train$Resample1)
    length_test <- length(y_all) - length_train
    x_train <- matrix(rep(0, 13*length_train), ncol=13, nrow=length_train)
    y_train <- rep(0, length_train)
    x_test <- matrix(rep(0, (13*length_test)), ncol=13, nrow=(length(y_all) - length_
train))
    y_test <- rep(0, (length(y_all) - length_train))
    index_train <- 1
    index_test <- 1
    for (count in 1:length(y_all)) {
      if (index_train <= length_train && count == positions_train$Resample1[index_tra
in]){
        x_train[index_train, ] <- x_all[count, 1:13]
        y_train[index_train] <- y_all[count]
        index_train = index_train + 1
      } else {
        x_test[index_test, ] <- x_all[count, 1:13]
        y_test[index_test] <- y_all[count]
        index_test = index_test + 1
      }
    }

    # Treinando modelo:
    retlist<-trainELM(x_train, y_train, p, 1)
    W<-retlist[[1]]
    H<-retlist[[2]]
    Z<-retlist[[3]]

    # Calculando acurácia de treinamento
    y_hat_train <- as.matrix(YELM(x_train, Z, W, 1), nrow = length_train, ncol = 1)
    accuracy_train[execution]<-((sum(abs(y_hat_train + y_train)))/2)/length_train
    #print(paste("Acurácia de treinamento para execução", execution, "com", p, "nerôn
ios:", accuracy_train))

    # Calculando acurácia de Teste:
```

```
    y_hat_test <- as.matrix(YELM(x_test, Z, W, 1), nrow = length_test, ncol = 1)
    accuracy_test[execution]<-((sum(abs(y_hat_test + y_test)))/2)/length_test
    #print(paste("Acurácia de teste para execução", execution, "com", p, "nerônios:",
accuracy_test))
  }
  # Média das acurácias
  mean_accuracy_train <- mean(accuracy_train) * 100
  mean_accuracy_test <- mean(accuracy_test) * 100

  # Desvio Padrão das acurácias
  sd_accuracy_train <- sd(accuracy_train) * 100
  sd_accuracy_test <- sd(accuracy_test) * 100

  print(paste("Acurácia de treinamento do modelo com", p, "neurônios:", mean_accuracy
_train, "%", "±", sd_accuracy_train, "%"))
  print(paste("Acurácia de teste do modelo com", p, "neurônios:", mean_accuracy_test,
"%", "±", sd_accuracy_test, "%"))
}
```

```
## [1] "Acurácia de treinamento do modelo com 5 neurônios: 58.3684210526316 % ± 3.175
84923603511 %"
## [1] "Acurácia de teste do modelo com 5 neurônios: 56.1875 % ± 6.44402773279667 %"
## [1] "Acurácia de treinamento do modelo com 10 neurônios: 60.5 % ± 5.09273958699516
%"
## [1] "Acurácia de teste do modelo com 10 neurônios: 55.625 % ± 5.92580218864479 %"
## [1] "Acurácia de treinamento do modelo com 30 neurônios: 65.5263157894737 % ± 4.07
503610170615 %"
## [1] "Acurácia de teste do modelo com 30 neurônios: 60.3125 % ± 4.01221408228739 %"
## [1] "Acurácia de treinamento do modelo com 50 neurônios: 68.8684210526316 % ± 2.69
602527196878 %"
## [1] "Acurácia de teste do modelo com 50 neurônios: 64.875 % ± 6.88127703876327 %"
## [1] "Acurácia de treinamento do modelo com 100 neurônios: 74.1578947368421 % ± 2.9
2243110894485 %"
## [1] "Acurácia de teste do modelo com 100 neurônios: 63.8125 % ± 6.7555410979666 %"
## [1] "Acurácia de treinamento do modelo com 200 neurônios: 80.9473684210526 % ± 3.2
1554033790188 %"
## [1] "Acurácia de teste do modelo com 200 neurônios: 62.25 % ± 5.0425818366138 %"
## [1] "Acurácia de treinamento do modelo com 300 neurônios: 87.5526315789474 % ± 2.4
5843119653518 %"
## [1] "Acurácia de teste do modelo com 300 neurônios: 60.1875 % ± 5.27648395190745
%"
```

### *Discussão:*

Novamente pôde-se perceber que a acurácia de treinamento vai aumentando progressivamente a medida que aumenta-se o número de neurônios. Contudo, o mesmo não ocorre para a acurácia de teste. Como citado, uma explicação plausível para isso é que um modelo com um número muito elevado de neurônios pode se tornar super ajustado aos dados de treinamento e não funcionar da maneira esperada com dados de teste. Aparentemente, a acurácia média máxima de teste foi obtida para o número de 50 neurônios, um valor menor que o número de neurônios para acurácia máxima da base de dados anterior. Isso, pode ser explicado pela diferença de complexidade dos problemas, enquanto na classificação do tumor verificou-se um número de 30 features, nesse caso verificou-se apenas 13.

# Perceptron simples com base de dados Statlog (Heart)

Nessa etapa foi treinado um perceptron simples, utilizando a rotina de treinamento de perceptron mostrada no início desse documento e a base de dados Statlog (Heart), para também indicar a presença ou ausência de problemas de coração. Foram utilizados 70% dos dados para treino e 30% para teste. Além disso, foram realizados 20 execuções diferentes de treinamento e teste, e os valores de acurácia (de treinamento e teste), foram apresentados na forma de média ± desvio_padrão. Os resultados e o script desenvolvido podem ser vistos abaixo.

