Universidade Federal de Minas Gerais
Redes Neurais Artificiais

Perceptron e Adaline

Victor Marcius Magalhães Pinto
Mat: 2019717730

1 Introdução

Os exercícios propostos tem por objetivo exercitar uma maior conhecimento à respeito das implementações mais simples de redes neurais, tanto para classificação quanto para regressão. O Perceptron consiste em um único neurônio, realizando uma soma ponderada de parâmetros de entrada e aplicando uma função de ativação a fim de classificar classes linearmente separáveis, através de uma reta de limite. O Adaline consiste em um neurônio capaz de realizar regressões lineares e polinomiais.

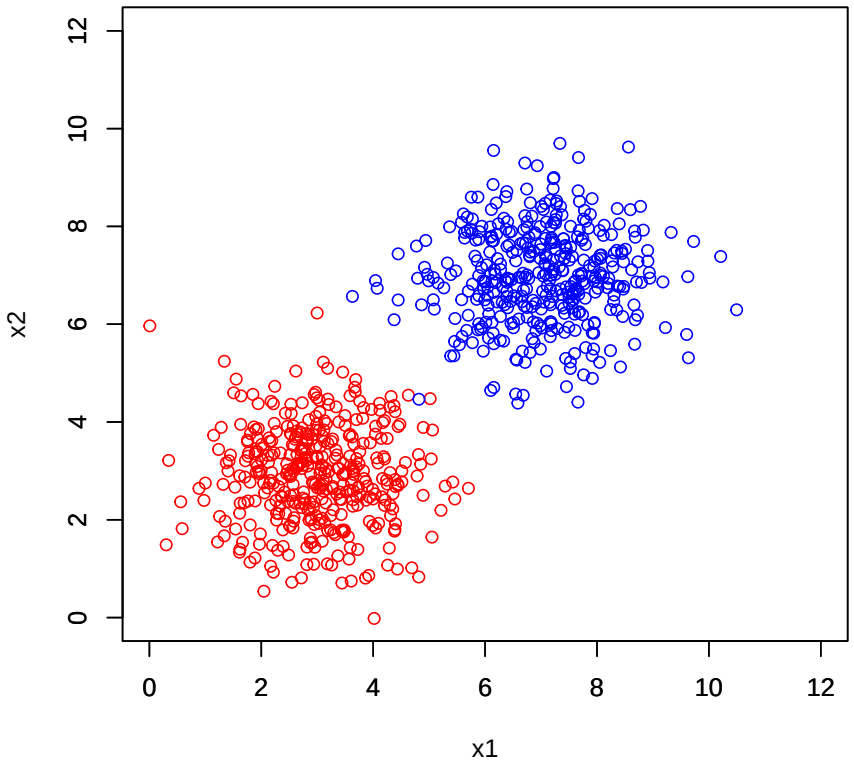
2 Classificação

As primeiras propostas de aplicação consistem em utilizar o perceptron para solucionar problemas de classificação. O mesmo realiza uma soma ponderada dos parâmetros de entrada, juntamente com a adição de um termo de bias, e depois aplica uma função de separação, comumente uma tangente hiperbólica, de forma a realizar uma classificação de um conjunto de dados em duas classes distintas. Graficamente, estabelece uma reta suporte, onde amostras ”acima”desta reta são classificadas como pertencentes a uma determinada classe, e retas abaixo, a outra.

2.1 Classes linearmente separáveis

Primeiramente, definimos dois consjutos de classes linearmente separáveis, como observado a seguir.

```
> set.seed(42)
> class1 = matrix(rnorm(800, 3, 1), ncol = 2)
> class2 = matrix(rnorm(800, 7, 1), ncol = 2)
```



2.2 Função de treinamento

O treinamento de um perceptron consiste no ajuste do valor de seus pesos. Inicialmente os pesos são definidos de uma maneira randômica, e então é avaliado o desempenho da superfície de separação resultante em classificar corretamente um dado conjunto de amostras, com base em seus rótulos reais.

