Redes Neurais Artificiais

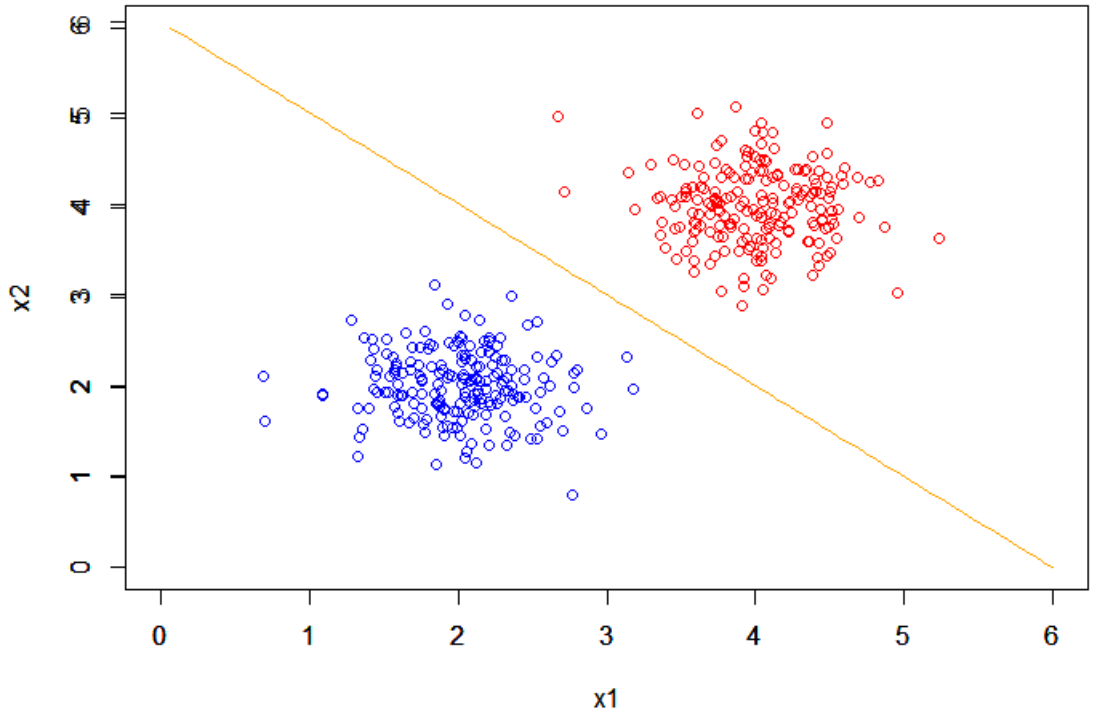
André Costa Werneck, Matrícula: 2017088140

LISTA 4

27/04/2022

1) Ao realizar a leitura do presente exercício, conclui-se que ele apresenta o objetivo de realizar a implementação do modelo do Perceptron e, com ele, providenciar a familiarização dos alunos com seu algoritmo. Observou-se que o modelo presente é muito semelhante ao do Adaline, se diferindo apenas no fato de que não se usou uma função de ativação identidade e sim uma função de limiar, no caso com limiar na forma de degrau.

Desta forma, gerou-se a base os dados como sugerido pelo professor através das figuras, implementou-se e treinou-se o modelo. Vale ressaltar que, diferentemente do exercício feito para o Adaline, no caso do Perceptron, rotulou-se os dados separando-os em duas classes, com rótulos 0 e 1, respectivamente, para fazer jus ao problema de classificação e dar sentido ao limiar que caracteriza o modelo do Perceptron.



Os pesos obtidos foram os que seguem:

