Trabalho Intermediário RNA 2020/2

Aluno: Vítor Gabriel Reis Caitité **Matrícula:** 2016111849

1 Introdução

1.1 Objetivo

O objetivo do trabalho é aplicar os modelos de redes neurais artificiais estudados na disciplina a um problema prático referente a um conjunto de dados disponibilizado publicamente e que envolve o reconhecimento de sotaques por falantes em línguas diferentes, como Alemão, Francês e Inglês. Para facilitar o problema e utilizarmos apenas os métodos aprendidos até aqui, o trabalho considera apenas duas línguas, Francês e Inglês Britânico (ou seja, se trata de um problema de classificação binário).

1.2 Dados

Basicamente o arquivo contendo os dados de treinamento contêm 1 coluna de Id (apenas um identificador) seguida de 12 colunas contendo os vetores de entrada x (12 coeficientes espectrais da fala), e por fim, uma última coluna contendo a classificação da língua correspondente o rótulo y daquela amostra. Foram disponibilizados 53 dados para treinamento.

Além disso, foi disponibilizado um conjunto de teste (é sobre esse conjunto que os resultados foram obtidos e enviados para avaliação). Assim como o arquivo de treino, o arquivo contendo os dados a serem testados contém um identificados e mais 12 colunas correspondendo as características de entrada extraídas. Foram disponibilizados 22 dados para teste.

2 Desenvolvimento

Inicialmente, buscou-se entender mais sobre os dados do problema e por se tratar de uma pequena quantidade de dados decidiu - se plotar gráficos das amostras usando apenas 2 dimensões de cada vez. Isso foi feito somente no intuito de tentar visualizar uma possível separação espacial. Esses gráficos podem ser vistos na Figura 1.

OBS: O problema será resolvido utilizando todas as 12 dimensões, nesse ponto buscou-se apenas investigar e mostrar uma possível separação espacial.

Como pode ser observado nos gráficos, e já era esperado, para alguns pares de variáveis de entrada tem-se uma separação mais clara e para outros isso não ocorre.

É interessante notar como algumas características de entrada são mais influentes em causar uma separação espacial que outras (por exemplo, na Figura 1, observa-se maior separação espacial em gráficos que têm a entrada X1 como um de seu eixos do que aqueles que possuem a entrada X2 como um de seu eixos). Apesar de a seleção de atributos não ser abordada nesse trabalho, vale destacar que ela desempenha uma tarefa essencial dentro do processo de pré-processamento em problemas mais complexos. Seu objetivo é selecionar os atributos mais importantes, pois atributos não relevantes e/ou redundantes podem reduzir a precisão e a compreensibilidade das hipóteses induzidas por algoritmos de aprendizado supervisionado.

