Universidade Federal de Minas Gerais

Nome: Guilherme Vinícius Amorim

Matrícula: 2017089081

Data: 09/2020

Exercício 6

O objetivo dos exercícios desta semana é utilizar as ELMs para resolver problemas multidimensionais, a partir de bases de dados reais. Os resultados obtidos pelas ELMs foram comparados com os resultados de um algoritmo Perceptron.

As bases de dados são *Breast Cancer (diagnostic)* e *Statlog (Heart)*. Segue abaixo as rotinas em R usadas para carregar os dados. Vale ressaltar que a saída teve que ser ajustada para -1 e 1. Além disso, dados incompletos (NA) foram removidos, permitindo assim uma manipulação matemática futura pelos algoritmos.

```r
### Data

data("BreastCancer")
BreastCancer$Class<-1*(BreastCancer$Class=='benign')
for(i in 1:length(BreastCancer$Class)){
  if(BreastCancer[[11]][i] == 0){
    BreastCancer[[11]][i] <- -1
  }
}
# Removing NA data
BreastCancer<-BreastCancer[complete.cases(BreastCancer),]
dt<-data.matrix(BreastCancer, rownames.force = NA)

X<-as.matrix(dt[1:dim(dt)[1], 2:10])
Y<-as.matrix(dt[1:dim(dt)[1], 11])
```

```r
### Data

data(heart)

# Removing NA data
heart<-heart[complete.cases(heart),]
dt<-data.matrix(heart, rownames.force = NA)

X<-as.matrix(dt[1:dim(dt)[1], 1:12])
Y<-as.matrix(dt[1:dim(dt)[1], 13])

for(i in 1:length(Y)){
  if(Y[i] == 2){
    Y[i] <- -1
  }
}
```

Figura 1: Carregamento das bases de dados.

Após o carregamento das bases de dados, o próximo passo é utilizar o algoritmo da ELM para que um modelo de classificação seja encontrado. O fluxo utilizado no algoritmo da ELM foi:

1. Separação de 30% dos dados para testes e 70% dos dados para treinamento;
2. Cálculo das matrizes Z e H (a partir dos dados de treinamento) e, por conseguinte, o vetor de parâmetros W;
3. Cálculo do YHat para os dados separados para teste e para os dados separados para treinamento;
4. Cálculo de erro para os dados separados para teste e para os dados separados para treinamento;

```r
## Extreme Learning Machine Parameters
extremeLearningMachineParameters <- function(X, Y, p){

  # X: Matrix with learning data
  # Y: Vector with classes (-1 or 1)
  # p: Number of neurons in the hidden layer
  # returns:
  # [[1]] -> Z
  # [[2]] -> W
  # [[3]] -> H

  n_dim<-dim(X)[2]
  Z <- replicate(p, runif(n_dim+1, -0.5, 0.5))
  H <- tanh(cbind(1, X) %*% Z)
  W <- pseudoinverse(H) %*% Y
  l <- list(Z,W,H)
  return(l)
}
```