$$\overline{w} = w_1, w_2, ..., w_n \tag{1}$$

$$\hat{y} = \tanh\left(\overline{w} \cdot \overline{x}'\right) \tag{2}$$

$$error = y_{label} - \hat{y} \tag{3}$$

O peso, então, é atualizado conforme o seguinte. Seja t a interação atual do algoritmo,

$$w_{t+1} = w_t + \eta \cdot error \cdot \overline{x} \tag{4}$$

sendo η uma constante de aprendizado, de correção dos pesos. O algoritmo, portanto, tem a forma,

```
> train <- function(x, y, eta=0.01) {
+     epochs <- 100
+     correction_factor <- eta
+
+     x_aug <- cbind(x, replicate(dim(x)[1], 1))
+
+     w = rnorm(dim(x)[2] +1, 0, 1)
+
+     errors = c()
+
+     for (iter in seq(epochs)) {
+
+         curr_error <- c()
+         for (index in sample(dim(x)[1])){
+             y_partial <- 1*((x_aug[index,] %*% w) >= 0)
+             error <- y[index] - y_partial
+             delt_w <- correction_factor*error*x_aug[index,]
+             w <- w + delt_w
+
+             curr_error <- c(curr_error, error)
+         }
+         errors <- c(errors, mean(curr_error))
+     }
+
+     result <- list('errors'=errors, 'weights'=w)
+
+     return(result)
+ }
```

Os pessos são revaliados várias vezes, a partir de um número de épocas determinado, e a cada iteração espera-se que o erro associado à classificação diminua, e o valor obtido pelo modelo se aproxime do valor real das observações.

A função de avaliação dos resultados, pode ser implementada como:

```
> evaluate <- function(x, y, w) {
+     y_result <- c()
+
+     errors <- c()
+     false_positive <- 0
+     true_positive <- 0
+     false_negative <- 0
+     true_negative <- 0
+
+     x_aug <- cbind(x, replicate(dim(x)[1], 1))
+
+     for (index in seq(dim(x)[1])) {
+         y_partial <- 1*((x_aug[index,] %*% w) >= 0)
```

```
+         error <- y[index] - y_partial
+
+         if (error > 0) {
+             false_negative <- false_negative + 1
+         } else if (error < 0) {
+             false_positive <- false_positive + 1
+         } else if (y[index] == 1) {
+             true_positive <- true_positive + 1
+         } else {
+             true_negative <- true_negative + 1
+         }
+
+         errors <- c(errors, error)
+         y_result <- c(y_result, y_partial)
+
+     }
+
+     confusion_matrix <- matrix(replicate(4, 0), nrow = 2, ncol = 2)
+     confusion_matrix[1,1] <- true_positive
+     confusion_matrix[1,2] <- false_positive
+     confusion_matrix[2,1] <- false_negative
+     confusion_matrix[2,2] <- true_negative
+
+     return(list(
+         'y_result' = y_result,
+         'errors' = errors,
+         'mean_error'= mean(errors),
+         'accuracy' = (1 - mean(errors)),
+         'specitivity'= true_negative / (true_negative + false_positive),
+         'sensibility' = true_positive / (true_positive + false_negative),
+         'confusion_matrix' = confusion_matrix
+     ))
+ }
```

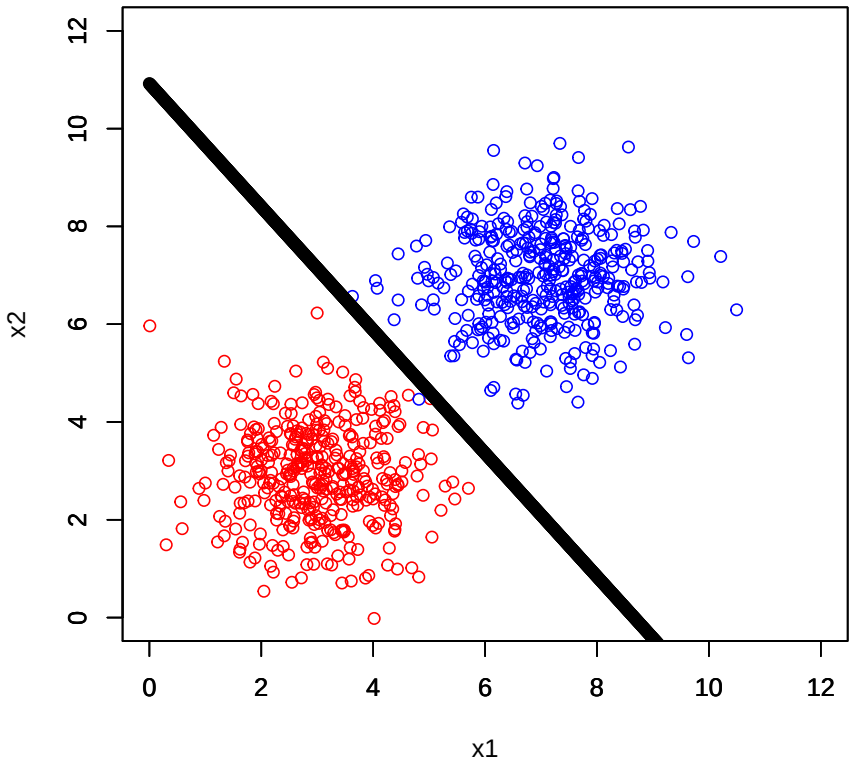
Após realizar o treinamento com os conjuntos de dados obtidos anteriormente, foram obtidos os seguintes parâmetros de peso,

```
[1]  0.1355933  0.1708632 -1.4806843
```