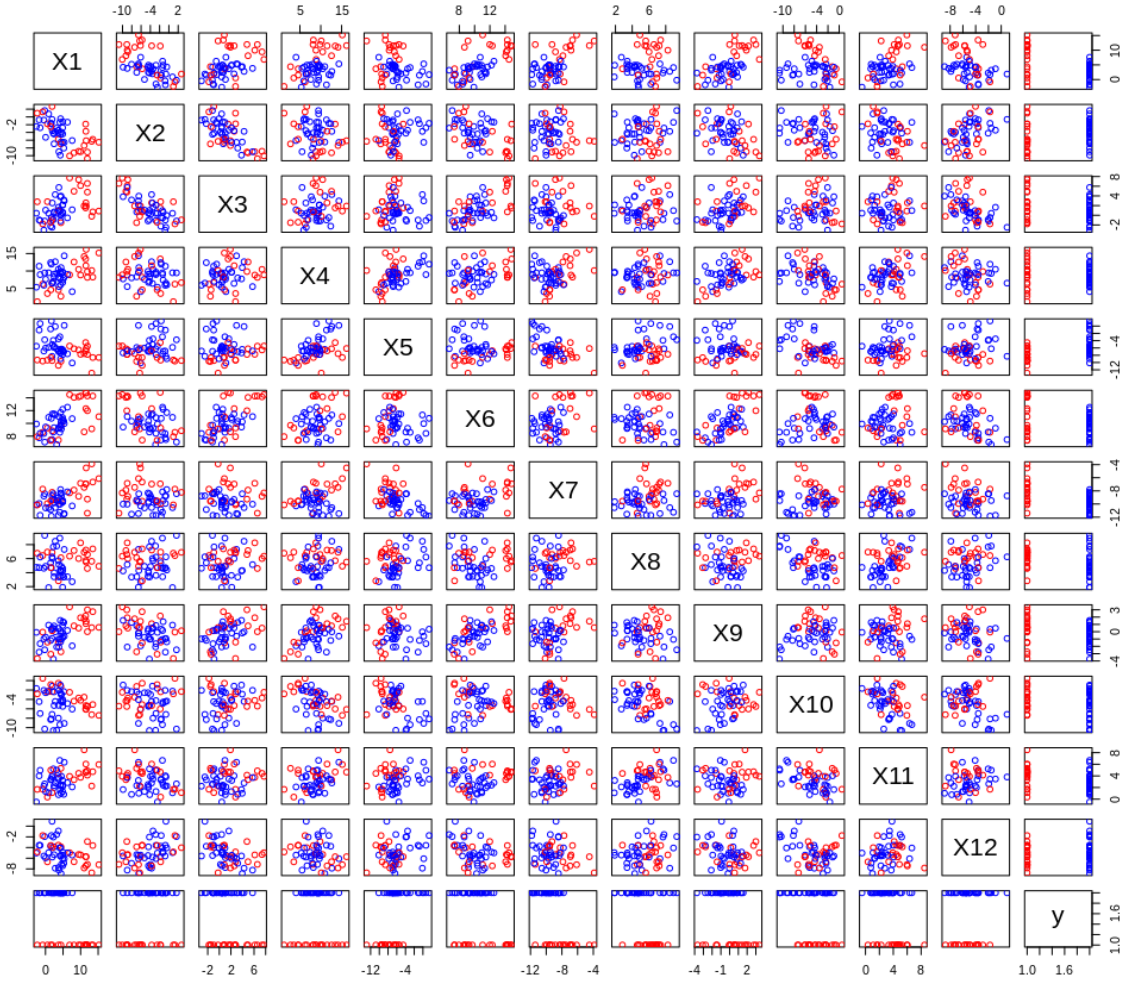

Figura 1: Amostras das 2 classes do conjunto de treinamento para todos os pares de variáveis de entrada. Fonte: Imagem produzida pelo autor utilizando a função de plot do R.

Feita essa análise inicial dos dados, o próximo passo, antes de partir de vez para resolução do problema de classificação de fato, foi realizar um escalonamento dos dados. O objetivo disso é alterar os valores das colunas numéricas no conjunto de dados para uma escala comum, aumentando a coesão dos tipos de entrada, sem distorcer as diferenças nos intervalos de valores. Esse processo pode gerar um impacto significativo no modelo.

Finalmente, terminada essa fase de pré-processamento, partiu-se para geração dos classificadores de fato. Inicialmente decidiu-se começar com o algoritmo do Perceptron Simples.

2.1 Perceptron Simples

Como se sabe, o perceptron simples pode ser utilizado para dividir duas classes linearmente separáveis. Ou seja, o modelo de classificador desenvolvido com esse perceptron de camada única (cuja a rede está mostrada na Figura 2) é na verdade um classificador linear. Dependendo da dimensão o problema esse classificador representa uma reta, um plano ou um hiperplano no espaço de entrada. Como estamos lidando com um problema com 12 variáveis de entrada, então o classificador gerado a partir do perceptron simples representará um hiperplano nesse espaço de entrada, tentando separar os dados linearmente.

Dito isso, é preciso reconhecer que apesar de estar aplicando primeiramente esse método, caso o problema apresente classes não linearmente separáveis, então a classificação feita apresentará um erro alto.

