Redes Neurais Artificiais

Exercício 4 - Perceptron

Vítor Gabriel Reis Caitité - 2021712430

11/11/2021

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)
}
```

Questão 1

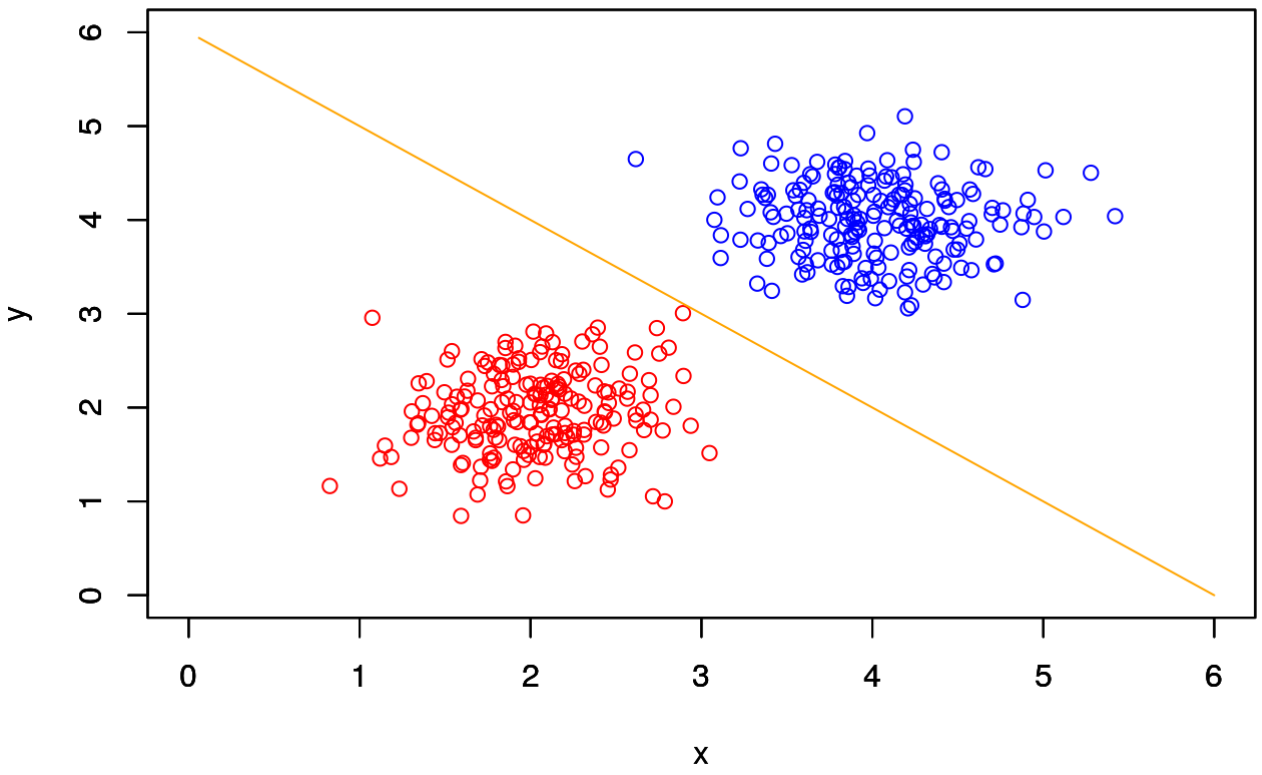
Inicialmente, devem-se amostrar duas distribuições normais no espaço R^2 , ou seja, duas distribuições com duas variáveis cada (Ex: x1 e x2). As distribuição são caracterizadas como N (2, 2, σ^2) e N (4, 4, σ^2). Nesta atividade o aluno irá fazer o treinamento do perceptron afim de encontrar o vetor de pesos w e encontrar a superfície de separação.

Resolução:

Geração de ditribuições normais

Inicialmente, foram amostradas as amostras como requisitado. As distribuições são caracterizadas como N(2, 2, σ^2) e N(4, 4, σ^2).