E o erro de aprendizado, sendo avaliado na porcentagem de erros, obtido, foi de,

```
[1] 0.004053571
```

A reta de separação das classes, portanto, pode ser vista a seguir, graficamente. É possível observar como as amostras obtiveram uma separação satisfatória. Qualquer reta que separe as classes corretamente pode ser consisderada como solução do problema, porém é possível observar como a reta obtida possui uma boa adequação ao espaço considerado.



2.3 Winston Breast Cancer

O perceptron implementado nas atividades anteriores será utilizado, portanto, para a identificação de tumores malignos, através dos dados disponíveis no dataset Wisconsin Breast Cancer, discponível no pacote mlbench do R.

O conjunto de dados consiste em 699 observações, sobre pacientes, caracterizadas por 11 parâmetros, sendo eles 1 parâmetro de ID, 9 de características de um observador e um parâmetro indicando o label do dado, isto é, se o tipo de câncer da amostra é maligno ou benigno.

Primeiramente, portanto, os dados são treinados através da função de trinamento implementada. Para tanto, uma limpeza do dataset é necessária. Alguns campos contém valores nulos (NA), que podem atrapalhar na classificação. Além disso, é necessário garantir que todos os campos sejam numéricos, incluindo os rótulos, uma vez que todo o processo é feito matematicamente. Portanto, a preparação dos dados, incluindo a divisão dos datasets em treinamento e teste, são feitas anteriormete como mostrado a seguir:

```
> data("BreastCancer")
> bc <- BreastCancer[complete.cases(BreastCancer),]
```

```
> x <- bc[, 2:10]
> y <- bc[,11]
> x <- sapply(x, as.numeric)
> y <- replicate(dim(bc)[1], 0)
> y[which(bc[,11]== 'benign')] = 0
> y[which(bc[,11]== 'malignant')] = 1
> sprintf('Total number of valid observations: %d', dim(x)[1])

[1] "Total number of valid observations: 683"
```

Os dados então são treinados, e em seguida avaliados, através de

```
> index_train = sample(seq(dim(x)[1]), as.integer(0.7*dim(x)[1]), replace=FALSE)
> x_train = x[index_train,]
> x_test = x[-index_train,]
> y_train <- y[index_train]
> y_test = y[-index_train]
> results_bc <- train(x_train, y_train, eta = 0.5)
> test_results_bc <- evaluate(x_test, y_test, results_bc$weights)
```

Para avaliar o desempenho do modelo em classificação, podemos usar métricas como porcentagem de acertos dos dados de teste, sensibilidade e especificidade. Sensibilidade é definida como sendo a medida da proporção de casos positivos nas amostras que foram corretamente classificadas como positivas, também chamada na literatura de *recall*. Matematicamente, pode se calculado como sendo:

$$Sentibilidade = \frac{TruePositive}{TruePositive + FalsePositive} \tag{5}$$

Especificidade é definida como sendo a proporção de negativos nas amostras que foram preditas como negativas. Matematicamente, pode ser definidocomo sendo:

$$Specificity = \frac{TrueNegative}{TrueNegativa + FalsePositive} \tag{6}$$

A porcentagem de erro do modelo, para o conjunto de teste, é:

```
[1] 0.004878049
```