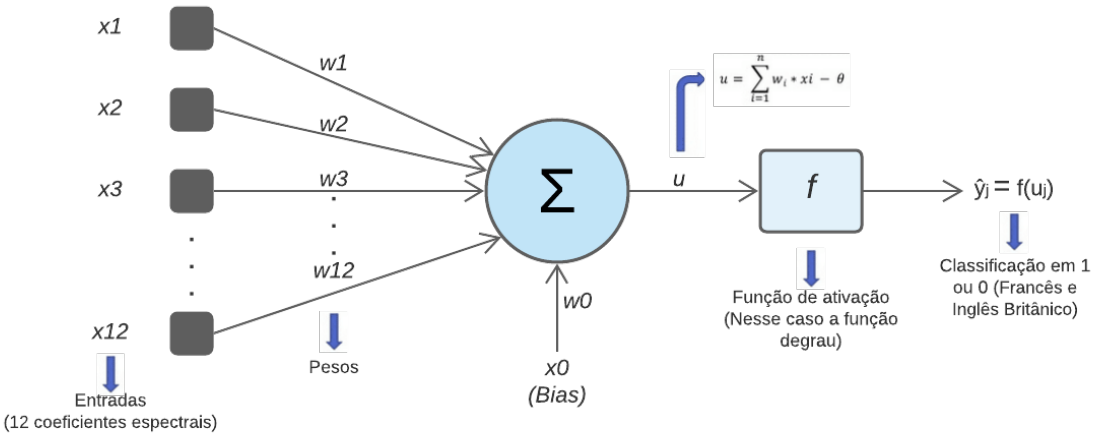

Figura 2: Rede Perceptron Simples. Fonte: Imagem produzida pelo autor

Utilizando - se o algoritmo de treinamento do perceptron simples apresentado durante a disciplina pôde-se, então, treinar o modelo. Esse algoritmo de treinamento baseia-se na aplicação da Equação 1 sobre os dados de treinamento até que o erro global atinja um critério de parada ou complete-se o número máximo de iterações passado como parâmetro.

$$w(t+1) = w(t) + \eta * e(t) * x(t) \tag{1}$$

onde:

- w(t) - valores d vetor de pesos no instante t;
- e(t) - valor do erro no instante t;
- x(t) - vetor de entradas no instante t;
- η - passo de treinamento.

A rede foi então treinada (invocou-se a função de treinamento, passando como parâmetros um passo de 0.1, uma tolerância de 0.01 e um máximo de épocas igual a 1000). Obteve-se uma acurácia de treinamento de 100%, o que indica que aparentemente as classes estavam sim linearmente separáveis. Assim, obteve-se o seguinte vetor de pesos w:

[1.65067954 ($w0$); -1.28948662; -0.14778199; -0.30544622; 0.36410129; 1.73950845; -0.03441666; -0.45398101; -1.24740465; -0.13023062; -0.67522570; -0.61194051; 0.01362064]

Por fim, aplicou-se esse modelo para a classificação dos dados de teste e submeteu-se o resultado no *Kaggle*, obtendo-se uma acurácia de 0.90909 de acordo com o leaderboard do *Kaggle* [1]. Considerou-se esse resultado bastante positivo, principalmente por se tratar de um classificador linear, ou seja que depende das classes serem linearmente separáveis.

2.2 Máquinas de Aprendizado Extremo - ELM

Apesar da solução anterior parecer bastante satisfatória ainda era necessário continuar avançando. Para isso, esse próximo classificador que foi desenvolvido e testado segue o modelo ELM. O algoritmo da ELM nada mais é do que uma maneira diferente de treinar uma rede neural de apenas uma camada oculta. O princípio de funcionamento da ELM é o mesmo de uma RNA, todavia a metodologia de treinamento de uma ELM não é baseada em gradiente descendente. O treinamento da

[1] This leaderboard is calculated with approximately 50% of the test data. The final results will be based on the other 50%, so the final standings may be different.

ELM é bastante simples e evita-se gasto computacional com métodos iterativos. Os pesos de entrada e o bias da camada escondida são escolhidos aleatoriamente. E os pesos da camada de saída são determinados analiticamente (sem ajuste iterativo de parâmetros). O princípio básico da ELM é que a matriz de pesos da camada escondida, selecionada aleatoriamente, seja suficientemente grande para garantir a separabilidade. Assim, garantida uma projeção linearmente separável, pode-se encontrar um separador linear de maneira analítica através da pseudo-inversa.