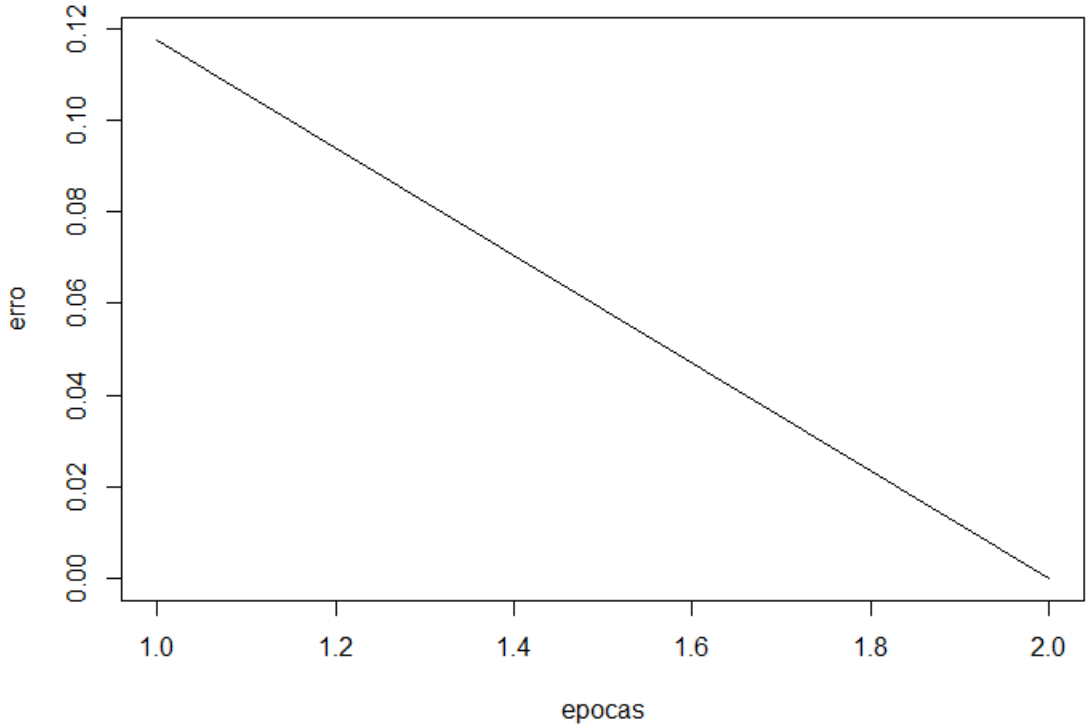
[,1]