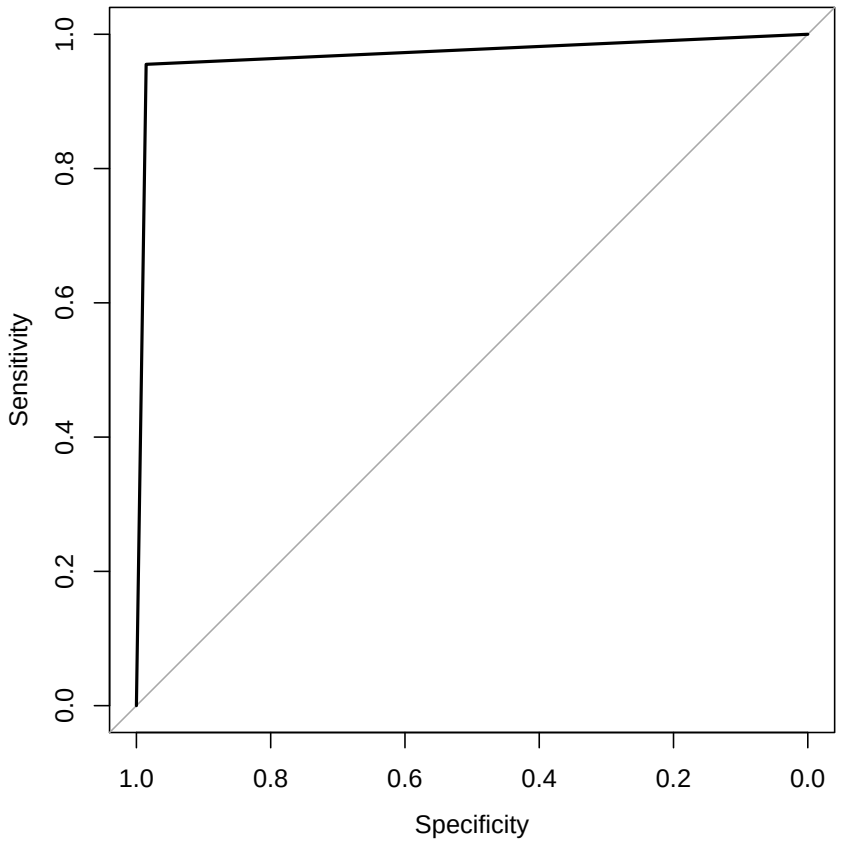
A sensibilidade para os dados foi de:

```
[1] 0.9552239
```

e a especificidade foi de:

```
[1] 0.9855072
```

A curva ROC (Receiver Operating Characteristic) pode ser utilizada para a avaliação do desempenho do modelo. A mesma é construída cruzando sensibilidade com specificidade.



A área de uma curva ROC pode ser utilizada para a avaliação do desempenho de um modelo de classificação. Quanto maios próximo de 1, melhor o desempenho da rede em classificar um conjunto de amostras. A área sob a curva ROC para este modelo, é de

```
Area under the curve: 0.9704
```

A matriz de confusão é uma tabela que relaciona a quantidade de verdadeiros positivos, negativos, e de falsos positivos e negativos. Ela é construída como sendo:

True Positive	False Positive
False Negative	True Negative

Para o conjunto de dados avaliado, a matriz é, portanto,

```
     [,1] [,2]
[1,]   64    2
[2,]    3  136
```

Os resultados demonstram um desempenho considerado alto para o modelo e o conjunto de dados especificados. A acurácia é elevada, com menos de 1% de erro nas predições para o conjunto de teste.