Para decidir-se o número de neurônios da camada escondida, separou-se os dados de treinamento em dados que realmente foram usados para treinamento (80%) e em dados que foram utilizados para validação (20%). Assim, rodou-se o algoritmo, validando-o para diferentes valores de neurônios e percebeu-se que com 20 neurônios na camada escondida obteve-se maior acurácia, com a solução normalmente acertando todas as classificações. Então. reagrupou-se os dados novamente em um conjunto de treinamento, treinou-se o modelo e aplicou-se aos dados de teste. Ao submeter os resultados obteve-se uma acurácia de 100% de acordo com o leaderboard do *Kaggle*, alcançando assim o resultado final que foi enviado para avaliação.

2.3 Redes RBF

Apesar de já se ter alcançado o objetivo do trabalho, para fins didáticos decidiu-se testar também um classificador baseado no modelo de redes RBF (*Radial Basis Functions Neural Networks*). Esse tipo de rede é caracterizada pela utilização de funções radiais nos neurônios da camada escondida, cujas as respostas são combinadas de maneira linear para gerar a saída.

Utilizou-se um algoritmo de treinamento de uma rede RBF com centros e raios selecionados a partir do algoritmo *K-means* (método de *Clustering* que visa particionar n observações dentre k grupos onde cada observação pertence ao grupo mais próximo da média).

Utilizando a mesma estratégia de validação explicada na subseção anterior encontrou-se que o melhor resultado com esse método foi obtido utilizando 5 neurônios na camada escondida (ou seja, aplicando o algoritmo *K-means* com 5 *clusters*).

Contudo há de se citar que esse método não conseguiu obter uma acurácia de teste de 100%, como o anterior. A acurácia de teste máxima obtida foi 0.9545455.

OBS: À título de curiosidade testou-se também um classificados baseado em uma RBF com centros selecionados aleatoriamente (como feito na lista 8), porém esse modelo obteve os piores resultados com acurácias de teste entre 70% e 80%.

3 Anexo - Códigos Utilizados

3.1 Treinamento Perceptron Simples

```
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)
}
```

Listing 1: Função de treinamento de um perceptron simples em R

3.2 Resposta do Perceptron Simples

```
yperceptron <- function(xvec, w, par){
  # xvec: vetor de entrada
  # w: vetor de pesos
```

```
  # par: se adiciona ou nao 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))
}
```

Listing 2: Função que calcula a resposta de um perceptron simples em R

3.3 Classificador Utilizando o Perceptron Simples

```
rm(list=ls())
source("~/Documents/UFMG/9/Redes Neurais/TP1/prediction-competition-
    accent-recognition/Simple_Perceptron/trainPerceptron.R")
source("~/Documents/UFMG/9/Redes Neurais/TP1/prediction-competition-
    accent-recognition/Simple_Perceptron/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/TP1/prediction-
    competition-accent-recognition/databases", "treino.csv")
data_train <- read.csv(path)
path <- file.path("~/Documents/UFMG/9/Redes Neurais/TP1/prediction-
    competition-accent-recognition/databases", "teste.csv")
data_test <- read.csv(path)

# Separando dados de entrada e saida e treino e teste:
x_train <- as.matrix(data_train[1:53, 2:13])
class <- as.matrix(data_train[1:53, 14])
y_train <- rep(0,53)
for (count in 1:length(class)) {
  if (class[count] == 1 ){
    y_train[count] <- 1
  }
  else{
    y_train[count] <- 0
  }
}
x_test <- as.matrix(data_test[1:22, 2:13])

# Escalonando os valores dos atributos para que fiquem restritos entre
    0 e 1
x_all <- rbind(x_train, x_test)
```

```
x_all <- staggeringMatrix(x_all, nrow(x_all), ncol(x_all))
x_train <- x_all[1:53, ]
x_test <- x_all[54:75, ]

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

# Calculando acuracia de treinamento
length_train <- length(y_train)
y_hat_train <- as.matrix(yperceptron(x_train, W, 1), nrow = length_
    train, ncol = 1)
accuracy_train <- 1-((t(y_hat_train-y_train) %*% (y_hat_train-y_train))
    /length_train)

# Rodando dados de teste:
y_hat_test <- as.matrix(yperceptron(x_test, W, 1), nrow = length_test,
    ncol = 1)
y <- ifelse(y_hat_test == 0, -1, 1)

Id <- 54:75
table <- data.frame(Id, y)
write.csv(table, "prediction_perceptron.csv", row.names = FALSE)
```