Figura 2: Rotina para calcular parâmetros da ELM.

```
## ELM
n_max_neurons<-100
n_interaction_per_neurons<-20

error_vec_per_neurons_train<-matrix(nrow=2,ncol=n_max_neurons)
error_vec_per_neurons_test<-matrix(nrow=2,ncol=n_max_neurons)
for(p in 1:n_max_neurons){
  error_vec_train<-matrix(nrow=1,ncol=n_interaction_per_neurons)
  error_vec_test<-matrix(nrow=1,ncol=n_interaction_per_neurons)
  for(i in 1:n_interaction_per_neurons){

    # 30% data for testing and 70% for training
    return<-splitDataTrainTest(X, Y, 0.7)
    XTraining<-return[[1]]
    YTraining<-return[[2]]
    XTest<-return[[3]]
    YTest<-return[[4]]

    # Calc ELM parameters
    return<-extremeLearningMachineParameters(XTraining, YTraining, p)
    Z<-return[[1]]
    W<-return[[2]]
    H<-return[[3]]

    # Calc Yhat of Training data
    return<-extremeLearningMachineOutput(H,W)
    YHatTrain<-return

    # Calc Error of Training data
    acumulated_error<-sum((YTraining-YHatTrain)^2)/4
    error_train<-acumulated_error/nrow(YHatTrain)

    # Calc Yhat of Test data
    H<-tanh(cbind(1, XTest) %*% Z)
    W<-pseudoinverse(H) %*% YTest
    return<-extremeLearningMachineOutput(H,W)
    YHatTest<-return

    # Calc Error of Test data
    acumulated_error<-sum((YTest-YHatTest)^2)/4
    error_test<-acumulated_error/nrow(YHatTest)

    error_vec_train[1,i]<-error_train
    error_vec_test[1,i]<-error_test
  }
  return<-calcMeanVarianceStandDev(error_vec_train)
  error_vec_per_neurons_train[1,p]<-return[[1]]
  error_vec_per_neurons_train[2,p]<-return[[3]]
  return<-calcMeanVarianceStandDev(error_vec_test)
  error_vec_per_neurons_test[1,p]<-return[[1]]
  error_vec_per_neurons_test[2,p]<-return[[3]]
}
```

Figura 3: Fluxo do algoritmo da ELM.

Para estudarmos o comportamento da ELM com o crescimento do número de neurônios p, esse fluxo foi realizado para neurônios p entre 1 e 100. Além disso, como o algoritmo da ELM é susceptível à aleatoriedade da matriz Z, esse fluxo foi realizado

20 vezes para cada número p diferentes de neurônios. Assim, a partir dessas 20 execuções, os valores de média e desvio padrão foram calculados.

O que foi observado nesse estudo é que um número maior de neurônios implica em um erro menor, o que já era esperado. Para esse experimento, foi observado que o erro ficou abaixo de 1% para 100 neurônios na base *Breast Cancer*. Para a base de dados *Heart,* o menor erro foi $\mathrm{Erro}(\mathrm{p} = 92) = (19 +/- 4)\%$. Além disso, com o aumento de p, uma diferença entre o erro na base de testes e o erro da base de treinamento se tornou mais perceptível. Isso demonstra que, embora o erro para os dados de treinamento esteja diminuindo com o aumento de p, provavelmente um *overfit* está ocorrendo, fazendo com que o modelo se adeque bem à base de treinamento, mas não ao sistema em si como um todo.

Segue abaixo os gráficos que relacionam o erro da base de treinamento e da base de teste para um número variável de neurônios (1:100):

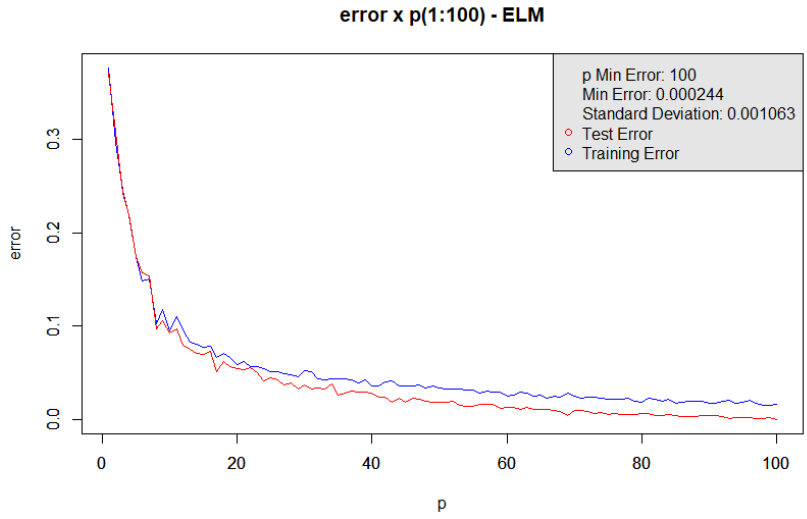

Figura 4: Erro x n° de neurônios no algoritmo da ELM para a base *Breast Cancer*.