```
rm(list=ls())
source("~/Documents/UFMG/9/Redes Neurais/listas/lista 4/trainPerceptron.R")
source("~/Documents/UFMG/9/Redes Neurais/listas/lista 4/yperceptron.R")
library(caret)

# Carregando base de dados:
path <- file.path("~/Documents/UFMG/9/Redes Neurais/listas/lista 6/heart", "heart.cs
v")
data <- read.csv(path)

# Separando dados de entrada e saída:
x_all <- as.matrix(data[1:270, 1:13])
class <- as.matrix(data[1:270, 14])
y_all <- rep(0,270)
for (count in 1:length(class)) {
  if (class[count] == 1 ){
    y_all[count] = 1
  }
  else if(class[count] == 2){
    y_all[count] = 0
  }
}

# Realiza pelo 20 execuções diferentes
accuracy_train <- rep(0, 20)
accuracy_test <- rep(0, 20)
for(execution in 1:20){
  # Separando dados entre treino e teste aleatoriamente:
  positions_train <- createDataPartition(1:270,p=.7)
  length_train <- length(positions_train$Resample1)
  length_test <- length(y_all) - length_train
  x_train <- matrix(rep(0, 13*length_train), ncol=13, nrow=length_train)
  y_train <- rep(0, length_train)
  x_test <- matrix(rep(0, (13*length_test)), ncol=13, nrow=(length(y_all) - length_tr
ain))
  y_test <- rep(0, (length(y_all) - length_train))
  index_train <- 1
  index_test <- 1
  for (count in 1:length(y_all)) {
    if (index_train <= length_train && count == positions_train$Resample1[index_trai
n]){
      x_train[index_train, ] <- x_all[count, 1:13]
      y_train[index_train] <- y_all[count]
      index_train = index_train + 1
    } else {
      x_test[index_test, ] <- x_all[count, 1:13]
      y_test[index_test] <- y_all[count]
      index_test = index_test + 1
    }
  }

  # Treinando modelo:
  retlist<-trainPerceptron(x_train, y_train, 0.1, 0.01, 1000, 1)
  W<-retlist[[1]]

  # Calculando acurácia de treinamento
  y_hat_train <- as.matrix(yperceptron(x_train, W, 1), nrow = length_train, ncol = 1)
  accuracy_train[execution]<-1-((t(y_hat_train-y_train) %*% (y_hat_train-y_train))/le
ngth_train)
  #print(paste("Acurácia de treinamento para execução", execution, "com", p, "nerônio
s:", accuracy_train))

  # Calculando acurácia de Teste:
  y_hat_test <- as.matrix(yperceptron(x_test, W, 1), nrow = length_test, ncol = 1)
  accuracy_test[execution]<-1-((t(y_hat_test-y_test) %*% (y_hat_test-y_test))/length_
```

```
test)
  #print(paste("Acurácia de teste para execução", execution, "com", p, "nerônios:", a
ccuracy_test))
}
# Média das acurácias
mean_accuracy_train <- mean(accuracy_train) * 100
mean_accuracy_test <- mean(accuracy_test) * 100

# Desvio Padrão das acurácias
sd_accuracy_train <- sd(accuracy_train) * 100
sd_accuracy_test <- sd(accuracy_test) * 100

print(paste("Acurácia de treinamento do modelo com perceptron simples", mean_accuracy
_train, "%", "±", sd_accuracy_train, "%"))
```