[1,] 0.2401009

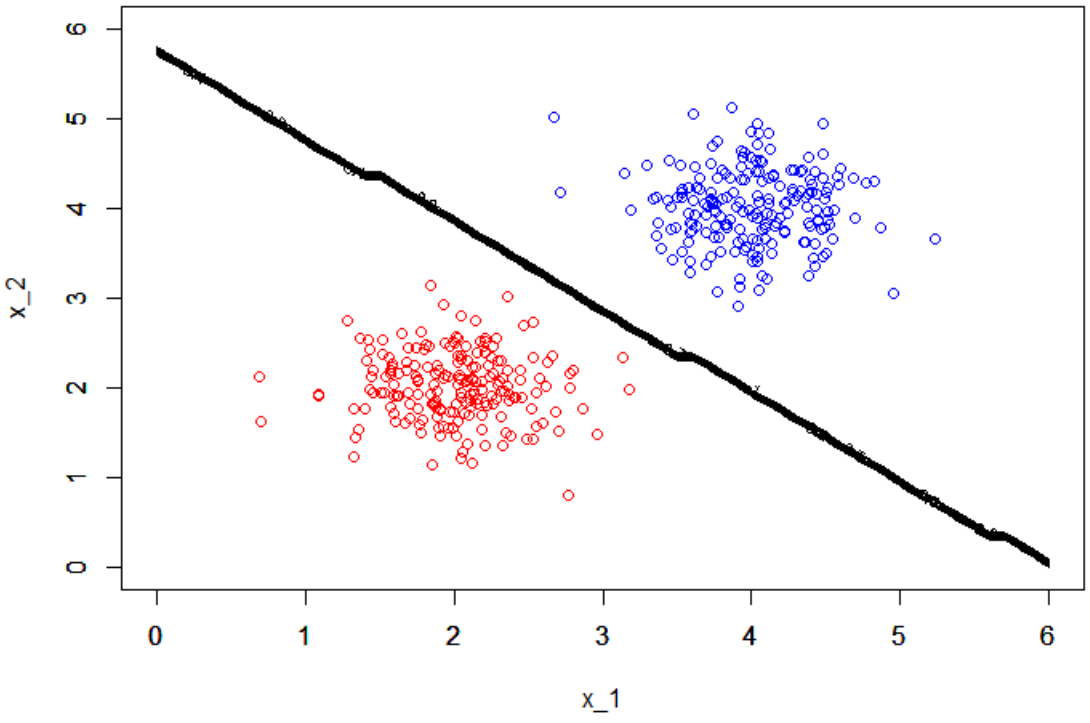
[2,] 0.2521493

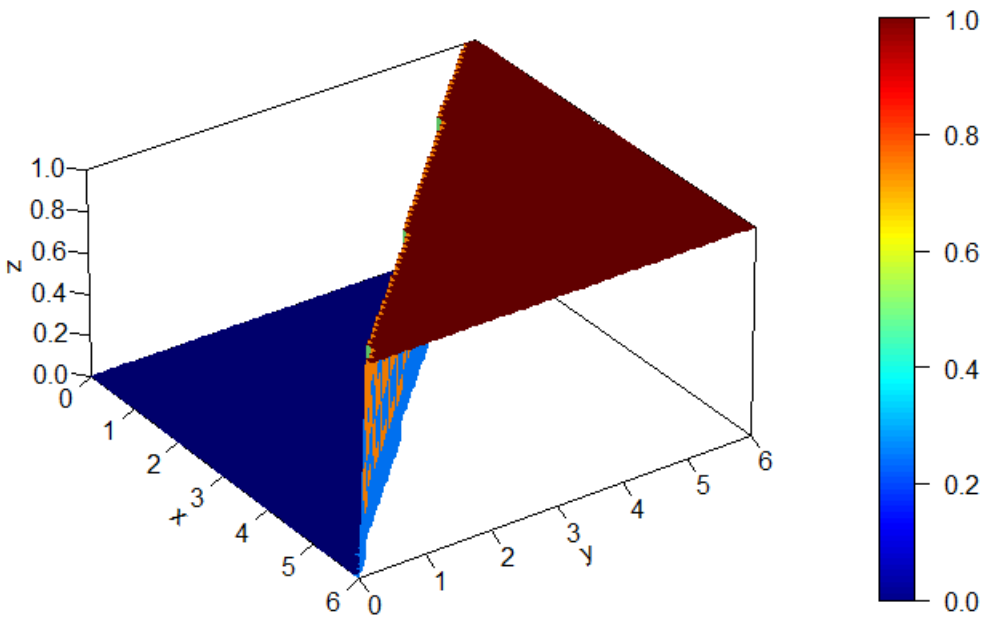
[3,] 1.4450100

Por ser um problema simples, de fácil resolução, o erro convergiu em apenas 2 épocas para o mínimo esperado e, assim, atingiu a condição de parada do modelo.



Repetindo o teste para um novo conjunto de dados bem semelhante, mas com mais amostras, observou-se um bom desempenho do modelo, novamente ilustrado pelas seguintes curvas:





2) Para o exercício 2, utilizou-se a mesma implementação do Perceptron e gerou-se o conjunto de dados como sugerido no exercício. Após separá-los, também como pedido, em conjunto de treino e teste, rotulou-se a base e treinou-se o modelo. Os pesos obtidos foram:

[,1]