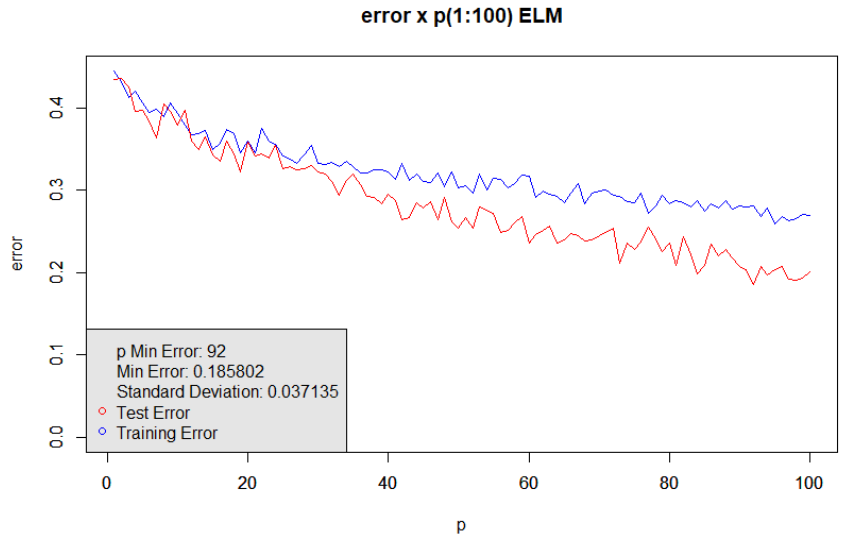

Figura 5: Erro x n° de neurônios no algoritmo da ELM para a base *Heater*.

A fim de comparação, o algoritmo do Perceptron foi utilizado nas duas bases de dados. Para tanto, os valores de saída tiveram que ser novamente ajustados, dessa vez para 0 e 1. Os parâmetros utilizados no algoritmo do Percepron foram:

$$\eta = 0.01$$

$$tolerância = 0.0001$$

$$Número\max de\ épocas = 10000$$

```
## Perceptron

for(i in 1:length(Y)){
  if(Y[i] == -1){
    Y[i] <- 0
  }
}

# 30% data for testing and 70% for training
return<-splitDataTrainTest(X, Y, 0.7)
XTraining<-return[[1]]
YTraining<-return[[2]]
XTest<-return[[3]]
YTest<-return[[4]]

# Calc Perceptron
return<-trainperceptron(XTraining,YTraining,0.01,0.0001,10000,1)
Wp<-return[[1]]
error_per_epoch_p<-return[[2]]

# Calc Yhat of Training data
return<-yperceptron(XTraining,Wp,1)
YHatTrain<-return

# Calc Error of Training data
acumulated_error<-sum(abs(YTraining-YHatTrain))
error_train<-acumulated_error/nrow(YHatTrain)

# Calc Yhat of Test data
return<-yperceptron(XTest,Wp,1)
YHatTest<-return

# Calc Error of Test data
acumulated_error<-sum(abs(YTest-YHatTest))
error_test<-acumulated_error/nrow(YHatTest)
```

Figura 6: Algoritmo Perceptron.