```
## [1] "Acurácia de treinamento do modelo com perceptron simples 70.2631578947368 % ±
14.0506031507283 %"
```

```
print(paste("Acurácia de teste do modelo com perceptron simples", mean_accuracy_test,
"%", "±", sd_accuracy_test, "%"))
```

```
## [1] "Acurácia de teste do modelo com perceptron simples 70.25 % ± 13.6966669014956
%"
```

### *Discussão:*

Pôde-se perceber que obteve-se uma acurácia de treinamento de cerca de 75%, assim como para a ELM com 100 neurônios. Contudo, obteve-se uma acurácia de teste média também próxima dos 75%, o que é melhor do que a obtida para as ELMs (que no melhor caso obteve uma acurácia média de 65%).

# Testes com dados escalonados

Como sugerido pelo enunciado, por questões de convergência, pode ser interessante escalonar os valores dos atributos para que fiquem restritos entre 0 e 1. Isso foi feito para todos os exercícios anteriores, utilizando a função abaixo desenvolvida pelo autor. Os resultados para cada um dos testes podem ser vistos nas subseções seguintes.

```
# Função que recebe uma matriz e suas dimensões e retorna uma matriz
# de mesma dimensão porém com sua colunas escalonadas.
staggeringMatrix <- function(matrix, rows, columns ){
  staggeredMatrix <- matrix(rep(0, rows*columns), ncol = columns, nrow = rows)
  # Escalonando dados:
  for (j in 1:columns) {
    for (i in 1:rows) {
      staggeredMatrix[i,j] <- (matrix[i,j] - min(matrix[,j])) / (max(matrix[,j]) - mi
n(matrix[,j]))
    }
  }
  return(staggeredMatrix)
}
```

### *ELM com base de dados Breast Cancer (diagnostic) - Dados escalonados*

```
rm(list=ls())
source("~/Documents/UFMG/9/Redes Neurais/exemplos/trainELM.R")
source("~/Documents/UFMG/9/Redes Neurais/exemplos/YELM.R")
source("~/Documents/UFMG/9/Redes Neurais/exemplos/escalonamento_matrix.R")
library(caret)

# Carregando base de dados:
path <- file.path("~/Documents/UFMG/9/Redes Neurais/listas/lista 6/cancer", "wdbc.cs
v")
data <- read.csv(path)

# Separando dados de entrada e saída:
x_all <- as.matrix(data[1:569, 3:32])
class <- as.matrix(data[1:569, 2])
y_all <- rep(0,569)
for (count in 1:length(class)) {
  if (class[count] == 'M' ){
    y_all[count] = -1
  }
  else if(class[count] == 'B'){
    y_all[count] = 1
  }
}

# Escalonando os valores dos atributos para que fiquem restritos entre 0 e 1
x_all <- staggeringMatrix(x_all, nrow(x_all), ncol(x_all))

for (p in c(5,10,30,50,100,200,300)){
  # Realiza pelo 20 execuções diferentes
  accuracy_train <- rep(0, 20)
  accuracy_test <- rep(0, 20)
  for(execution in 1:20){
    # Separando dados entre treino e teste aleatoriamente:
    positions_train <- createDataPartition(1:569,p=.7)
    length_train <- length(positions_train$Resample1)
    length_test <- length(y_all) - length_train
    x_train <- matrix(rep(0, 30*length_train), ncol=30, nrow=length_train)
    y_train <- rep(0, length_train)
    x_test <- matrix(rep(0, (30*length_test)), ncol=30, nrow=(length(y_all) - length_
train))
    y_test <- rep(0, (length(y_all) - length_train))
    index_train <- 1
    index_test <- 1
    for (count in 1:length(y_all)) {
      if (index_train <= length_train && count == positions_train$Resample1[index_tra
in]){
        x_train[index_train, ] <- x_all[count, 1:30 ]
        y_train[index_train] <- y_all[count]
        index_train = index_train + 1
      } else {
        x_test[index_test, ] <- x_all[count, 1:30 ]
        y_test[index_test] <- y_all[count]
        index_test = index_test + 1
      }
    }

    # Treinando modelo:
    retlist<-trainELM(x_train, y_train, p, 1)
    W<-retlist[[1]]
    H<-retlist[[2]]
    Z<-retlist[[3]]