3 Regressão

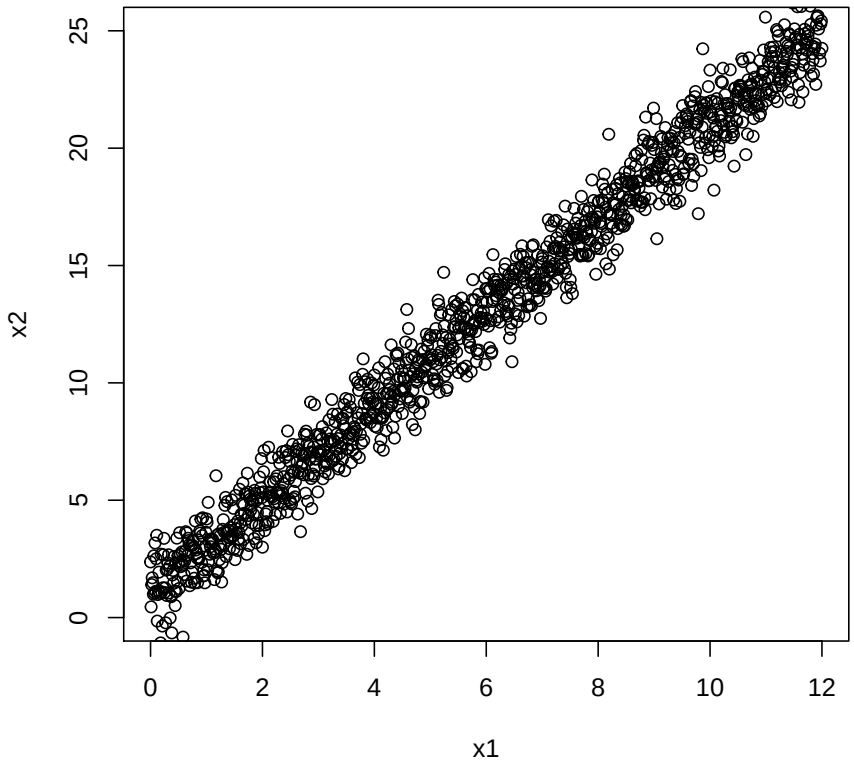
Um neurônio MCP é caracterizado pela aplicação de uma função de limiar à soma ponderada das features consideradas. No caso do perceptron, como especificado anteriormente, esta corresponde a uma função de separação, tal como tangente hiperbólica. O modelo adaline consiste em aplicar, à soma ponderada, uma função de identidade, de forma que a saída corresponda exatamente ao valor da soma ponderada de entrada. Considerando a função de treinamento definida anteriormente, aplicando a modificação na função de ativação, a mesma se torna:

```
> train <- function(x, y, eta=0.01, dimension=2) {
+     epochs <- 100
+     correction_factor <- eta
+
+     x_aug <- cbind(x, replicate(dim(x)[1], 1))
+
+     w = rnorm(dim(x)[2]+1, 0, 1)
+
+     errors = c()
+
+     for (iter in seq(epochs)) {
+         x_seq <- sample(dim(x_aug)[1])
+
+         curr_error <- c()
+         for (index in x_seq){
+             y_partial <- 1*(x_aug[index,] %*% w)
+             error <- y[index,1] - y_partial
+             delt_w <- correction_factor*error*x_aug[index,]
+             w <- w + delt_w
+
+             curr_error <- c(curr_error, error**2)
+         }
+         errors <- c(errors, mean(curr_error))
+     }
+     result <- list('errors'=errors, 'weights'=w)
+
+     return(result)
+ }
>
```

3.1 Regressão de uma Reta

Para testar o conceito deste modelo, primeiramente é gerado um conjunto de dados lineares onde cada amostra possui um erro associado, conforme:

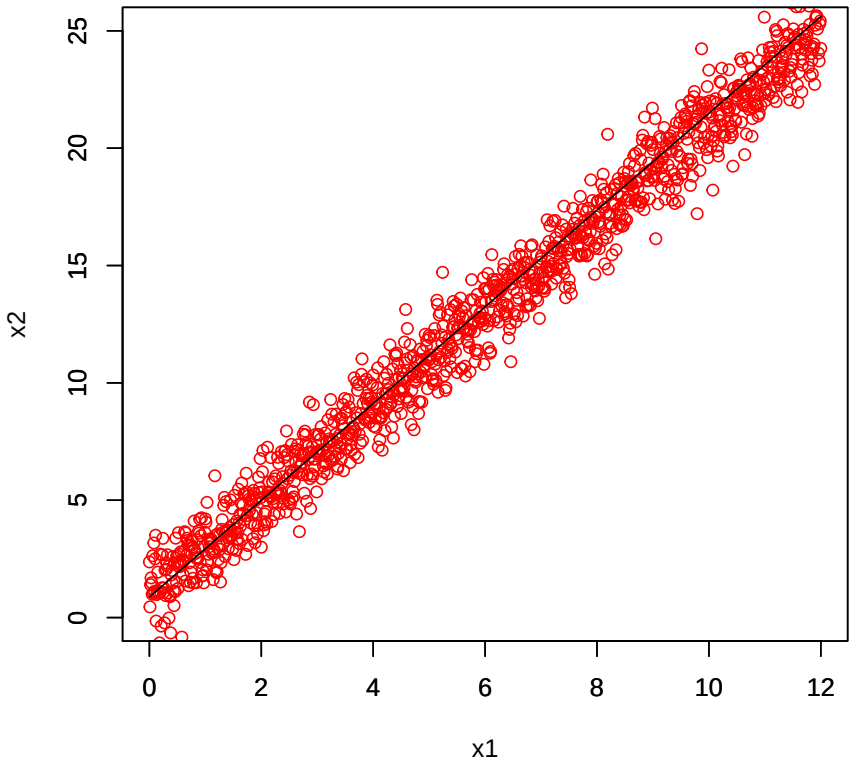
```
> set.seed(42)
> f_generator <- c(2, 1)
> y <- c()
> x <- seq(0, 12, 0.01)
> for (index in x) {
+     data_y <- index * f_generator[1] + f_generator[2]
+     y <- c(y, data_y + rnorm(1, 0, 1))
+ }
> x <- as.matrix(x, ncol=1)
> y <- as.matrix(y, ncol=1)
> index_train = sample(dim(x)[1], replace=FALSE)
> x_train = x[index_train,]
> y_train = y[index_train,]
> xlim <- c(0,12)
> ylim <- c(0, 25)
> plot(x, y, xlab = 'x1', ylab = 'x2', xlim=xlim, ylim=ylim)
```