Segue abaixo gráficos que relacionam o erro do Perceptron pelo número de épocas para as bases de *Breast Cancer* e *Heater:*

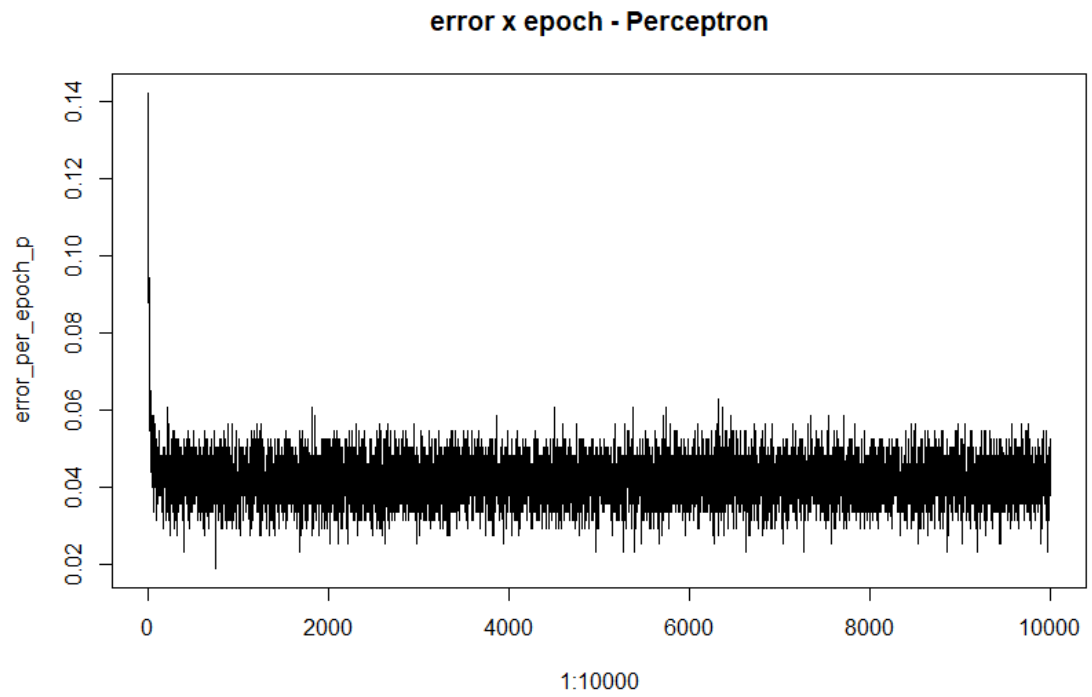

Figura 7: Erro x Época no Perceptron para a base de *Breast Cancer*.