    # Calculando acurácia de treinamento
    y_hat_train <- as.matrix(YELM(x_train, Z, W, 1), nrow = length_train, ncol = 1)
    accuracy_train[execution]<-((sum(abs(y_hat_train + y_train)))/2)/length_train
```

```
    #print(paste("Acurácia de treinamento para execução", execution, "com", p, "nerôn
ios:", accuracy_train))

    # Calculando acurácia de Teste:
    y_hat_test <- as.matrix(YELM(x_test, Z, W, 1), nrow = length_test, ncol = 1)
    accuracy_test[execution]<-((sum(abs(y_hat_test + y_test)))/2)/length_test
    #print(paste("Acurácia de teste para execução", execution, "com", p, "nerônios:",
accuracy_test))
  }
  # Média das acurácias
  mean_accuracy_train <- mean(accuracy_train) * 100
  mean_accuracy_test <- mean(accuracy_test) * 100

  # Desvio Padrão das acurácias
  sd_accuracy_train <- sd(accuracy_train) * 100
  sd_accuracy_test <- sd(accuracy_test) * 100

  print(paste("Acurácia de treinamento do modelo com", p, "neurônios:", mean_accuracy
_train, "%", "±", sd_accuracy_train, "%"))
  print(paste("Acurácia de teste do modelo com", p, "neurônios:", mean_accuracy_test,
"%", "±", sd_accuracy_test, "%"))
}
```

```
## [1] "Acurácia de treinamento do modelo com 5 neurônios: 77.5436408977556 % ± 7.179
67241078705 %"
## [1] "Acurácia de teste do modelo com 5 neurônios: 77.1428571428572 % ± 5.915928482
51565 %"
## [1] "Acurácia de treinamento do modelo com 10 neurônios: 85.3366583541147 % ± 3.09
425861860818 %"
## [1] "Acurácia de teste do modelo com 10 neurônios: 84.5238095238095 % ± 4.42916426
894008 %"
## [1] "Acurácia de treinamento do modelo com 30 neurônios: 94.6259351620948 % ± 0.92
4181940096817 %"
## [1] "Acurácia de teste do modelo com 30 neurônios: 92.2619047619048 % ± 2.48068081
388836 %"
## [1] "Acurácia de treinamento do modelo com 50 neurônios: 95.6857855361596 % ± 0.47
246649317557 %"
## [1] "Acurácia de teste do modelo com 50 neurônios: 94.7916666666667 % ± 1.65988370
908215 %"
## [1] "Acurácia de treinamento do modelo com 100 neurônios: 97.2069825436409 % ± 0.4
82746155385031 %"
## [1] "Acurácia de teste do modelo com 100 neurônios: 91.6964285714286 % ± 2.1804263
7110366 %"
## [1] "Acurácia de treinamento do modelo com 200 neurônios: 98.7531172069825 % ± 0.5
11709901289355 %"
## [1] "Acurácia de teste do modelo com 200 neurônios: 83.3630952380952 % ± 3.3753339
8647143 %"
## [1] "Acurácia de treinamento do modelo com 300 neurônios: 99.8379052369077 % ± 0.2
02675237128989 %"
## [1] "Acurácia de teste do modelo com 300 neurônios: 72.3809523809524 % ± 4.2195887
2274769 %"
```

### *Perceptron simples com base de dados Breast Cancer (diagnostic) - Dados escalonados*

```
rm(list=ls())
source("~/Documents/UFMG/9/Redes Neurais/listas/lista 4/trainPerceptron.R")
source("~/Documents/UFMG/9/Redes Neurais/listas/lista 4/yperceptron.R")
source("~/Documents/UFMG/9/Redes Neurais/exemplos/escalonamento_matrix.R")
library(caret)

# Carregando base de dados:
path <- file.path("~/Documents/UFMG/9/Redes Neurais/listas/lista 6/cancer", "wdbc.cs
v")
data <- read.csv(path)

# Separando dados de entrada e saída:
x_all <- as.matrix(data[1:569, 3:32])
class <- as.matrix(data[1:569, 2])
y_all <- rep(0,569)
for (count in 1:length(class)) {
  if (class[count] == 'M' ){
    y_all[count] = 0
  }
  else if(class[count] == 'B'){
    y_all[count] = 1
  }
}

# Escalonando os valores dos atributos para que fiquem restritos entre 0 e 1
x_all <- staggeringMatrix(x_all, nrow(x_all), ncol(x_all))


