Redes Neurais Artificiais

Exercício 8 - RBF com bases de dados reais

Vítor Gabriel Reis Caitité - 2016111849

1/25/2021

Função que calcula a saída de uma rede RBF

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

```
# Função que calcula a saída de uma rede RBF.
library("corpcor")

YRBF <- function(xin, modRBF){
  ####### Função radial Gaussiana ########
  pdfnvar<-function(x,m,K,n){
    if (n==1) {
      r<-sqrt(as.numeric(K))
      px<-(1/(sqrt(2*pi*r*r))) * exp(-0.5 *((x-m)/r)^2)
    }
    else {
      px<-((1/(sqrt((2*pi)^n * (det(K))))) * exp (-0.5 * (t(x-m) %*% (solve(K)) %*% (x-m))))
    }
  }
  ########################################
  N <- dim(xin)[1] # número de amostras
  n <- dim(xin)[2] # dimensão de entrada (deve ser maior que 1)
  m <- as.matrix(modRBF[[1]])
  covlist <- modRBF[[2]]
  p <- length(covlist) # Número de funções radiais
  W <- modRBF [[3]]

  xin <- as.matrix(xin) # garante que xin seja matriz

  H <- matrix(nrow = N, ncol = p)
  # Calcula matriz H
  for (j in 1:N) {
    for (i in 1:p) {
      mi <- m[i, ]
      covi <- covlist[i]
      covi <- matrix(unlist(covlist[i]), ncol = n, byrow = T) + 0.001 * diag(n)
      H[j,i] <- pdfnvar(xin[j, ], mi, covi, n)
    }
  }

  Haug <- cbind(1, H)
  Yhat <- Haug %*% W
  return(Yhat)
}
```

Treinamento da rede RBF com centros e raios selecionados a partir do K-means

A função abaixo é uma implementação possível, em R, para o algoritmo de treinamento de uma rede RBF com centros e raios selecionados a partir do algoritmo K-means.

```r
# Função de treinamento de uma rede RBF.
library("corpcor")

trainRBF <- function(xin, yin, p){
  ####### Função radial Gaussiana ########
  pdfnvar<-function(x,m,K,n){
    if (n==1) {
      r<-sqrt(as.numeric(K))
      px<-(1/(sqrt(2*pi*r*r)))*exp(-0.5 *((x-m)/r)^2)
    }
    else {
      px<-((1/(sqrt((2*pi)^n * (det(K))))) * exp (-0.5 * (t(x-m) %*% (solve(K)) %*% (x-m)))) #eq 6.5
    }
  }
  ########################################
  N<-dim(xin)[1] # número de amostras
  n<-dim(xin)[2] # dimensão de entrada (deve ser maior que 1)

  xin <- as.matrix(xin) # garante que xin seja matriz
  yin <- as.matrix(yin) # garante que yin seja matriz

  # Aplica o algoritmo kmeans para separar os clusters
  xclust<-kmeans(xin, p)

  # Armazena vetores de centros das funções:
  m <- as.matrix (xclust$centers)
  covlist <- list()

  # Estima matrizes de covariância para todos os centros:
  for ( i in 1:p)
  {
    ici <- which(xclust$cluster == i )
    xci <- xin [ici, ]
    if(n==1){
      covi <- var(xci)
    }
    else{
      row <- dim(xci)[1];
      if(is.null(row)){
        row <- 0
      }
      # Para garantir que não haverá erro (caso tenha apeanas uma linha)
      if(row > 1){
        covi <- cov(xci)
      }
      else{
        # cov de 2 linhas iguais que vai dar 0
        covi <- cov(matrix(c(xci, xci), nrow = 2))
      }
    }
    covlist [[i]] <- covi
  }

  H <- matrix(nrow = N, ncol = p)
  # Calcula matriz H
  for (j in 1:N) {
    for (i in 1:p) {
      mi <- m[i, ]
      covi <- covlist[i]
      covi <- matrix(unlist(covlist[i]), ncol = n, byrow = T) + 0.001 * diag(n)
      H[j,i] <- pdfnvar(xin[j, ] , mi, covi, n)
    }
  }

  Haug <-cbind(1, H)
  W <- pseudoinverse(Haug) %*% yin

  return (list(m, covlist, W, H))
}
```

Treinamento da rede RBF com centros e raios atribuídos de

Treinamento da rede RBF com centros e raios atribuídos de forma aleatória aos neurônios