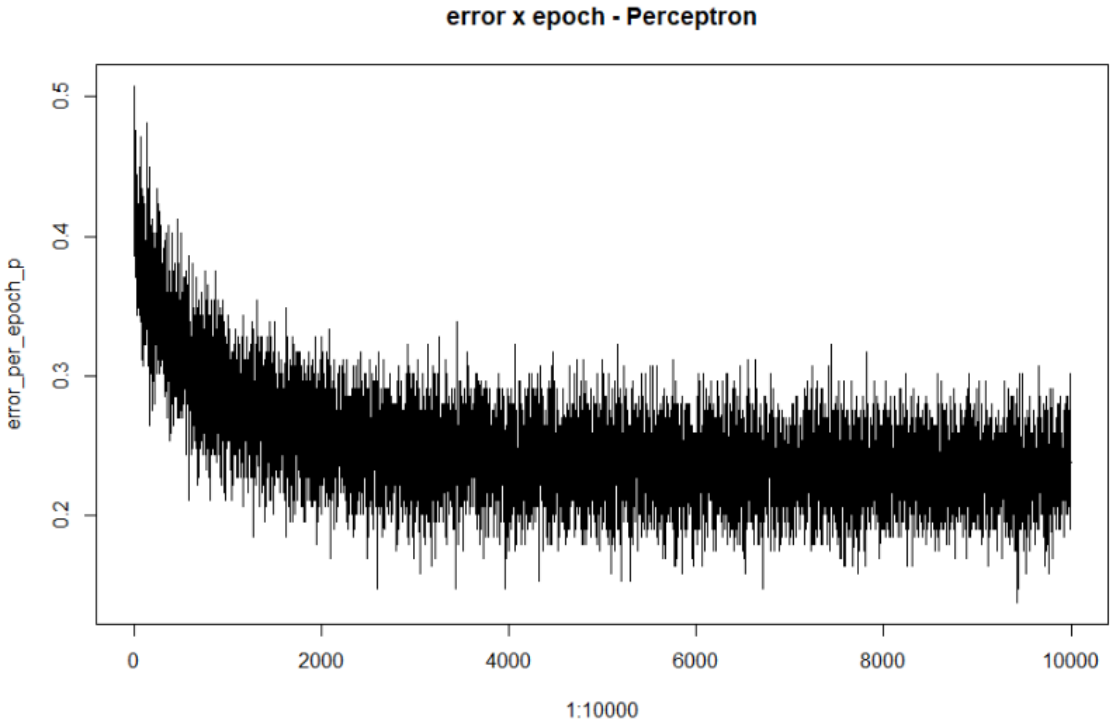

Figura 8: Erro x Época no Perceptron para a base de *Heater*.

Os valores de erro obtidos pelo algoritmo de Perceptron para as bases *Breast Cancer* e *Heater* estão nos *bar plots* a seguir:

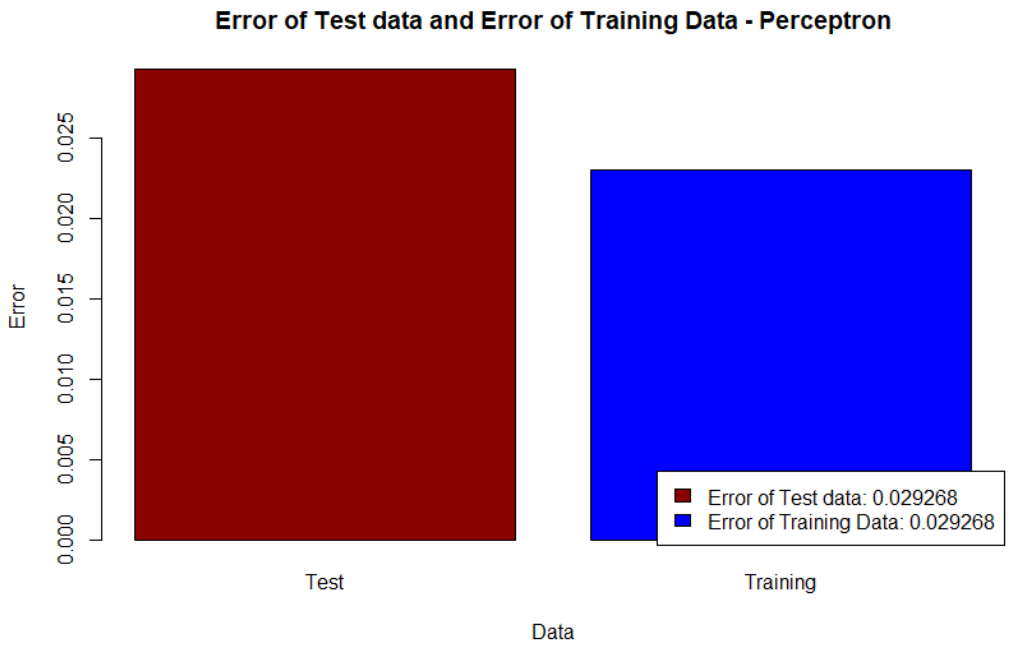

Figura 9: Erro no Perceptron para os dados de Teste e de Treinamento para a base *Breat Cancer.*