# Realiza pelo 20 execuções diferentes
accuracy_train <- rep(0, 20)
accuracy_test <- rep(0, 20)
for(execution in 1:20){
  # Separando dados entre treino e teste aleatoriamente:
  positions_train <- createDataPartition(1:569,p=.7)
  length_train <- length(positions_train$Resample1)
  length_test <- length(y_all) - length_train
  x_train <- matrix(rep(0, 30*length_train), ncol=30, nrow=length_train)
  y_train <- rep(0, length_train)
  x_test <- matrix(rep(0, (30*length_test)), ncol=30, nrow=(length(y_all) - length_tr
ain))
  y_test <- rep(0, (length(y_all) - length_train))
  index_train <- 1
  index_test <- 1
  for (count in 1:length(y_all)) {
    if (index_train <= length_train && count == positions_train$Resample1[index_trai
n]){
      x_train[index_train, ] <- x_all[count, 1:30 ]
      y_train[index_train] <- y_all[count]
      index_train = index_train + 1
    } else {
      x_test[index_test, ] <- x_all[count, 1:30 ]
      y_test[index_test] <- y_all[count]
      index_test = index_test + 1
    }
  }

  # Treinando modelo:
  retlist<-trainPerceptron(x_train, y_train, 0.1, 0.01, 1000, 1)
  W<-retlist[[1]]

  # Calculando acurácia de treinamento
  y_hat_train <- as.matrix(yperceptron(x_train, W, 1), nrow = length_train, ncol = 1)
  accuracy_train[execution]<-1-((t(y_hat_train-y_train) %*% (y_hat_train-y_train))/le
ngth_train)
  #print(paste("Acurácia de treinamento para execução", execution, "com", p, "nerônio
```

```
s:", accuracy_train))

  # Calculando acurácia de Teste:
  y_hat_test <- as.matrix(yperceptron(x_test, W, 1), nrow = length_test, ncol = 1)
  accuracy_test[execution]<-1-((t(y_hat_test-y_test) %*% (y_hat_test-y_test))/length_
test)
  #print(paste("Acurácia de teste para execução", execution, "com", p, "nerônios:", a
ccuracy_test))
}
# Média das acurácias
mean_accuracy_train <- mean(accuracy_train) * 100
mean_accuracy_test <- mean(accuracy_test) * 100

# Desvio Padrão das acurácias
sd_accuracy_train <- sd(accuracy_train) * 100
sd_accuracy_test <- sd(accuracy_test) * 100

print(paste("Acurácia de treinamento do modelo com perceptron simples", mean_accuracy
_train, "%", "±", sd_accuracy_train, "%"))
```

```
## [1] "Acurácia de treinamento do modelo com perceptron simples 96.645885286783 % ±
2.64717156616714 %"
```

```
print(paste("Acurácia de teste do modelo com perceptron simples", mean_accuracy_test,
"%", "±", sd_accuracy_test, "%"))
```

```
## [1] "Acurácia de teste do modelo com perceptron simples 94.1071428571429 % ± 2.114
65005102173 %"
```

### *ELM com base de dados Statlog (Heart) - Dados escalonados*

```
rm(list=ls())
source("~/Documents/UFMG/9/Redes Neurais/exemplos/trainELM.R")
source("~/Documents/UFMG/9/Redes Neurais/exemplos/YELM.R")
source("~/Documents/UFMG/9/Redes Neurais/exemplos/escalonamento_matrix.R")
library(caret)

# Carregando base de dados:
path <- file.path("~/Documents/UFMG/9/Redes Neurais/listas/lista 6/heart", "heart.cs
v")
data <- read.csv(path)

# Separando dados de entrada e saída:
x_all <- as.matrix(data[1:270, 1:13])
class <- as.matrix(data[1:270, 14])
y_all <- rep(0,270)
for (count in 1:length(class)) {
  if (class[count] == 2 ){
    y_all[count] = -1
  }
  else if(class[count] == 1){
    y_all[count] = 1
  }
}