A função abaixo é uma implementação possível, em R, para o algoritmo de treinamento de uma rede RBF com centros e raios atribuídos de forma aleatória aos neurônios. A estratégia utilizada para a construção dos centros foi colocá-los entre 2 pontos escolhidos aleatoriamente do conjunto de treinamento, com o raio da função igual à distância entre os pontos.

```
# Função de treinamento de uma rede RBF.
library("corpcor")

trainRandomRBF <- function(xin, yin, p){
  ####### Função para cálculo de dist. euclidiana ########
  euc.dist <- function(x1, x2){
    return (sqrt(sum((x1 - x2) ^ 2)))
  }

  ####### Função radial Gaussiana ########
  pdfnvar<-function(x,m,K,n){
    if (n==1) {
      r<-sqrt(as.numeric(K))
      px<-(1/(sqrt(2*pi*r*r)))*exp(-0.5 *((x-m)/r)^2)
    }
    else {
      px<-((1/(sqrt((2*pi)^n * (det(K))))) * exp (-0.5 * (t(x-m) %*% (solve(K)) %*% (x-m)))) #eq 6.5
    }
  }
  ########################################

  N<-dim(xin)[1] # número de amostras
  n<-dim(xin)[2] # dimensão de entrada (deve ser maior que 1)

  xin <- as.matrix(xin) # garante que xin seja matriz
  yin <- as.matrix(yin) # garante que yin seja matriz

  center <- matrix(rep(0, (p*ncol(xin))), nrow = p)
  radius <- rep(0, p)
  for (index in 1:p) {
    # Escolhendo 2 pontos aleatoriamente (e garantindo que eles são diferentes):
    point1 <- sample(1:N, 1)
    point2 <- point1
    while (point1 == point2) {
      point2 <- sample(1:N, 1)
    }
    point1 <- xin[point1,]
    point2 <- xin[point2,]
    # Centros e Raios:
    center[index,] <- (point1+point2)/2
    radius[index] <- euc.dist(point1, point2)
  }

  # Armazena vetores de centros das funções:
  m <- as.matrix (center)

  covlist <- list()

  # Estima matrizes de covariância para todos os centros:
  for (i in 1:p)
  {
    ici<-rep(0, N)
    for (index in 1:N) {
      if(euc.dist(center[i,], xin[index,]) <= radius[i]){
        ici[index]<-1
      }
    }
    ici <- which(ici == 1 )
    xci <- xin [ici, ]
    if(n==1){
      covi <- var(xci)
    }
    else{
      row <- dim(xci)[1];
      if(is.null(row)){
        row <- 0
```

```
      row <- 0
    }
    # Para garantir que não haverá erro (caso tenha apeanas uma linha)
    if(row > 1){
      covi <- cov(xci)
    }
    else{
      # cov de 2 linhas iguais que vai dar 0
      covi <- cov(matrix(c(xci, xci), nrow = 2))
    }
  }
  covlist [[i]] <- covi
}

H <- matrix(nrow = N, ncol = p)
# Calcula matriz H
for (j in 1:N) {
  for (i in 1:p) {
    mi <- m[i, ]
    covi <- covlist[i]
    covi <- matrix(unlist(covlist[i]), ncol = n, byrow = T) + 0.001 * diag(n)
    H[j,i] <- pdfnvar(xin[j, ] , mi, covi, n)
  }
}

Haug <-cbind(1, H)
W <- pseudoinverse(Haug) %*% yin

return (list(m, covlist, W, H))
}
```

Enunciado Exercício 8

O objetivo do exercício desta semana é combinar os conceitos aprendidos na Unidade 2 e construir uma rede neural que soma elementos das redes RBF e das redes ELM.

As bases de dados a serem estudadas são as mesmas do exercício 6:

• Breast Cancer (diagnostic)

• Statlog (Heart)

Os mesmos cuidados para separação de conjunto de treinamento e teste, já mencionados no enunciado do exercício 6, devem ser tomados, bem como deve ser dada atenção ao escalonamento dos dados (entre [0; 1] ou [−1; 1]).

Para o exercício desta semana, o aluno deve combinar os algoritmos de treinamento de redes ELM e RBF: construir uma rede RBF com centros e raios atribuídos de forma aleatória aos neurônios. Uma possibilidade, que não é a única nem a melhor, para a construção de centros é colocá-los entre 2 pontos escolhidos aleatoriamente do conjunto de treinamento, com o raio da função igual à distância entre os pontos.