```
rm(list = ls())

library("plot3D")
source("~/Documents/UFMG/Mastering/1/RNA/Listas/2 - Perceptron/trainPerceptron.R")
source("~/Documents/UFMG/Mastering/1/RNA/Listas/2 - Perceptron/yperceptron.R")

s1<-0.4
s2<-0.4
nc<-200

# Distribuição normal N(2, 2, σ^2)
xc1<-matrix(rnorm(nc*2), ncol=2)*s1 + t(matrix((c(2,2)), ncol=nc, nrow=2))
# Distribuição normal N(4, 4, σ^2)
xc2<-matrix(rnorm(nc*2), ncol=2)*s2 + t(matrix((c(4,4)), ncol=nc, nrow=2))

# Plotando dados:
plot(xc1[,1], xc1[,2], col = "red", xlim=c(0,6), ylim=c(0,6), ylab="y", xlab="x")
par(new=T)
plot(xc2[,1], xc2[,2], col = "blue", xlim=c(0,6), ylim=c(0,6), ylab="y", xlab="x")

# Reta que separa os dados (inferida visualmente)
x1_reta<-seq(6/100,6,0.01)
x2_reta<- -x1_reta+6
par(new=TRUE)
plot(x1_reta, x2_reta, type="l", col="orange", xlim=c(0,6), ylim=c(0,6), ylab="", xla
b="")
```



Treinamento do modelo

Geradas as amostras, o próximo passo foi o treinamento do modelo, ou seja encontrar o vetor de pesos w:

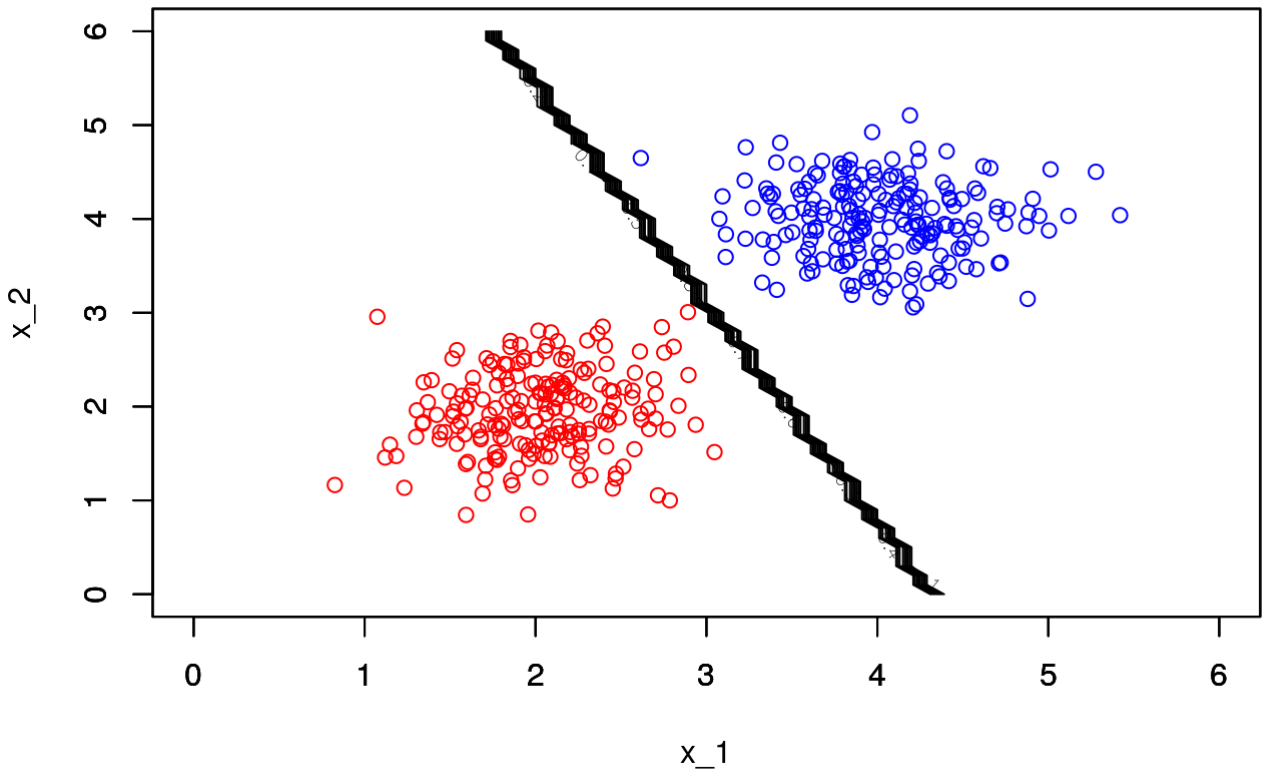
```
# Treinando modelo:

# Definindo entradas da função (Nesse ex. todos os dados etão sendo usados para trein
o):
xin = as.matrix(rbind(xc1,xc2))
yc1_train<-matrix(0, nrow=nc)
yc2_train<-matrix(1, nrow=nc )
yp<-as.matrix(rbind(yc1_train, yc2_train))
retlist<-trainPerceptron(xin, yp, 0.1, 0.01, 100, 1) #função que faz o treinamento do
 perceptron
w<-retlist[[1]]
print(w)
```

```
##            [,1]
## [1,] -1.3467856
## [2,]  0.3117834
## [3,]  0.1331907
```

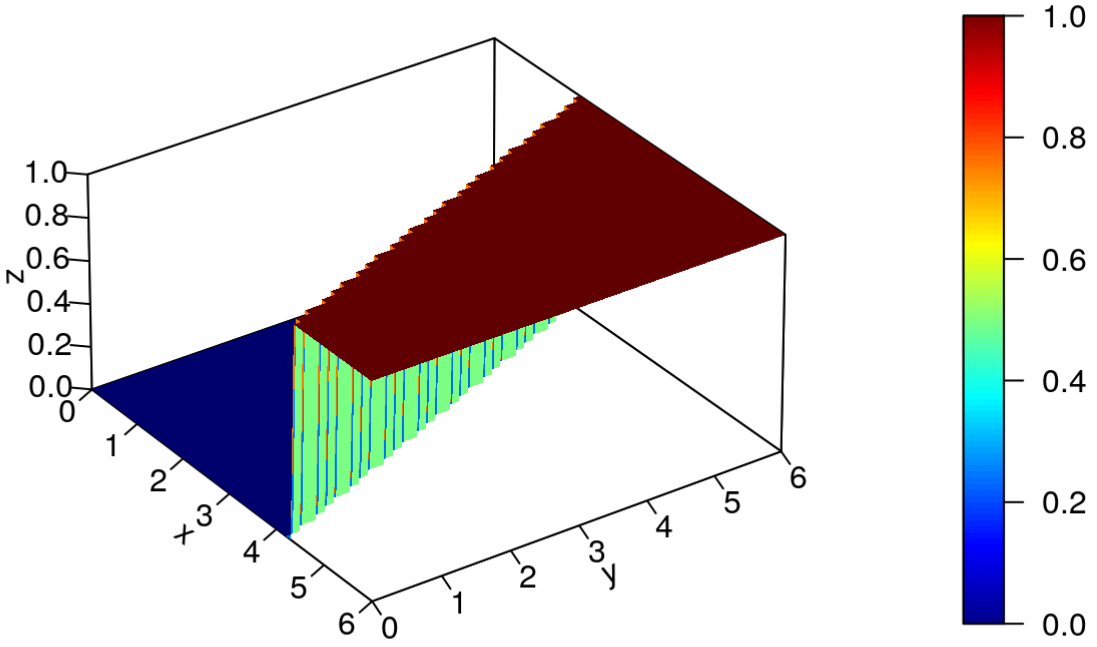
Teste do modelo

Por fim foi realizado o teste do modelo e a geração dos plots requisitados.

```
# Teste do modelo:
# Pontos de teste
seqi<-seq(0,6,0.1)
seqj<-seq(0,6,0.1)

# Cálculo do perceptron para pontos de teste
M<-matrix(0, nrow=length(seqi), ncol=length(seqj))
ci<-0
for(i in seqi) {
  ci<-ci+1
  cj<-0
  for(j in seqj) {
    cj<-cj+1
    x<-matrix(c(i,j), nrow = 1, ncol = 2)
    M[ci,cj]<-yperceptron(x,w,1)
  }
}