# Escalonando os valores dos atributos para que fiquem restritos entre 0 e 1
x_all <- staggeringMatrix(x_all, nrow(x_all), ncol(x_all))

for (p in c(5,10,30,50,100,200,300)){
  # Realiza pelo 20 execuções diferentes
  accuracy_train <- rep(0, 20)
  accuracy_test <- rep(0, 20)
  for(execution in 1:20){
    # Separando dados entre treino e teste aleatoriamente:
    positions_train <- createDataPartition(1:270,p=.7)
    length_train <- length(positions_train$Resample1)
    length_test <- length(y_all) - length_train
    x_train <- matrix(rep(0, 13*length_train), ncol=13, nrow=length_train)
    y_train <- rep(0, length_train)
    x_test <- matrix(rep(0, (13*length_test)), ncol=13, nrow=(length(y_all) - length_
train))
    y_test <- rep(0, (length(y_all) - length_train))
    index_train <- 1
    index_test <- 1
    for (count in 1:length(y_all)) {
      if (index_train <= length_train && count == positions_train$Resample1[index_tra
in]){
        x_train[index_train, ] <- x_all[count, 1:13]
        y_train[index_train] <- y_all[count]
        index_train = index_train + 1
      } else {
        x_test[index_test, ] <- x_all[count, 1:13]
        y_test[index_test] <- y_all[count]
        index_test = index_test + 1
      }
    }

    # Treinando modelo:
    retlist<-trainELM(x_train, y_train, p, 1)
    W<-retlist[[1]]
    H<-retlist[[2]]
    Z<-retlist[[3]]

    # Calculando acurácia de treinamento
    y_hat_train <- as.matrix(YELM(x_train, Z, W, 1), nrow = length_train, ncol = 1)
    accuracy_train[execution]<-((sum(abs(y_hat_train + y_train)))/2)/length_train
```

```
    #print(paste("Acurácia de treinamento para execução", execution, "com", p, "nerôn
ios:", accuracy_train))

    # Calculando acurácia de Teste:
    y_hat_test <- as.matrix(YELM(x_test, Z, W, 1), nrow = length_test, ncol = 1)
    accuracy_test[execution]<-((sum(abs(y_hat_test + y_test)))/2)/length_test
    #print(paste("Acurácia de teste para execução", execution, "com", p, "nerônios:",
accuracy_test))
  }
  # Média das acurácias
  mean_accuracy_train <- mean(accuracy_train) * 100
  mean_accuracy_test <- mean(accuracy_test) * 100

  # Desvio Padrão das acurácias
  sd_accuracy_train <- sd(accuracy_train) * 100
  sd_accuracy_test <- sd(accuracy_test) * 100

  print(paste("Acurácia de treinamento do modelo com", p, "neurônios:", mean_accuracy
_train, "%", "±", sd_accuracy_train, "%"))
  print(paste("Acurácia de teste do modelo com", p, "neurônios:", mean_accuracy_test,
"%", "±", sd_accuracy_test, "%"))
}
```

```
## [1] "Acurácia de treinamento do modelo com 5 neurônios: 75.0263157894737 % ± 4.605
67865161688 %"
## [1] "Acurácia de teste do modelo com 5 neurônios: 73.6875 % ± 7.40593423155413 %"
## [1] "Acurácia de treinamento do modelo com 10 neurônios: 82.7368421052632 % ± 3.06
226391272479 %"
## [1] "Acurácia de teste do modelo com 10 neurônios: 78.4375 % ± 4.07323985234565 %"
## [1] "Acurácia de treinamento do modelo com 30 neurônios: 87.9473684210526 % ± 1.76
552328207311 %"
## [1] "Acurácia de teste do modelo com 30 neurônios: 81.625 % ± 4.80028782031815 %"
## [1] "Acurácia de treinamento do modelo com 50 neurônios: 89.8947368421053 % ± 1.87
524601097862 %"
## [1] "Acurácia de teste do modelo com 50 neurônios: 78.8125 % ± 4.39562806122533 %"
## [1] "Acurácia de treinamento do modelo com 100 neurônios: 95.8421052631579 % ± 1.5
4534889977766 %"
## [1] "Acurácia de teste do modelo com 100 neurônios: 72.9375 % ± 3.65617268249334
%"
## [1] "Acurácia de treinamento do modelo com 200 neurônios: 100 % ± 0 %"
## [1] "Acurácia de teste do modelo com 200 neurônios: 58.3125 % ± 4.50283171138481
%"
## [1] "Acurácia de treinamento do modelo com 300 neurônios: 100 % ± 0 %"
## [1] "Acurácia de teste do modelo com 300 neurônios: 63.6875 % ± 5.56769051147592
%"
```

### *Perceptron simples com base de dados Statlog (Heart) - Dados escalonados*