Além da RBF com centros e raios aleatórios, deve ser construída uma RBF com centros e raios selecionados a partir do k-médias. As acurácias obtidas por cada uma das redes nas duas bases devem ser apresentadas no formato media ± desvio e comparadas com os resultados obtidos no exercício 6 para ELMs.

Deve ser comparado, também, o número de centros necessários para desempenho semelhante entre as redes RBF com centros aleatórios e com centros selecionados por agrupamento (k-médias).

RBF com centros selecionados pelo k-means - Base de dados Breast Cancer (diagnostic)

Ulizando a base de dados Breast Cancer (diagnostic) foi desenvolvida uma rede neural RBF 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, 2, 5, 10, 30, 50 e 100 neurônios. Para cada número de neurônios foram realizados 10 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/trainRBF.R")
source("~/Documents/UFMG/9/Redes Neurais/exemplos/YRBF.R")
source("~/Documents/UFMG/9/Redes Neurais/exemplos/escalonamento_matrix.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 8/databases/cancer", "wdbc.csv")
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(2,5,10,30,50,75,100)){
  # Realiza pelo 20 execuções diferentes
  accuracy_train <- rep(0, 10)
  accuracy_test <- rep(0, 10)
  for(execution in 1:10){
    # 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_train]){
        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:
    modRBF<-trainRBF(x_train, y_train, p)

    # Calculando acurácia de treinamento
    y_hat_train <- as.matrix(YRBF(x_train, modRBF), nrow = length_train, ncol = 1)
    yt <- (1*(y_hat_train >= 0)-0.5)*2
    accuracy_train[execution] <- 1 - ((t(yt - y_train) %*% (yt - y_train))/(4*length(y_train)))
    #print(paste("Acurácia de treinamento para execução", execution, "com", p, "nerônios:", accuracy_train))

    # Calculando acurácia de Teste:
    y_hat_test <- as.matrix(YRBF(x_test, modRBF), nrow = length_test, ncol = 1)
    yt <- (1*(y_hat_test >= 0)-0.5)*2
    accuracy_test[execution] <- 1 - ((t(yt - y_test) %*% (yt - y_test))/(4*length(y_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_ac
curacy_train, "%"))
  print(paste("Acurácia de teste do modelo com", p, "neurônios:", mean_accuracy_test, "%", "±", sd_accuracy_
test, "%"))
}
```