Aplicando a função de treinamento aos dados associados, temos os seguintes valores para os pesos:

```
[1] 2.0611384 0.8751295
```

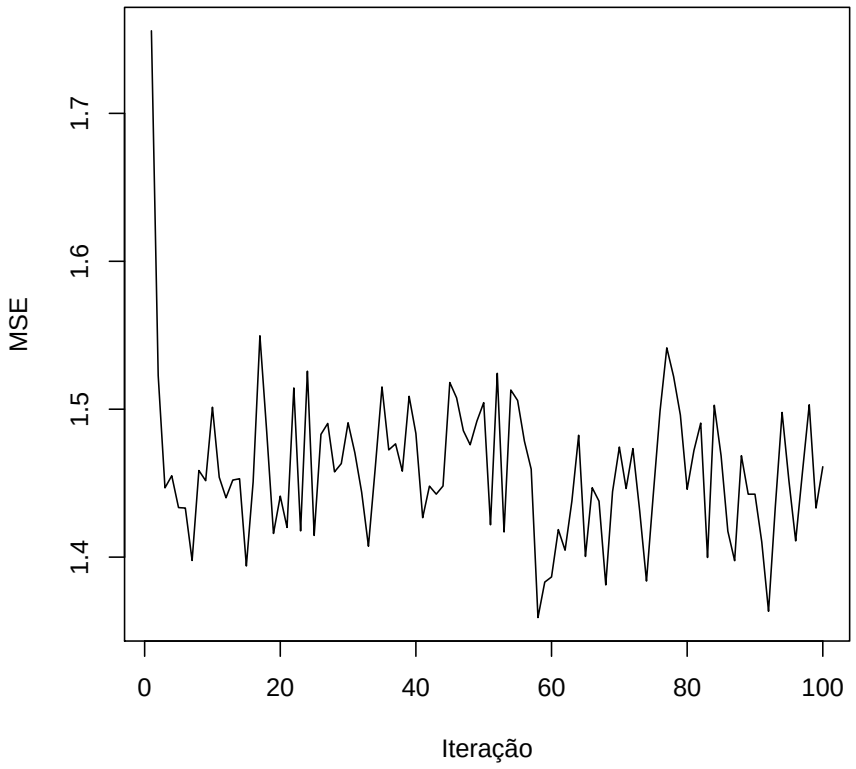
Onde a reta de regressão obtida pode ser vista à seguir:



O erro médio quadrático para o treinamento foi de:

```
[1] 1.458989
```

Se observarmos a evolução dos erros à cada interação, é possível verificar que os mesmos apresentam uma distribuição normal, sem que haja uma queda observável da primeira interação até a última. Uma hipótese para esta observação é o fato de que, sendo uma regressão linear, com poucas amostras é possível estimar uma reta característica, e os próximos pontos apenas se enquadram de realizar o ajuste. Ao realizar o treinamento com todos os pontos, mesmo variando de forma aleatória suas ordem, a cada iteração, é pouco improvável que haja uma variação considerável do ajuste da reta em uma próxima iteração. Em outras palavras, pode-se inferir que a equação que se deseja obter a partr da aplicação do modelo é pouco complexa para que sejam observadas variações consideráveis do erro médio quadrático de cada uma delas.