# Plotando resultados:
plot(xc1[,1], xc1[,2], xlim=c(0,6), ylim=c(0,6), col="red", ylab="x_2", xlab="x_1")
par(new=TRUE)
plot(xc2[,1], xc2[,2], xlim=c(0,6), ylim=c(0,6), col="blue",  ylab="", xlab="")
par(new=TRUE)
contour(seqi, seqj, M, xlim=c(0,6), ylim=c(0,6), ylab="", xlab="")
```




```
persp3D(seqi, seqj, M, counter=T, theta=55, phi=30, r=40, d=0.1, expand=0.5, ltheta=9
0, lphi=180, shade=0.4, ticktype="detailed", nticks=5)
```



Questão 2

Nesta segunda atividade o aluno deverá criar um conjunto de amostras de cada uma das duas distribuições do Exercício 1, ou seja, 200 amostras da classe 1 e 200 amostras da classe 2. O aluno deverá utilizar essas amostras para criar dois conjuntos balanceados, um chamado de conjunto de treinamento que será usado para achar o pesos w e outro chamado de teste que servirá para avaliar a performance do seu separador dado pelos pesos encontrados no treinamento. O conjunto de treinamento irá conter 70% da amostras e o de teste 30%. Essa distribuição deve ser obrigatoriamente aleatória. Após a separação dos dois conjuntos o aluno usará o conjunto de treinamento para encontrar os pesos do perceptron e utilizará o conjunto de teste para avaliar a performance do perceptron simples. Apresente a acurácia e a matriz de confusão.

Resolução:

Separando amostras em amostras de treinamento e teste

```
rm(list = ls())

library("plot3D")
library("bnlearn")
library("caret") #Para plotar matriz de confusão
```

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

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

```
source("~/Documents/UFMG/Mastering/1/RNA/Listas/2 - Perceptron/trainPerceptron.R")
source("~/Documents/UFMG/Mastering/1/RNA/Listas/2 - Perceptron/yperceptron.R")


#_____________________________________________________________________________
#Gerando dados:
# Distribuição normal N(2, 2, σ^2)
xc1<-matrix(rnorm(200*2), ncol=2)*0.4 + t(matrix((c(2,2)), ncol=200, nrow=2))
# Distribuição normal N(4, 4, σ^2)
xc2<-matrix(rnorm(200*2), ncol=2)*0.4 + t(matrix((c(4,4)), ncol=200, nrow=2))

#_____________________________________________________________________________
# Separando em conjunto de treinamento e conjunto de teste:

# Seleção das amostra de Treinamento:
ntrain<-200*0.7 # Número de amostras, de cada uma das classes, selecionadas para trei
namento.
seqc1<-sample(200) #Gera um vetor com números int de 1 a 200 em posições aleatórias
xc1_train<-xc1[seqc1[1:ntrain],] # 140 amostras de treino
yc1_train<-matrix(0, nrow=ntrain) # 0 - amostras distribuidas em torno de 2
seqc2<-sample(200) #Gera um vetor com números int de 1 a 50 em posições aleatórias
xc2_train<-xc2[seqc2[1:ntrain],] # 140 amostras de treino
yc2_train<-matrix(1, nrow=ntrain) # 1 - amostras distribuídas em trono de 4

# Seleção das amostra de Teste:
xc1_test<-xc1[seqc1[(ntrain+1):200],] # 60 amostras de teste
yc1_test<-matrix(0, (nrow=200-ntrain)) # 0 - amostras distribuidas em torno de 2
xc2_test<-xc2[seqc2[(ntrain+1):200],] # 60 amostras de teste
yc2_test<-matrix(1, (nrow=200-ntrain)) # 1 - amostras distribuídas em trono de 4

#Concatenando dados das 2 classes
xin_train<-as.matrix(rbind(xc1_train, xc2_train))
yd_train<-rbind(yc1_train, yc2_train)
xin_test<-as.matrix(rbind(xc1_test, xc2_test))
yd_test<-rbind(yc1_test, yc2_test)
```

Treinamento do modelo

Abaixo pode-se observar o vetor de pesos w.

```
#_____________________________________________________________________________
# Treinamento:

retlist<-trainPerceptron(xin_train, yd_train, 0.1, 0.01, 100, 1)
wt<-retlist[[1]]
print(wt)
```

```
##               [,1]
## [1,] -0.5693431611
## [2,]  0.2004138797
## [3,]  0.0005088706
```

Teste da performance do perceptron

Nesse passo foi utilizado o conjunto de teste para avaliar a performance do perceptron simples. Apresentou-se, abaixo, a acurácia, a matriz de confusão. entre outros dados.