```
## [1] "Acurácia de treinamento do modelo com 2 neurônios: 77.9800498753117 % ± 12.670058113519 %"
## [1] "Acurácia de teste do modelo com 2 neurônios: 76.7261904761905 % ± 13.4261051151803 %"
## [1] "Acurácia de treinamento do modelo com 5 neurônios: 83.4164588528678 % ± 7.85810997499113 %"
## [1] "Acurácia de teste do modelo com 5 neurônios: 81.8452380952381 % ± 6.9656705085484 %"
## [1] "Acurácia de treinamento do modelo com 10 neurônios: 82.6683291770574 % ± 7.18182360115978 %"
## [1] "Acurácia de teste do modelo com 10 neurônios: 79.2857142857143 % ± 5.98404277655375 %"
## [1] "Acurácia de treinamento do modelo com 30 neurônios: 89.2518703241895 % ± 2.31396722387273 %"
## [1] "Acurácia de teste do modelo com 30 neurônios: 80.3571428571429 % ± 2.84775795523955 %"
## [1] "Acurácia de treinamento do modelo com 50 neurônios: 90.8229426433915 % ± 1.2650382297421 %"
## [1] "Acurácia de teste do modelo com 50 neurônios: 79.7619047619048 % ± 2.31386979954179 %"
## [1] "Acurácia de treinamento do modelo com 75 neurônios: 90.9476309226933 % ± 0.99785253063492 %"
## [1] "Acurácia de teste do modelo com 75 neurônios: 81.6071428571429 % ± 4.49351104053427 %"
## [1] "Acurácia de treinamento do modelo com 100 neurônios: 92.5935162094763 % ± 0.622609586403126 %"
## [1] "Acurácia de teste do modelo com 100 neurônios: 82.6190476190476 % ± 3.78339251954065 %"
```

RBF com centros selecionados pelo k-means - Base de dados Statlog (Heart)

Ulizando a base de dados Statlog (Heart), foi desenvolvida uma rede RBF 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, 2, 5, 10, 30, 50, 75 e 100 neurônios. Para cada número de neurônios foram realizados 10 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/trainRBF.R")
source("~/Documents/UFMG/9/Redes Neurais/exemplos/YRBF.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 8/databases/heart", "heart.csv")
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(2,5,10,30,50,75,100)){
  # Realiza pelo 20 execuções diferentes
  accuracy_train <- rep(0, 20)
```

```
  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_train]){
        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:
    modRBF<-trainRBF(x_train, y_train, p)

    # Calculando acurácia de treinamento
    y_hat_train <- as.matrix(YRBF(x_train, modRBF), nrow = length_train, ncol = 1)
    yt <- (1*(y_hat_train >= 0)-0.5)*2
    accuracy_train[execution] <- 1 - ((t(yt - y_train) %*% (yt - y_train))/(4*length(y_train)))
    #print(paste("Acurácia de treinamento para execução", execution, "com", p, "nerônios:", accuracy_train))

    # Calculando acurácia de Teste:
    y_hat_test <- as.matrix(YRBF(x_test, modRBF), nrow = length_test, ncol = 1)
    yt <- (1*(y_hat_test >= 0)-0.5)*2
    accuracy_test[execution] <- 1 - ((t(yt - y_test) %*% (yt - y_test))/(4*length(y_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_ac
curacy_train, "%"))
  print(paste("Acurácia de teste do modelo com", p, "neurônios:", mean_accuracy_test, "%", "±", sd_accuracy_
test, "%"))
}
```

```
## [1] "Acurácia de treinamento do modelo com 2 neurônios: 65.9210526315789 % ± 4.70402606330547 %"
## [1] "Acurácia de teste do modelo com 2 neurônios: 62.375 % ± 9.42334028385175 %"
## [1] "Acurácia de treinamento do modelo com 5 neurônios: 69.0789473684211 % ± 3.45708701782044 %"
## [1] "Acurácia de teste do modelo com 5 neurônios: 63.9375 % ± 5.22951831836406 %"
## [1] "Acurácia de treinamento do modelo com 10 neurônios: 77.2894736842105 % ± 4.20454908147929 %"
## [1] "Acurácia de teste do modelo com 10 neurônios: 67.8125 % ± 6.92861866312091 %"
## [1] "Acurácia de treinamento do modelo com 30 neurônios: 83.2105263157895 % ± 1.56410382041726 %"
## [1] "Acurácia de teste do modelo com 30 neurônios: 68.125 % ± 5.1057887683852 %"
## [1] "Acurácia de treinamento do modelo com 50 neurônios: 84.9210526315789 % ± 1.7184453384106 %"
## [1] "Acurácia de teste do modelo com 50 neurônios: 64.5 % ± 5.69799133582686 %"
## [1] "Acurácia de treinamento do modelo com 75 neurônios: 87.6842105263158 % ± 1.31606645284183 %"
## [1] "Acurácia de teste do modelo com 75 neurônios: 63 % ± 4.35663438423455 %"
## [1] "Acurácia de treinamento do modelo com 100 neurônios: 90.1842105263158 % ± 1.43147913942445 %"
## [1] "Acurácia de teste do modelo com 100 neurônios: 61 % ± 2.8562028528874 %"
```

RBF com centros selecionados aleatoriamente - Base de dados

Breast Cancer (diagnostic)

Ulizando a base de dados Breast Cancer (diagnostic) foi desenvolvida uma rede neural RBF, com centros selecionados aleatoriamente, 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, 2, 5, 10, 30, 50 e 100 neurônios. Para cada número de neurônios foram realizados 10 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/trainRandomCentersRBF.R")
source("~/Documents/UFMG/9/Redes Neurais/exemplos/YRBF.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 8/databases/cancer", "wdbc.csv")
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(2,5,10,30,50,75,100)){
  # Realiza pelo 20 execuções diferentes
  accuracy_train <- rep(0, 10)
  accuracy_test <- rep(0, 10)
  for(execution in 1:10){
    # 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_train]){
        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:
    modRBF<-trainRandomRBF(x_train, y_train, p)

    # Calculando acurácia de treinamento
    y_hat_train <- as.matrix(YRBF(x_train, modRBF), nrow = length_train, ncol = 1)
    yt <- (1*(y_hat_train >= 0)-0.5)*2
    accuracy_train[execution] <- 1 - ((t(yt - y_train) %*% (yt - y_train))/(4*length(y_train)))
    #print(paste("Acurácia de treinamento para execução", execution, "com", p, "nerônios:", accuracy_train))