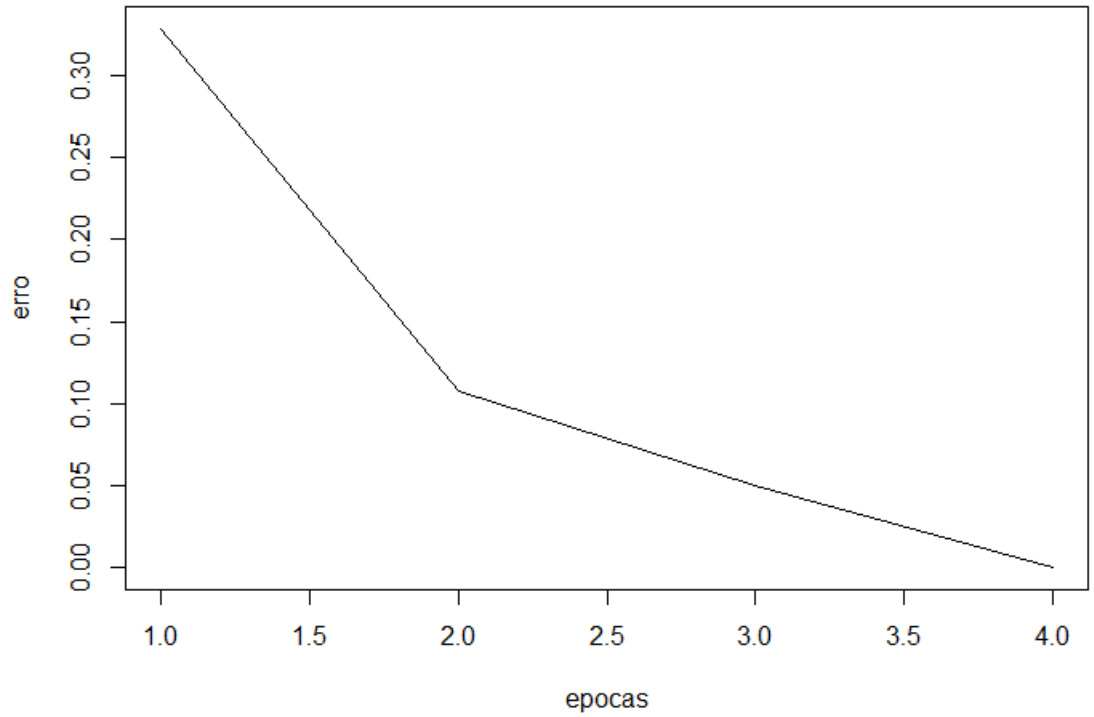
[1,] 0.02394425

[2,] 0.07903310

[3,] 0.29409350

Nos quais o termo de polarização é o 3º.

Dessa vez, o erro convergiu em 4 épocas até atingir a condição de parada do Perceptron implementado. De qualquer forma, o modelo foi bastante eficiente e rápido.



Após treinar o modelo e obter os parâmetros mostrados acima, testou-se o conjunto de dados previamente separado e calculou-se tanto a acurácia quanto a matriz de confusão para ambos os conjuntos de treino e teste. A acurácia e a matriz de confusão de treino ficaram como segue:

```
   [,1]
[1,]   1
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 70  0
         1  0 70

               Accuracy : 1
                 95% CI : (0.974, 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
```

E as de teste:

```
     [,1]
[1,]    1

Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 30  0
         1  0 30

               Accuracy : 1
                 95% CI : (0.9404, 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

Dessa forma, constatou-se que o modelo conseguiu separar o conjunto de dados 100% corretamente, representando uma ótima solução para o problema. Vale ressaltar que a eficiência do Perceptron e a boa resposta encontrada têm relação com o fato de que o problema era simples e de fácil resolução para o presente classificador. Veremos isso mais claramente com os próximos exercícios.

3) Para o exercício 3, estudou-se uma base de maior dimensão, a da Iris, já apresentada anteriormente nas notas de aula. Ela é padrão do R e foi carregada como código em Anexo. Após ser rotulada e separada em conjuntos de treino e teste, como sugerido no passo a passo apresentado, treinou-se o modelo.

Interessantemente, notou-se que o modelo apresentou maior dificuldade para aprender as características apresentadas e classificar corretamente o conjunto de dados amostrado. Ao se treinar poucas vezes o Perceptron, obteve-se uma acurácia de treino como segue:

```
        [,1]
[1,] 0.3238095
```

Entretanto, como ensinado pelo professor, em casos de maior dimensão e de problemas mais complexos, a estocasticidade dos modelos aumenta e é necessário treiná-los várias vezes e obter um conjunto de pesos médios de treinamento, para tentar eliminar essa aleatoriedade. Fez-se isso e o resultado foi incrível. Os pesos obtidos foram:

```
         [,1]
[1,] -0.1499081
[2,] -0.4757700
[3,]  0.7393348
[4,]  0.3213300
[5,]  0.1395204
```

A acurácia de teste e sua matriz de confusão foram:

```
    [,1]
[1,]    1
```

Confusion Matrix and Statistics

```
          Reference
Prediction  0  1
         0 17  0
         1  0 28

               Accuracy : 1
```

95% CI : (0.9213, 1)
No Information Rate : 0.6222
P-Value [Acc > NIR] : 5.34e-10

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 : 0.3778
Detection Rate : 0.3778
Detection Prevalence : 0.3778
Balanced Accuracy : 1.0000

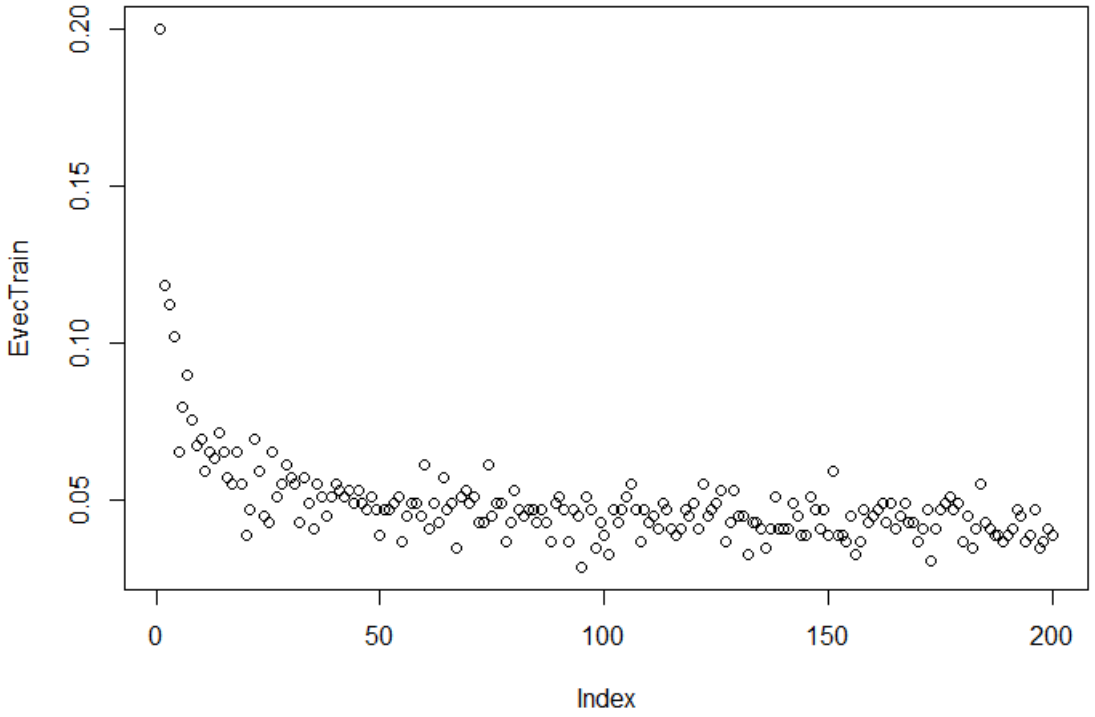
'Positive' Class : 0

Dessa forma, observou-se claramente o relatado pelo Professor Antônio e concluiu-se com o problema foi bem aprendido pelo modelo e a solução apresentada foi excelente, com boa capacidade de generalização para a base de dados apresentada.

4) Para o exercício de número 4, foram realizados os mesmo passos do ex3, mas com a diferença de que a base de dados utilizada era a da Breast Cancer e possuía 9 entradas, 5 a mais do que a do problema 3. Após seguir corretamente o passo a passo aprendido na atividade anterior, treinou-se e testou-se o modelo, com diferentes números de épocas, inúmeros testes nos hiper parâmetros para tentar aumentar a convergência e a tolerância do modelo e com números diferentes de repetições visando eliminar uma possível falha por estocasticidade do resultado.

Entretanto, esse modelo apresentou complexidade bem maior do que os anteriormente vistos e, por isso, não convergiu. Os resultados de saída não foram bem distinguidos e muito menos bem classificados, resultando em um conjunto apenas de 1 e com acurácia pouco acima dos 30% tanto para o conjunto de treino quanto para o de teste. Pode ter ocorrido alguma falha na construção do conjunto de dados ou não, mas é fato que o modelo implementado não foi um bom solvente para o problema do Breast Cancer.

Vale ressaltar que observou-se uma curva do erro de treinamento e foi vista uma convergência, mas que foi insuficiente uma vez que o erro continuou acima do limiar de parada (e isso se manteve mesmo aumentando o número de épocas para 1000). Como segue:



O resultado de acurácia é como segue:

```
        [,1]
[1,] 0.3588517
```

Vale dizer também que o modelo demorou bastante para treinar o conjunto de dados e isso é um ilustrador do maior tamanho, da maior complexidade do problema e, portanto, da maior dificuldade apresentada para aprendizado.

ANEXOS

O código para realização dos exercícios 1, 2, 3 e 4 se encontra devidamente documentado, através dos comentários, em anexo:

```
rm(list = ls())
library('plot3D')

treinap <- function(xin,yin,eta,tol_parada,maxepocas){

  # pegando as dimensoes dos dados de entrada
  N<-dim(xin)[1]
  n<-dim(xin)[2]

  # adicionando coluna de 1s ao vetor de entrada
```

```
  xin<-cbind(xin,-1)
  # inicializando o vetor de pesos aleatoriamente
  w<-as.matrix(runif(n+1)-0.5)
  erroepoca<-tol_parada+10
  epocaatual<-0
  vetorEpocas <- matrix(nrow = 1,ncol = maxepocas)

  while((erroepoca>tol_parada)&&(epocaatual<maxepocas)) {
    indexvec = sample(N)
    erroQuad<-0
    for (i in 1:N) {
      #add estocasticidade ao treinamento
      index<-indexvec[i]
      #acha yhat
      yhat<-1*((xin[index,]%*%w)>=0)
      #calcula o erro
      erroatual<-yin[index,]-yhat
      w = w + eta*erroatual*xin[index,]
      erroQuad <- erroQuad + erroatual*erroatual
    }
    epocaatual <- epocaatual + 1 # atualiza as epocas
    vetorEpocas[epocaatual]<-erroQuad/N # erro medio de cada época
    erroepoca <- vetorEpocas[epocaatual]
  }
  return(list(w,vetorEpocas[1:epocaatual]))
}