```
rm(list=ls())
source("~/Documents/UFMG/9/Redes Neurais/listas/lista 4/trainPerceptron.R")
source("~/Documents/UFMG/9/Redes Neurais/listas/lista 4/yperceptron.R")
source("~/Documents/UFMG/9/Redes Neurais/exemplos/escalonamento_matrix.R")
library(caret)

# Carregando base de dados:
path <- file.path("~/Documents/UFMG/9/Redes Neurais/listas/lista 6/heart", "heart.cs
v")
data <- read.csv(path)

# Separando dados de entrada e saída:
x_all <- as.matrix(data[1:270, 1:13])
class <- as.matrix(data[1:270, 14])
y_all <- rep(0,270)
for (count in 1:length(class)) {
  if (class[count] == 1 ){
    y_all[count] = 1
  }
  else if(class[count] == 2){
    y_all[count] = 0
  }
}

# Escalonando os valores dos atributos para que fiquem restritos entre 0 e 1
x_all <- staggeringMatrix(x_all, nrow(x_all), ncol(x_all))

# Realiza pelo 20 execuções diferentes
accuracy_train <- rep(0, 20)
accuracy_test <- rep(0, 20)
for(execution in 1:20){
  # Separando dados entre treino e teste aleatoriamente:
  positions_train <- createDataPartition(1:270,p=.7)
  length_train <- length(positions_train$Resample1)
  length_test <- length(y_all) - length_train
  x_train <- matrix(rep(0, 13*length_train), ncol=13, nrow=length_train)
  y_train <- rep(0, length_train)
  x_test <- matrix(rep(0, (13*length_test)), ncol=13, nrow=(length(y_all) - length_tr
ain))
  y_test <- rep(0, (length(y_all) - length_train))
  index_train <- 1
  index_test <- 1
  for (count in 1:length(y_all)) {
    if (index_train <= length_train && count == positions_train$Resample1[index_trai
n]){
      x_train[index_train, ] <- x_all[count, 1:13]
      y_train[index_train] <- y_all[count]
      index_train = index_train + 1
    } else {
      x_test[index_test, ] <- x_all[count, 1:13]
      y_test[index_test] <- y_all[count]
      index_test = index_test + 1
    }
  }

  # Treinando modelo:
  retlist<-trainPerceptron(x_train, y_train, 0.1, 0.01, 1000, 1)
  W<-retlist[[1]]

  # Calculando acurácia de treinamento
  y_hat_train <- as.matrix(yperceptron(x_train, W, 1), nrow = length_train, ncol = 1)
  accuracy_train[execution]<-1-((t(y_hat_train-y_train) %*% (y_hat_train-y_train))/le
ngth_train)
  #print(paste("Acurácia de treinamento para execução", execution, "com", p, "nerônio
s:", accuracy_train))
```

```
  # Calculando acurácia de Teste:
  y_hat_test <- as.matrix(yperceptron(x_test, W, 1), nrow = length_test, ncol = 1)
  accuracy_test[execution]<-1-((t(y_hat_test-y_test) %*% (y_hat_test-y_test))/length_
test)
  #print(paste("Acurácia de teste para execução", execution, "com", p, "nerônios:", a
ccuracy_test))
}
# Média das acurácias
mean_accuracy_train <- mean(accuracy_train) * 100
mean_accuracy_test <- mean(accuracy_test) * 100

# Desvio Padrão das acurácias
sd_accuracy_train <- sd(accuracy_train) * 100
sd_accuracy_test <- sd(accuracy_test) * 100

print(paste("Acurácia de treinamento do modelo com perceptron simples", mean_accuracy
_train, "%", "±", sd_accuracy_train, "%"))
```

```
## [1] "Acurácia de treinamento do modelo com perceptron simples 82.8157894736842 % ±
3.7226917914431 %"
```

```
print(paste("Acurácia de teste do modelo com perceptron simples", mean_accuracy_test,
"%", "±", sd_accuracy_test, "%"))
```

```
## [1] "Acurácia de teste do modelo com perceptron simples 78.1875 % ± 6.431253356604
92 %"
```

### *Discussão*

Pôde-se perceber que escalonar os dados fez diferença no resultado da acurácia de todos os modelos testados. Novamente, constatou-se que em ambas as bases de dados o modelo com o perceptron simples se saiu melhor que as ELMs, obtendo maiores valores acurácia.