```
#_____________________________________________________________________________
# Teste:

yt<-yperceptron(xin_test,wt,1)
acc<-1-(t(yd_test-yt)%*%(yd_test-yt))/60
paste("Acurácia: ", acc*100, '%')
```

```
## [1] "Acurácia:  100 %"
```

```
print(confusionMatrix(table(yt, yd_test)))
```

```
## Confusion Matrix and Statistics
##
##    yd_test
## yt   0  1
##   0 60  0
##   1  0 60
##
##                Accuracy : 1
##                  95% CI : (0.9697, 1)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0
##             Specificity : 1.0
##          Pos Pred Value : 1.0
##          Neg Pred Value : 1.0
##              Prevalence : 0.5
##          Detection Rate : 0.5
##    Detection Prevalence : 0.5
##       Balanced Accuracy : 1.0
##
##        'Positive' Class : 0
##
```

Questão 3

No Exercício 3 iremos trabalhar com uma base de dados conhecida como Iris (comando: data(”iris”)). Essa base de dados possui 150 amostras e 4 características, sendo 50 para cada uma das três espécies de plantas que constitui a base. Nesta atividade o aluno irá realizar o treinamento do perceptron para separar a espécie 1 (50 primeiras amostras) das outras duas espécies e avaliar o desempenho do mesmo. Com isso a espécie 1 será a Classe 1 e o conjunto das espécies 2 e 3 será a Classe 2. O aluno deverá então:

1. Importar as funções yperceptron e trainperceptron desenvolvida por ele em sala de aula.

```
#Limpando ambiente e importando funções
rm(list=ls())
source("~/Documents/UFMG/Mastering/1/RNA/Listas/2 - Perceptron/trainPerceptron.R")
source("~/Documents/UFMG/Mastering/1/RNA/Listas/2 - Perceptron/yperceptron.R")
```

2. Carregar os dados da Iris e armazená-los, sendo que a Classe 1 será composta das 50 primeiras amostras e a Classe 2 das 100 amostras posteriores as 50 primeiras, como descrito na introdução do problema.

```
#Carregando dados
data(iris)
```

3. e 4. Rotular as amostras da Classe 1 com o valor de 0 e as amostras da Classe 2 com o valor 1. Selecionar aleatoriamente 70% da amostras para o conjunto de treinamento e 30% para o conjunto de teste, para cada uma das duas classes.

```
# Como o problema da Iris tem 3 classes e, no momento, estamos estudando
# classificadores binários, o problema  será tratado como um problema de
# duas classes para a sua utilização com o Perceptron Simples. Assim, ao invés
# de discriminar as 3 classes, o nosso problema será aqui o de discriminar a classe
# setosa da classes vesicolor e  virginica. A classe 1 do nosso problema será, então,
 caracterizada
# pelas 50 primeiras amostras e a classe 2 pelas 100 amostras seguintes,
# armazenadas nas matrizes xc1 e xc2, conforme linhas de código a seguir.
xc1<-as.matrix(iris[1:50,1:4]) # setosa
xc2<-as.matrix(iris[51:150,1:4]) # vesicolor e  virginica

# Seleção das amostra de Treinamento:
ntrain_class1<-50*0.7 # Número de amostras da classe 1 selecionadas para treinamento.
seqc1<-sample(50) #Gera um vetor com números int de 1 a 50 em posições aleatórias
xc1_train<-xc1[seqc1[1:ntrain_class1],] # 35 amostras de treino
yc1_train<-matrix(0, nrow=ntrain_class1) # 0 - setosa
ntrain_class2<-100*0.7 # Número de amostras da classe 1 selecionadas para treinament
o.
seqc2<-sample(100) #Gera um vetor com números int de 1 a 100 em posições aleatórias
xc2_train<-xc2[seqc2[1:ntrain_class2],] # 70 amostras de treino
yc2_train<-matrix(1, nrow=ntrain_class2) # 1 - vesicolor e virginica