    # Calculando acurácia de Teste:
```

```
    # Calculando acurácia de teste:
    y_hat_test <- as.matrix(YRBF(x_test, modRBF), nrow = length_test, ncol = 1)
    yt <- (1*(y_hat_test >= 0)-0.5)*2
    accuracy_test[execution] <- 1 - ((t(yt - y_test) %*% (yt - y_test))/(4*length(y_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_ac
curacy_train, "%"))
  print(paste("Acurácia de teste do modelo com", p, "neurônios:", mean_accuracy_test, "%", "±", sd_accuracy_
test, "%"))
}
```

```
## [1] "Acurácia de treinamento do modelo com 2 neurônios: 63.3167082294264 % ± 2.63246853137391 %"
## [1] "Acurácia de teste do modelo com 2 neurônios: 63.5714285714286 % ± 1.70219131662914 %"
## [1] "Acurácia de treinamento do modelo com 5 neurônios: 62.4688279301746 % ± 9.19974576801087 %"
## [1] "Acurácia de teste do modelo com 5 neurônios: 62.5 % ± 10.8203160352667 %"
## [1] "Acurácia de treinamento do modelo com 10 neurônios: 65.5860349127182 % ± 3.55404128452734 %"
## [1] "Acurácia de teste do modelo com 10 neurônios: 67.4404761904762 % ± 3.33250651121139 %"
## [1] "Acurácia de treinamento do modelo com 30 neurônios: 73.0922693266833 % ± 4.06627407410361 %"
## [1] "Acurácia de teste do modelo com 30 neurônios: 71.0714285714286 % ± 3.97023759945421 %"
## [1] "Acurácia de treinamento do modelo com 50 neurônios: 73.216957605985 % ± 4.28269904886776 %"
## [1] "Acurácia de teste do modelo com 50 neurônios: 70.2976190476191 % ± 5.89556214972114 %"
## [1] "Acurácia de treinamento do modelo com 75 neurônios: 76.6583541147132 % ± 2.08709623627268 %"
## [1] "Acurácia de teste do modelo com 75 neurônios: 74.1071428571429 % ± 4.62135503922703 %"
## [1] "Acurácia de treinamento do modelo com 100 neurônios: 78.4538653366584 % ± 2.89439191579679 %"
## [1] "Acurácia de teste do modelo com 100 neurônios: 74.7023809523809 % ± 5.46446006107953 %"
```

RBF com centros selecionados aleatoriamente - Base de dados Statlog (Heart)

Ulizando a base de dados Statlog (Heart), foi desenvolvida uma rede RBF, com centros selecionados aleatoriamente, 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, 2, 5, 10, 30, 50, 75 e 100 neurônios. Para cada número de neurônios foram realizados 10 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/trainRandomCentersRBF.R")
source("~/Documents/UFMG/9/Redes Neurais/exemplos/YRBF.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 8/databases/heart", "heart.csv")
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(2,5,10,30,50,75,100)){
  # 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_train]){
        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:
    modRBF<-trainRandomRBF(x_train, y_train, p)

    # Calculando acurácia de treinamento
    y_hat_train <- as.matrix(YRBF(x_train, modRBF), nrow = length_train, ncol = 1)
    yt <- (1*(y_hat_train >= 0)-0.5)*2
    accuracy_train[execution] <- 1 - ((t(yt - y_train) %*% (yt - y_train))/(4*length(y_train)))
    #print(paste("Acurácia de treinamento para execução", execution, "com", p, "nerônios:", accuracy_train))

    # Calculando acurácia de Teste:
    y_hat_test <- as.matrix(YRBF(x_test, modRBF), nrow = length_test, ncol = 1)
    yt <- (1*(y_hat_test >= 0)-0.5)*2
    accuracy_test[execution] <- 1 - ((t(yt - y_test) %*% (yt - y_test))/(4*length(y_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_ac
curacy_train, "%"))
  print(paste("Acurácia de teste do modelo com", p, "neurônios:", mean_accuracy_test, "%", "±", sd_accuracy_
test, "%"))
}
```

```
## [1] "Acurácia de treinamento do modelo com 2 neurônios: 58.0526315789474 % ± 2.22052874839253 %"
## [1] "Acurácia de teste do modelo com 2 neurônios: 54.8125 % ± 5.22951831836406 %"
## [1] "Acurácia de treinamento do modelo com 5 neurônios: 61.7894736842105 % ± 4.33473310202059 %"
## [1] "Acurácia de teste do modelo com 5 neurônios: 56.8125 % ± 4.02449326656557 %"
## [1] "Acurácia de treinamento do modelo com 10 neurônios: 62.0789473684211 % ± 6.69558654166313 %"
## [1] "Acurácia de teste do modelo com 10 neurônios: 59.375 % ± 8.31632035280786 %"
## [1] "Acurácia de treinamento do modelo com 30 neurônios: 67.1842105263158 % ± 9.96022853655299 %"
## [1] "Acurácia de teste do modelo com 30 neurônios: 60.75 % ± 7.22522991803175 %"
## [1] "Acurácia de treinamento do modelo com 50 neurônios: 66.9473684210526 % ± 11.3892801351699 %"
## [1] "Acurácia de teste do modelo com 50 neurônios: 64.25 % ± 10.4535312593149 %"
## [1] "Acurácia de treinamento do modelo com 75 neurônios: 72.8947368421053 % ± 5.88191125499726 %"
## [1] "Acurácia de teste do modelo com 75 neurônios: 68.9375 % ± 6.31251628450218 %"
## [1] "Acurácia de treinamento do modelo com 100 neurônios: 89.4210526315789 % ± 1.55475464030308 %"
## [1] "Acurácia de teste do modelo com 100 neurônios: 63.8125 % ± 4.65018038699349 %"
```