3.2 Regressão de Datasets Personalizados

Utilizando o Adaline, é possível avaliar seu comportamento no treinamento de três conjuntos de dados específicos.

```
> rm(list = setdiff(ls(), lsf.str()))
> data = read.csv("BUILDING1paraR.DT", sep=" ")
> x <- as.matrix(data[1:14])
> y <- as.matrix(data[15:17])
> y_energy <- as.matrix(data[15])
> y_hot <- as.matrix(data[16])
> y_cold <- as.matrix(data[17])
> index_train = sample(seq(dim(x)[1]), as.integer(0.7*dim(x)[1]), replace=FALSE)
> x_train = as.matrix(x[index_train,])
> x_test = as.matrix(x[-index_train,])
> y_train <- as.matrix(y[index_train,])
> y_test = as.matrix(y[-index_train,])
```

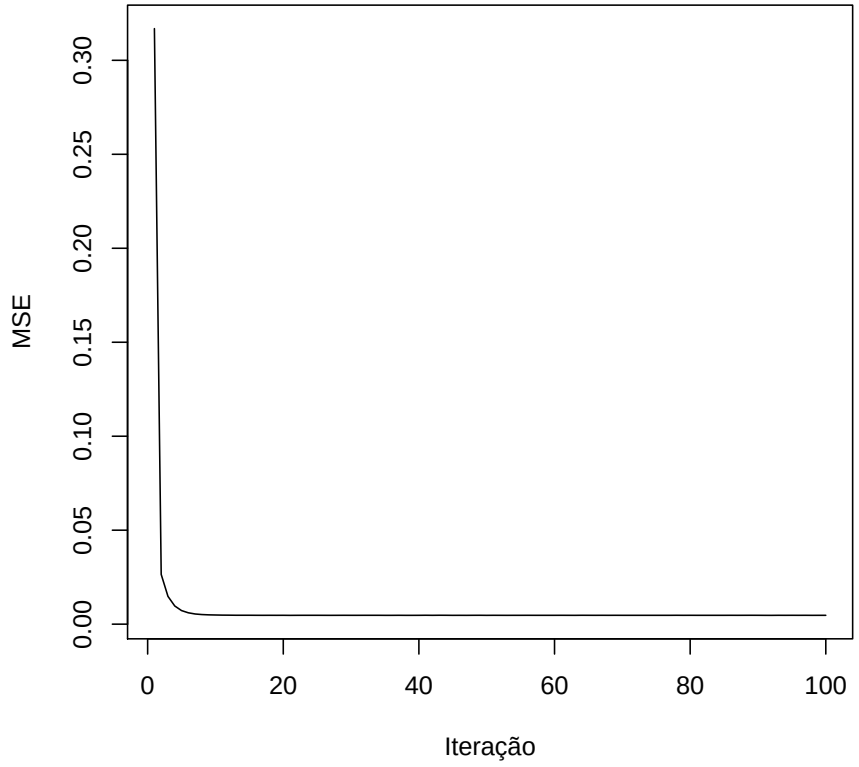
Realizando o treinamento para o parâmetro 'Energia',

```
> results_energy <- train(x_train, as.matrix(y_train[,1]))
```

o erro médio quadrático obtido para o treinamento é:

```
[1] 0.008258605
```

cuja evolução pode ser vista na figura à seguir.



Para o parâmetro "Hot Water',

```
> results_hot <- train(x_train, as.matrix(y_train[,2]))
```