# EX1 ----
# gerando os dados
x1 <- matrix(0.4*rnorm(200*2),ncol=2) + t(matrix((c(2,2)),ncol = 200,nrow = 2))
x2 <- matrix(0.4*rnorm(200*2),ncol=2) + t(matrix((c(4,4)),ncol = 200,nrow = 2))

# rotulando os dados
y1 <- matrix(0,nrow = dim(x1)[1])
y2 <- matrix(1,nrow = dim(x2)[1])

# agrupando os dados para chamar a funcao de treino do perceptron
```

```
xin <- rbind(x1,x2)
yin <- rbind(y1,y2)

# Plotando os dados

plot(x1[,1],x1[,2],xlim=c(0,6),ylim = c(0,6),xlab ='x1',ylab = 'x2',col='blue')
par(new=TRUE)
plot(x2[,1],x2[,2],xlim=c(0,6),ylim = c(0,6),xlab ='x1',ylab = 'x2',col='red')

x1_reta <- seq(6/100,6,6/100)
x2_reta <- -x1_reta+6
par(new=TRUE)
plot(x1_reta,x2_reta,type = 'l',col='orange',xlim = c(0,6),xlab = '',ylab = '')

ret <- treinap(xin,yin,0.1,0.01,100)

w <- ret[[1]]
evec <- ret[[2]]

#print(w)

yperceptron <-function(xvec,w){
  u<-xvec %*% w
  y<-1.0*(u>=0)

  return(as.matrix(y))
}

seqi <- seq(0,6,0.1)
seqj <- seq(0,6,0.1)

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<-as.matrix(cbind(i,j,-1))
    M[ci,cj]<- yperceptron(x,w)

  }

}

plot(x1[,1],x1[,2],col = 'red', xlim = c(0,6), ylim = c(0,6),xlab = 'x_1',ylab = 'x_2')
par(new=T)
plot(x2[,1],x2[,2],col = 'blue', xlim = c(0,6), ylim = c(0,6),xlab = '',ylab = '')
par(new=T)
contour(seqi,seqj,M,xlim = c(0,6),ylim = c(0,6),xlab= '', ylab='')

persp3D(seqi,seqj,M,counter                                                    =
T,theta=55,phi=30,r=40,d=0.1,expand=0.5,ltheta=90,lphi=180,shade=0.4,ticktype='de
tailed',nticks=5)

# EX2 -----
rm(list = ls())
library('caret')

# gerando os dados
amostras <- 200
xc1 <- matrix(0.4*rnorm(amostras),ncol=2) + t(matrix((c(2,2)),ncol = 100,nrow = 2))
xc2 <- matrix(0.4*rnorm(amostras),ncol=2) + t(matrix((c(4,4)),ncol = 100,nrow = 2))

xc1treino <- xc1[1:(0.7*(amostras/ncol(xc1))),]
xc2treino <- xc2[1:(0.7*(amostras/ncol(xc2))),]

xc1teste <- xc1[((0.7*(amostras/ncol(xc1)))+1):(amostras/ncol(xc1)),]
xc2teste <- xc2[((0.7*(amostras/ncol(xc2)))+1):(amostras/ncol(xc2)),]

y1treino <- matrix(0,nrow = dim(xc1treino)[1])
y2treino <- matrix(1,nrow = dim(xc2treino)[1])

y1teste <- matrix(0,nrow = dim(xc1teste)[1])
y2teste <- matrix(1,nrow = dim(xc2teste)[1])
```