Listing 3: Classificador para o problema proposto, utilizando um perceptron simples em R

3.4 Treinamento de ELMs

```
library("corpcor")

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

  #Adiciona ou nao o termo de polarizacao
  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))
```

```
}
```

Listing 4: Função de treinamento de ELMs em R

3.5 Resposta da Rede ELM

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

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

Listing 5: Função que calcula a resposta de uma rede ELM em R

3.6 Classificador Utilizando uma Rede ELM

```
rm(list=ls())
source("~/Documents/UFMG/9/Redes Neurais/TP1/prediction-competition-
    accent-recognition/ELM/trainELM.R")
source("~/Documents/UFMG/9/Redes Neurais/TP1/prediction-competition-
    accent-recognition/ELM/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/TP1/prediction-
    competition-accent-recognition/databases", "treino.csv")
data_train <- read.csv(path)
path <- file.path("~/Documents/UFMG/9/Redes Neurais/TP1/prediction-
    competition-accent-recognition/databases", "teste.csv")
data_test <- read.csv(path)

# Separando dados de entrada e saida e treino e teste:
x_train <- as.matrix(data_train[1:53, 2:13])
y_train <- as.matrix(data_train[1:53, 14])
x_test <- as.matrix(data_test[1:22, 2:13])

# Escalonando os valores dos atributos para que fiquem restritos entre
    0 e 1
x_all <- rbind(x_train, x_test)
```

```
x_all <- staggeringMatrix(x_all, nrow(x_all), ncol(x_all))
x_train <- x_all[1:53, ]
x_test <- x_all[54:75, ]

p <- 20 # numero de neuronios
executions <- 31
results <- matrix(nrow = nrow(x_test), ncol = executions)
for (index in 1:executions){
  # Treinando modelo:
  retlist<-trainELM(x_train, y_train, p, 1)
  W<-retlist[[1]]
  H<-retlist[[2]]
  Z<-retlist[[3]]

  # Calculando acuracia de treinamento
  length_train <- length(y_train)
  y_hat_train <- as.matrix(YELM(x_train, Z, W, 1), nrow = length_train,
      ncol = 1)
  accuracy_train<-((sum(abs(y_hat_train + y_train)))/2)/length_train
  #print(accuracy_train)

  # Rodando dados de teste:
  y_hat_test <- as.matrix(YELM(x_test, Z, W, 1), nrow = length_test,
      ncol = 1)
  results[,index] <- y_hat_test
}

y <- rep(0, 22)
for (index in 1:22) {
  if(sum(results[index,] == 1) > (executions/2)){
    y[index] <- 1
  }
  else{
    y[index] <- -1
  }
}

Id <- 54:75
table <- data.frame(Id, y)
write.csv(table, "prediction_eml20.csv", row.names = FALSE)
```

Listing 6: Classificador para o problema proposto, utilizando uma rede ELM em R

3.7 Treinamento de RBFs

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

```

```
trainRBF <- function(xin, yin, p){
  ######## Funcao 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] # numero de amostras
  n<-dim(xin)[2] # dimensao 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 funcoes:
  m <- as.matrix (xclust$centers)
  covlist <- list()

  # Estima matrizes de covariancia 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 nao havera erro (caso tenha apenas 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))
}
```

Listing 7: Função de treinamento da rede RBF em R

3.8 Resposta da Rede RBF

```
# Funcao que calcula a saida de uma rede RBF.
library("corpcor")