o erro médio quadrático obtido para o treinamento é:

```
[1] 0.003107431
```

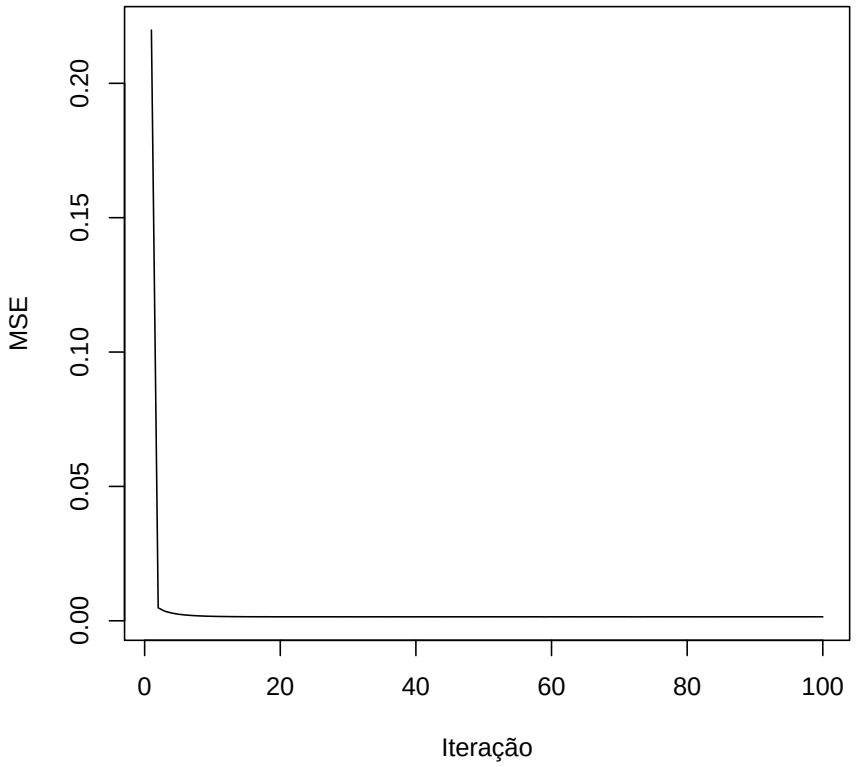
E, finalmente, para o parâmetro de "Cold Water".

```
> results_cold <- train(x_train, as.matrix(y_train[,2]))
```

o erro médio quadrático obtido para o treinamento é:

```
[1] 0.003758176
```

cuja evolução pode ser vista na figura à seguir.



Fazendo uso de funções de fit e avaliação, conforme definidas a seguir,

```
> fit <- function(x, w) {
+     x_aug <- cbind(x, replicate(dim(x)[1], 1))
+
+     rows <- dim(x_aug)[1]
+     cols <- dim(x_aug)[2]
+
+     y <- matrix(ncol = 1, nrow = rows)
+
+     for (index in seq(rows)) {
+         y[index,] <- x_aug[index, ] %*% w
+     }
```

```
+
+     return (y)
+
+ }
> evaluate <- function(y, y_hat) {
+     rows <- dim(y)
+
+     mse <- 0
+     err_percent <- 0
+     for (index in seq(rows)) {
+         error <- y[index,] - y_hat[index,]
+         err_percent <- abs(error / y[index,])
+         mse <- mse + error**2
+     }
+
+     return (list(
+         'mse'=mean(mse),
+         'err_percent'=mean(err_percent)
+     ))
+ }
```

é possível avaliar o desempenho dos modelos para conjuntos de teste dos valores especificados. Então, seguindo um algoritmo como o descrito a seguir,

```
> y_energy <- fit(x_test, results_energy$weights)
> y_hot <- fit(x_test, results_hot$weights)
> y_cold <- fit(x_test, results_cold$weights)
> mse_energy <- evaluate(as.matrix(y_test[, 1]), y_energy)
> mse_hot <- evaluate(as.matrix(y_test[, 2]), y_hot)
> mse_cold <- evaluate(as.matrix(y_test[, 3]), y_cold)
```

Obtemos os valores de erro quadrático médio para cara parâmetro.

```
[1] "MSE para \"Energy\": 0.017078"

[1] "MSE para \"Hot Water\": 0.001925"

[1] "MSE para \"Cold Water\": 0.402554"

[1] "Error percentual para \"Energy\": 0.277896"

[1] "Error percentual para \"Hot Water\": 0.053144"

[1] "Error percentual para \"Cold Water\": 3.088088"
```

Os resultados dos testes para os modelos, nos três parâmetros, apontam para um bom desempenho dos mesmos, se considerarmos um parâmetro de limite abaixo de 5% de erro. O maior erro percentual foi para ”Cold Water”, com 3% de erro. Um neurônio com uma função de ativação de identidade, portanto, apresenta-se como uma boa solução de modelagem do problema apresentado pelo conjunto de dados.

Referências

[1] ML Metrics: Sensitivity vs. Specificity - `https://dzone.com/articles/ml-metrics-sensitivity-vs-specificity-difference`. Acessado em 28 de agosto de 2019.