```
xinTreino <- rbind(xc1treino,xc2treino)
yinTreino <- rbind(y1treino,y2treino)

treinap <- function(xin,yin,eta,tol_parada,maxepocas){

  # pegando as dimensoes dos dados de entrada
  N<-dim(xin)[1]
  n<-dim(xin)[2]

  # adicionando coluna de 1s ao vetor de entrada
  xin<-cbind(xin,-1)
  # inicializando o vetor de pesos aleatoriamente
  w<-as.matrix(runif(n+1)-0.5)
  erroepoca<-tol_parada+10
  epocaatual<-0
  vetorEpocas <- matrix(nrow = 1,ncol = maxepocas)

  while((erroepoca>tol_parada)&&(epocaatual<maxepocas)) {
    indexvec = sample(N)
    erroQuad<-0
    for (i in 1:N) {
      #add estocasticidade ao treinamento
      index<-indexvec[i]
      #acha yhat
      yhat<-1*((xin[index,]%*%w)>=0)
      #calcula o erro
      erroatual<-yin[index,]-yhat
      w = w + eta*erroatual*xin[index,]
      erroQuad <- erroQuad + erroatual*erroatual
    }
    epocaatual <- epocaatual + 1 # atualiza as epocas
    vetorEpocas[epocaatual]<-erroQuad/N # erro medio de cada época
    erroepoca <- vetorEpocas[epocaatual]
  }
  return(list(w,vetorEpocas[1:epocaatual]))
}
```

```
r <- treinap(xinTreino,yinTreino,0.01,0.01,100)
w <- r[[1]]
evec <- r[[2]]

# testando os pesos


yperceptron <-function(xvec,w){
  u<-xvec %*% w
  y<-1.0*(u>=0)

  return(as.matrix(y))
}

xinTeste <- rbind(xc1teste,xc2teste)
yTeste <- rbind(y1teste,y2teste)

xinTeste <- cbind(xinTeste,-1)
yhat <- yperceptron(xinTeste,w)

# calculando a acuracia de teste
acuraciaTeste <- 1- (t(yTeste-yhat)%*%(yTeste-yhat))/60

# calculando a acuracia de treino
xinTreino <- cbind(xinTreino,-1)
yhatTreino <- yperceptron(xinTreino,w)

acuraciaTreino <- 1- (t(yinTreino-yhatTreino)%*%(yinTreino-yhatTreino))/140

# matrizes de confusao

confmTreino <- confusionMatrix(factor(yinTreino),factor(yhatTreino))

confmTeste <- confusionMatrix(factor(yTeste),factor(yhat))

print(confmTeste)
```

```
# EX3 ----------------------------------------------------------------
rm(list = ls())

# carregando os dados
library('caret')

iris <- data("iris")
data <- iris3
planta1 <- data[1:50,1:4,1]
planta2 <- data[1:50,1:4,2]
planta3 <- data[1:50,1:4,3]

xin <- rbind(planta1,planta2,planta3)

yclass1 <- matrix(0,nrow = 50)
yclass2 <- matrix(1,nrow = dim(xin)[1]- dim(yclass1)[1])

yin <- rbind(yclass1,yclass2)

# selecionando aleatoriamente 70% das amostras de treino

indexTreino <- sample(dim(xin)[1])

Xtrain <- xin[indexTreino[1:(dim(xin)[1]*0.7)],]
Ytrain <- as.matrix(yin[indexTreino[1:(dim(xin)[1]*0.7)],])

Xtest <- xin[indexTreino[((dim(xin)[1]*0.7)+1):dim(xin)[1]],]
Ytest <- as.matrix(yin[indexTreino[((dim(xin)[1]*0.7)+1):dim(xin)[1]],])

# treinando o modelo
yperceptron <-function(xvec,w){
  xvec <- cbind(xvec,1)
  u<-xvec %*% w
  y<-1.0*(u>=0)

  return(as.matrix(y))
}
```

```
Rtrain <- treinap(Xtrain,Ytrain,0.01,0.001,100)
Wtrain <- Rtrain[[1]]
EvecTrain <- Rtrain[[2]]

YhatTrain <- yperceptron(Xtrain,Wtrain)

accTrain <- 1- (t(Ytrain-YhatTrain)%*%(Ytrain-YhatTrain))/105

# testando o modelo

YhatTest <- yperceptron(Xtest,Wtrain)

accTest <- 1- (t(Ytest-YhatTest)%*%(Ytest-YhatTest))/45

# matrizes de confusao

confmTrain <- confusionMatrix(factor(Ytrain),factor(YhatTrain))

confmTest <- confusionMatrix(factor(Ytest),factor(YhatTest))

print(confmTest)
print(confmTrain)

# criando loop de treinamento

rep <- 100

wv <- matrix(0,nrow = 5,ncol = 1)

for (i in 1:100) {

  r <- treinap(Xtrain,Ytrain,0.1,0.01,100)
  wv <- wv + r[[1]]
}

wv <- wv/rep

YhatTest2 <- yperceptron(Xtest,wv)
```

```
accTest2 <- 1- (t(Ytest-YhatTest2)%*%(Ytest-YhatTest2))/45

confmTest2 <- confusionMatrix(factor(Ytest),factor(YhatTest2))

print(confmTest2)