# Seleção das amostra de Teste:
xc1_test<-xc1[seqc1[(ntrain_class1+1):50],] # 15 amostras de teste
yc1_test<-matrix(0, (nrow=50-ntrain_class1)) # 0 - setosa
xc2_test<-xc2[seqc2[(ntrain_class2+1):100],] # 35 amostras de teste
yc2_test<-matrix(1, (nrow=100-ntrain_class2)) # 1 - vesicolor e virginica

#Concatenando dados das 2 classes
xin_train<-as.matrix(rbind(xc1_train, xc2_train))
yd_train<-rbind(yc1_train, yc2_train)
```

5. e 6. Utilizar as amostras de treinamento para fazer o treinamento do perceptron utilizando a função trainperceptron. Extrair o vetor de pesos da função trainperceptron.

```
#Treinamento:
retlist<-trainPerceptron(xin_train, yd_train, 0.1, 0.01, 100, 1)
wt<-retlist[[1]]
print(wt)
```

```
##            [,1]
## [1,] -0.5218553
## [2,] -0.1896537
## [3,] -0.1817353
## [4,]  0.6695992
## [5,]  0.2582623
```

7. Concatenar as amostras de teste e seus respectivos y e dar entrada na função yperceptron (a função yperceptron não recebe o y), utilizando o vetor de peso extraído.

```
#Concatenando dados das 2 classes
xin_test<-as.matrix(rbind(xc1_test, xc2_test))
yd_test<-rbind(yc1_test, yc2_test)

#Teste:
yt<-yperceptron(xin_test,wt,1)
```

8. Calcular o erro percentual. (O erro é dado pelo número de amostras de teste classificadas de forma errada).

```
acc<-1-(t(yd_test-yt) %*% (yd_test-yt) )/(50 - ntrain_class1 + 100 - ntrain_class2)
paste("Acurácia:", acc*100, '%')
```

```
## [1] "Acurácia: 100 %"
```

```
error=1-acc
paste("Erro %:", error*100 , '%')
```

```
## [1] "Erro %: 0 %"
```

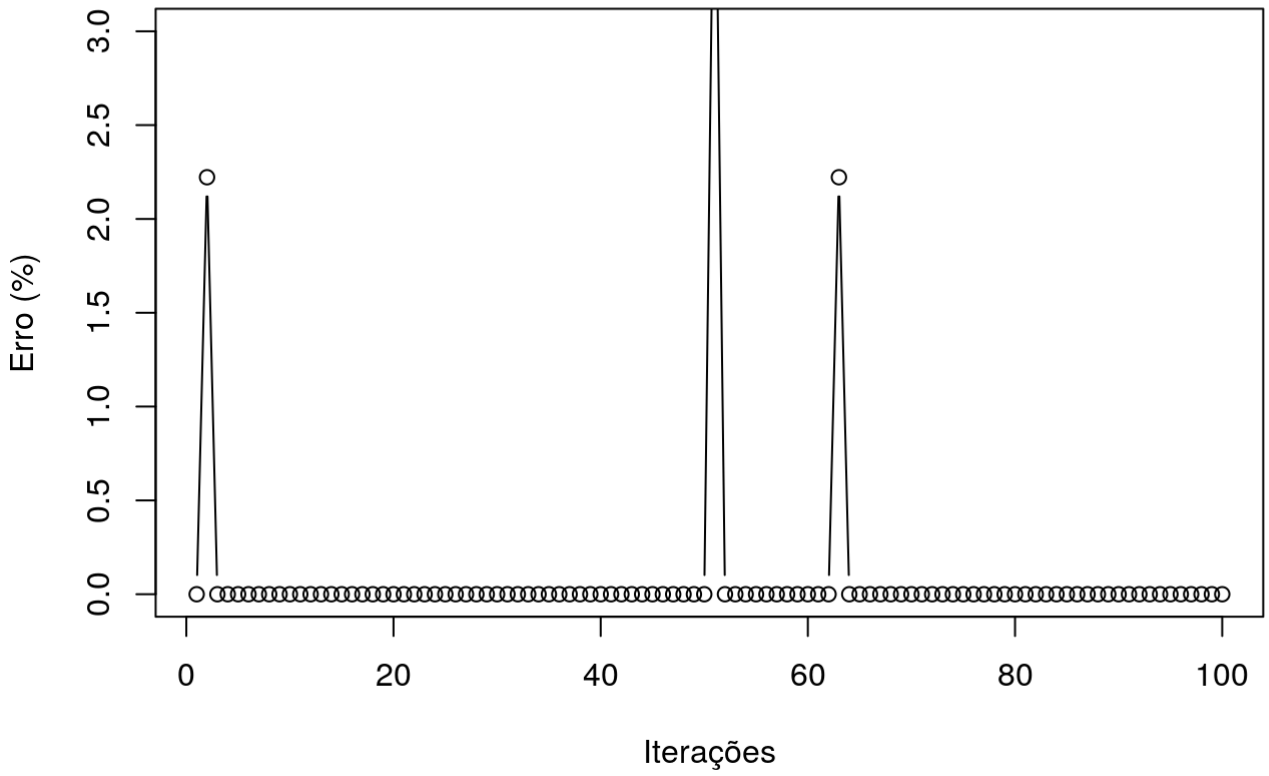
9. Imprimir a matriz de confusão.