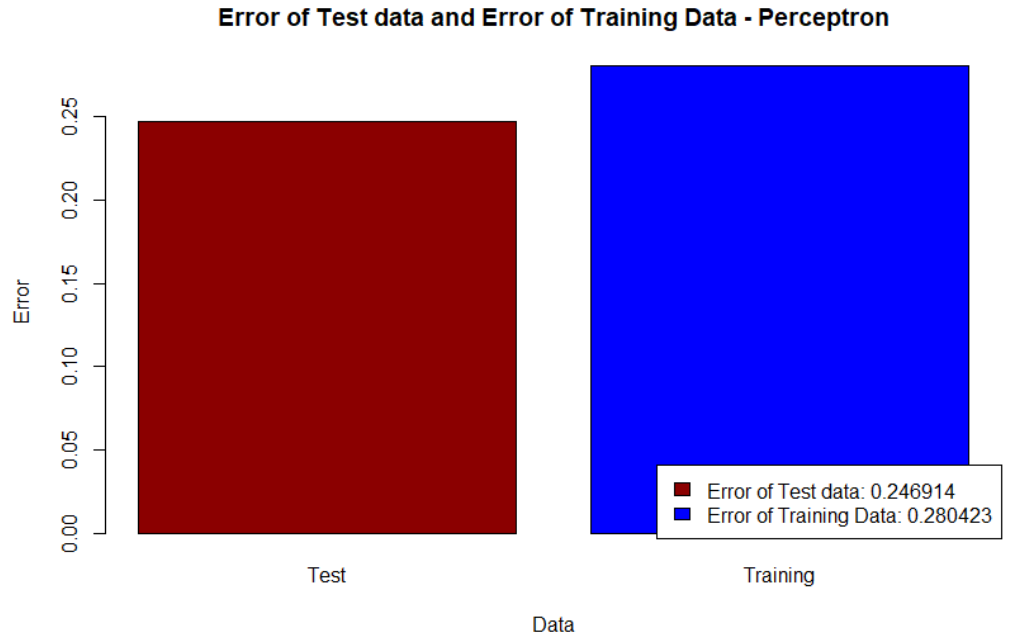

Figura 10: Erro no Perceptron para os dados de Teste e de Treinamento para a base *Heater*

Comparando os resultados das ELMs com os resultados do algoritmo de Perceptron, temos:

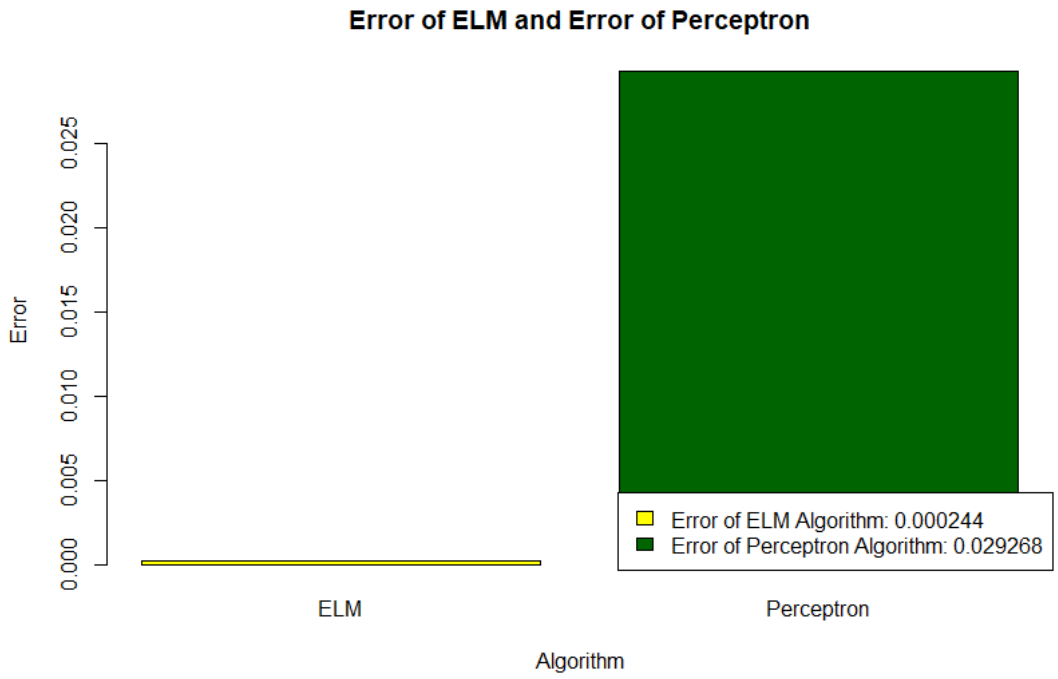

Figura 11: Erro dos dados de teste para os algoritmos ELM e Perceptron (base *Breast Cancer*).