# EX4 ----------------------------------------------------------------------
rm(list = ls())
# carregando os dados
library('caret')
library(mlbench)

data(BreastCancer)

bcdata <- as.matrix(BreastCancer)

# tratando os dados com NA

bcdata[is.na(bcdata)] <- 0
bcdata <- bcdata[,2:11]

# rotulando as classes

classes <- bcdata[,10]
originalClasses <- bcdata[,10]

for (i in 1:length(classes)) {
  if (classes[i]=="benign") {
    classes[i]<-as.numeric(0)
  }
  else{
    classes[i] <-as.numeric(1)
  }
}

bcdata <- as.matrix(bcdata[,1:9])
```

```
class(bcdata) <- "numeric"
classes <- as.matrix(as.numeric(classes))

# selecionando aleatoriamente 70% das amostras de treino

xin <- bcdata
yin <- classes

indexTreino <- sample(dim(xin)[1])

Xtrain <- xin[indexTreino[1:(dim(xin)[1]*0.7)],]
Ytrain <- as.matrix(yin[indexTreino[1:(dim(xin)[1]*0.7)],])

Xtest <- xin[indexTreino[((dim(xin)[1]*0.7)+1):dim(xin)[1]],]
Ytest <- as.matrix(yin[indexTreino[((dim(xin)[1]*0.7)+1):dim(xin)[1]],])

# treinando o modelo
yperceptron <-function(xvec,w){
  xvec <- cbind(xvec,1)
  u<-xvec %*% w
  y<-1.0*(u>=0)

  return(as.matrix(y))
}

treinap <- function(xin,yin,eta,tol_parada,maxepocas){

  # pegando as dimensoes dos dados de entrada
  N<-dim(xin)[1]
  n<-dim(xin)[2]

  # adicionando coluna de 1s ao vetor de entrada
  xin<-cbind(xin,-1)
  # inicializando o vetor de pesos aleatoriamente
  w<-as.matrix(runif(n+1)-0.5)
  erroepoca<-tol_parada+10
  epocaatual<-0
  vetorEpocas <- matrix(nrow = 1,ncol = maxepocas)
```

```
  while((erroepoca>tol_parada)&&(epocaatual<maxepocas)) {
    indexvec = sample(N)
    erroQuad<-0
    for (i in 1:N) {
      #add estocasticidade ao treinamento
      index<-indexvec[i]
      #acha yhat
      yhat<-1*((xin[index,]%*%w)>=0)
      #calcula o erro
      erroatual<-yin[index,]-yhat
      w = w + eta*erroatual*xin[index,]
      erroQuad <- erroQuad + erroatual*erroatual
    }
    epocaatual <- epocaatual + 1 # atualiza as epocas
    vetorEpocas[epocaatual]<-erroQuad/N # erro medio de cada época
    erroepoca <- vetorEpocas[epocaatual]
  }
  return(list(w,vetorEpocas[1:epocaatual]))
}


#Rtrain <- treinap(Xtrain,Ytrain,0.1,0.01,1000)
#Wtrain <- Rtrain[[1]]
#EvecTrain <- Rtrain[[2]]

#YhatTrain <- yperceptron(Xtrain,Wtrain)

#accTrain <- 1- (t(Ytrain-YhatTrain)%*%(Ytrain-YhatTrain))/489

# testando o modelo

#YhatTest <- yperceptron(Xtest,Wtrain)

#accTest <- 1- (t(Ytest-YhatTest)%*%(Ytest-YhatTest))/dim(Xtest)[1]

# matrizes de confusao
```

```
#confmTrain <- confusionMatrix(factor(Ytrain),factor(YhatTrain))

#confmTest <- confusionMatrix(factor(Ytest),factor(YhatTest))

#print(confmTest)
#print(confmTrain)

# criando loop de treinamento

rep <- 100

wv <- matrix(0,nrow = 10,ncol = 1)

for (i in 1:rep) {

  print(i)
  r <- treinap(Xtrain,Ytrain,1,0.001,250)
  wv <- wv + r[[1]]

}

wv <- wv/rep

YhatTest2 <- yperceptron(Xtest,wv)

accTest2 <- 1- (t(Ytest-YhatTest2)%*%(Ytest-YhatTest2))/209

#confmTest2 <- confusionMatrix(factor(Ytest),factor(YhatTest2))

#print(confmTest2)
```