```
print(confusionMatrix(table(yt, yd_test)))
```

```
## Confusion Matrix and Statistics
##
##    yd_test
## yt   0  1
##   0 15  0
##   1  0 30
##
##                Accuracy : 1
##                  95% CI : (0.9213, 1)
##     No Information Rate : 0.6667
##     P-Value [Acc > NIR] : 1.191e-08
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.3333
##          Detection Rate : 0.3333
##    Detection Prevalence : 0.3333
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : 0
##
```

10. Crie um loop para repetir 100 vezes os itens 4-8, armazenando o valor do erro percentual do item 8. Plote o erro percentual em função do número de iteração e imprima o valor da variância do erro.

```
error<-rep(0,100)

for(count in 1:100){
  # Seleção das amostra de Treinamento:
  ntrain_class1<-50*0.7 # Número de amostras da classe 1 selecionadas para treinament
o.
  seqc1<-sample(50) #Gera um vetor com números int de 1 a 50 em posições aleatórias
  xc1_train<-xc1[seqc1[1:ntrain_class1],] # 35 amostras de treino
  yc1_train<-matrix(0, nrow=ntrain_class1) # 0 - setosa
  ntrain_class2<-100*0.7 # Número de amostras da classe 1 selecionadas para treinamen
to.
  seqc2<-sample(100) #Gera um vetor com números int de 1 a 100 em posições aleatórias
  xc2_train<-xc2[seqc2[1:ntrain_class2],] # 70 amostras de treino
  yc2_train<-matrix(1, nrow=ntrain_class2) # 1 - vesicolor e virginica

  # Seleção das amostra de Teste:
  xc1_test<-xc1[seqc1[(ntrain_class1+1):50],] # 15 amostras de teste
  yc1_test<-matrix(0, (nrow=50-ntrain_class1)) # 0 - setosa
  xc2_test<-xc2[seqc2[(ntrain_class2+1):100],] # 30 amostras de teste
  yc2_test<-matrix(1, (nrow=100-ntrain_class2)) # 1 - vesicolor e virginica

  #Concatenando dados das 2 classes
  xin_train<-as.matrix(rbind(xc1_train, xc2_train))
  yd_train<-rbind(yc1_train, yc2_train)

  #Treinamento:
  retlist<-trainPerceptron(xin_train, yd_train, 0.1, 0.01, 100, 1)
  wt<-retlist[[1]]

  #Concatenando dados das 2 classes
  xin_test<-as.matrix(rbind(xc1_test, xc2_test))
  yd_test<-rbind(yc1_test, yc2_test)

  #Teste:
  yt<-yperceptron(xin_test,wt,1)
  acc<-1-(t(yd_test-yt) %*% (yd_test-yt) )/(50 - ntrain_class1 + 100 - ntrain_class2)

  error[count]=(1-acc)*100

}

plot(seq(1,100,1), error, type="b", xlim = c(1,100), ylim = c(0,3), ylab = "Erro (%)"
, xlab = "Iterações")
```




```
var_error<-var(error)
paste("A variãncia do erro foi de: ", var_error)
```

```
## [1] "A variãncia do erro foi de:  0.291308143159995"
```