DISCUSSÃO E RESULTADOS:

Base de dados Breast Cancer (diagnostic)

Resultados de teste da rede RBF – K-means

N° neurônios na camada escondida	Média das acurácias (%)	Desvio padrão (%)
2	76.72	13.42
5	81.84	6.96
10	79.29	5.98
30	80.35	2.84
50	79.76	2.31
75	81.61	4.49
100	82.62	3.78

Resultados da rede RBF – centros aleatórios

N° neurônios na camada escondida	Média das acurácias (%)	Desvio padrão (%)
2	63.57	1.70
5	62.50	10.82
10	67.44	3.33
30	71.07	3.97
50	70.29	5.89
75	74.10	4.62
100	74.70	5.46

Resultados da ELM

N° neurônios na camada escondida	Média das acurácias (%)	Desvio padrão (%)
5	77.14	5.91
10	84.52	4.42
30	92.26	2.48
50	94.79	1.65
100	91.69	2.18
200	83.36	3.37
300	72.38	4.21

Como pode-se notar nas tabelas acima, a rede RBF com centros selecionados com o algoritmo K-means obteve melhores resultados que a rede RBF com centros aleatórios, mostrando assim a importância do algortimo de seleção de centros. Contudo. ao compararmos os resultados da rede RBF com os da ELM, nota-se também que a última foi superior em todos os testes, obtendo por exemplo uma acurácia média máxima de 94,79%, enquanto a obtida pela rede RBF foi de 82,62.

Base de dados Statlog (Heart)

Resultados de teste da rede RBF – K-means

N° neurônios na camada escondida	Média das acurácias (%)	Desvio padrão (%)
2	62.37	9.42
5	63.94	5.22
10	67.81	6.92
30	68.12	5.10
50	64.50	5.69
75	63.00	4.35
100	61	2.85

Resultados da rede RBF – centros aleatórios

N° neurônios na camada escondida	Média das acurácias	Desvio padrão
2	54.81	5.23
5	56.81	4.02
10	59.37	8.31
30	60.75	7.22
50	64.25	10.45
75	68.93	6.31
100	63.81	4.65

Resultados da ELM

N° neurônios na camada escondida	Média das acurácias	Desvio padrão
5	73.68	7.40
10	78.43	4.07
30	81.62	4.80
50	78.81	4.39
100	72.93	3.65
200	58.31	4.50
300	63.68	5.56

Como pode-se notar nas tabelas acima, até 50 neurônios a rede RBF com centros selecionados com o algoritmo K-means obteve melhores resultados que a rede RBF com centros aleatórios. Porém para 75 e 100 neurônios, a RBF com centros aleatórios foi melhor. Isso mostra que um método mais simples as vezes pode ser interessante e obter resultados semelhantes aos mais complexos sem exigir tanto poder computacional. Contudo ao compararmos os resultados da rede RBF com os da ELM, nota-se também que a última foi superior na maioria dos testes, obtendo por exemplo uma acurácia média máxima de 81.62%, enquanto a obtida pela rede RBF foi de 68.93.