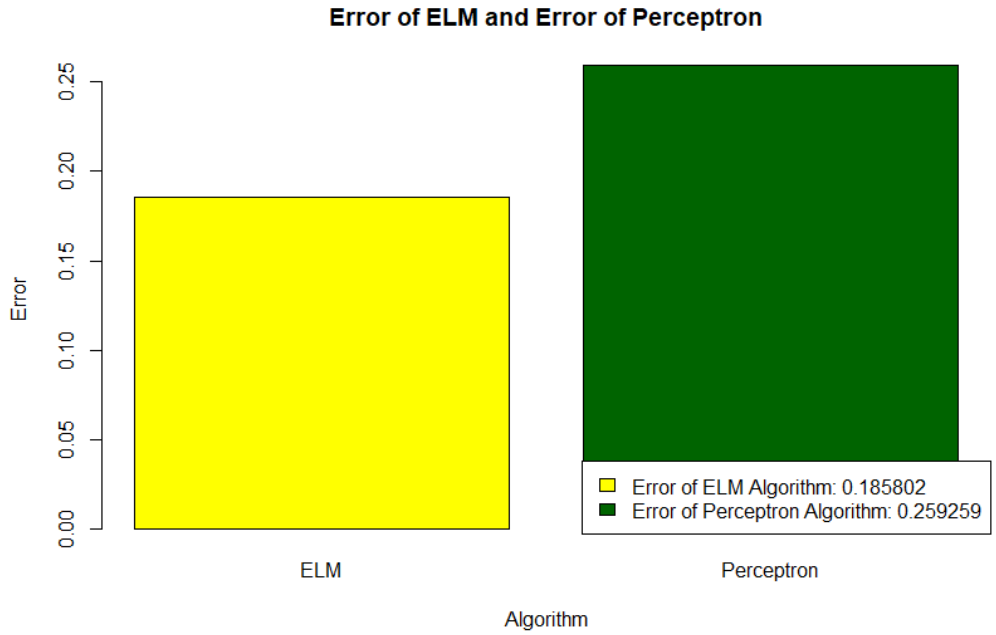

Figura 12: Erro dos dados de teste para os algoritmos ELM e Perceptron (base *Heater)*.

Analisando os resultados obtidos para as ELMs e Perceptron, observamos que os erros para a base *Breast Cancer* foram significavelmente menores. Isso mostra que a base *Breast Cancer* tem grande chance de ser um problema linearmente separável.

Contudo, os erros para a base *Heater* foram de 19% e 26% para a ELM e para o Perceptron, respectivamente.

Conclusão

No estudo dessa semana, foi detectado que para um número maior de neurônio no treinamento das ELMs, um erro percentual menor será gerado. Contudo, ao analisar o erro dos dados gerados para teste e para treinamento observamos um possível *overfit*, uma vez que o erro para o conjunto de teste passou a ser consideravelmente maior que o erro da base de treinamento para um número elevado de neurônios. Além disso, ao compararmos os resultados obtidos pelas ELMs com os resultados obtidos pelo Percepron, percebe-se que as ELMs possuem uma tendência de terem um erro menor. Contudo, vale ressaltar que ambos os algoritmos estão susceptíveis à aleatoriedade. Essa tendência de erro menor nas ELMs foi obtida a partir da média de erro de 20 interações. Vale também ressaltar que para um melhor resultado nos dois modelos de classificação seria possível um escalonamento dos dados, evitando possíveis mínimos locais e possivelmente aumentando a porcentagem de acerto.