YRBF <- function(xin, modRBF){
  ######## Funcao 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] # numero de amostras
  n <- dim(xin)[2] # dimensao de entrada (deve ser maior que 1)
  m <- as.matrix(modRBF[[1]])
  covlist <- modRBF[[2]]
  p <- length(covlist) # Numero de funcoes radiais
  W <- modRBF [[3]]

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

```
24
25   H <- matrix(nrow = N, ncol = p)
26   # Calcula matriz H
27   for (j in 1:N) {
28     for (i in 1:p) {
29       mi <- m[i, ]
30       covi <- covlist[i]
31       covi <- matrix(unlist(covlist[i]), ncol = n, byrow = T) + 0.001 *
             diag(n)
32       H[j,i] <- pdfnvar(xin[j, ], mi, covi, n)
33     }
34   }
35
36   Haug <- cbind(1, H)
37   Yhat <- Haug %*% W
38   return(Yhat)
39 }
```

Listing 8: Função que calcula a resposta de uma rede RBF em R

3.9 Classificador Utilizando uma Rede RBF

```
1  rm(list=ls())
2  source("~/Documents/UFMG/9/Redes Neurais/TP1/prediction-competition-
       accent-recognition/RBF_kmeans/trainRBF.R")
3  source("~/Documents/UFMG/9/Redes Neurais/TP1/prediction-competition-
       accent-recognition/RBF_kmeans/YRBF.R")
4  source("~/Documents/UFMG/9/Redes Neurais/exemplos/escalonamento_matrix.
       R")
5  library(caret)
6
7  # Carregando base de dados:
8  path <- file.path("~/Documents/UFMG/9/Redes Neurais/TP1/prediction-
       competition-accent-recognition/databases", "treino.csv")
9  data_train <- read.csv(path)
10 path <- file.path("~/Documents/UFMG/9/Redes Neurais/TP1/prediction-
       competition-accent-recognition/databases", "teste.csv")
11 data_test <- read.csv(path)
12
13 # Separando dados de entrada e saida e treino e teste:
14 x_train <- as.matrix(data_train[1:53, 2:13])
15 y_train <- as.matrix(data_train[1:53, 14])
16 x_test <- as.matrix(data_test[1:22, 2:13])
17
18 # Escalonando os valores dos atributos para que fiquem restritos entre
       0 e 1
19 x_all <- rbind(x_train, x_test)
20 x_all <- staggeringMatrix(x_all, nrow(x_all), ncol(x_all))
```

```
x_train <- x_all[1:53, ]
x_test <- x_all[54:75, ]

p <- 5 # numero de neuronios
executions <- 31
results <- matrix(nrow = nrow(x_test), ncol = executions)
for (index in 1:executions){
  # Treinando modelo:
  modRBF<-trainRBF(x_train, y_train, p)

  # Calculando acuracia de treinamento
  length_train <- length(y_train)
  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<-((sum(abs(yt + y_train)))/2)/length_train
  print(accuracy_train)

  # Rodando dados 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
  results[,index] <- yt
}

y <- rep(0, 22)
for (index in 1:22) {
  if(sum(results[index,] == 1) > (executions/2)){
    y[index] <- 1
  }
  else{
    y[index] <- -1
  }
}

Id <- 54:75
table <- data.frame(Id, y)
write.csv(table, "prediction_rbf5.csv", row.names = FALSE)
```

Listing 9: Classificador para o problema proposto, utilizando uma rede RBF em R

3.10 Escalonamento dos Dados

```
# Funcao que recebe uma matriz e suas dimensoes e retorna uma matriz
# de mesma dimensao porem com sua colunas escalonadas.
staggeringMatrix <- function(matrix, rows, columns ){
  staggeredMatrix <- matrix(rep(0, rows*columns), ncol = columns, nrow
      = rows)
```

```
5   # Escalonando dados:
6   for (j in 1:columns) {
7     for (i in 1:rows) {
8       staggeredMatrix[i,j] <- (matrix[i,j] - min(matrix[,j])) / (max(
            matrix[,j]) - min(matrix[,j]))
9     }
10  }
11  return(staggeredMatrix)
12 }
```

Listing 10: Função para escalonamento dos dados de uma matriz em R