Universidade Federal de Minas Gerais
Aluno: Giovanni Martins de Sá Júnior
Matrícula: 2017001850

Exercício 10: Redes Neurais Artificiais

Neste décimo exercício realizaremos a aplicação de uma rede MLP para a resolução de problemas multidimensionais por meio de dados reais. Com isso, foi utilizado o pacote RSNSS disponibilizado no R para a implementação deste exercício. Com isso, as bases de dados utilizadas são a Boston Housing, e a Statlog (Heart).

Contudo, foi necessário realizar um escalonamento dos valores presentes nos datasets, para que ficassem restritos entre 0 e 1, e não atrapalhasse a análise dos resultados obtidos. Esse escalonamento é feito pelo seguinte cálculo:

$$z_i = \frac{x_i - min(x)}{max(x) - min(x)}$$

Assim sendo, foi realizada a análise classificando o Erro de classificação do modelo, seguido do seu desvio padrão, variando-se a sua respectiva arquitetura.

1. Boston Housing

No conjunto de dados do Boston Housing, foi realizado um escalonamento inicial das variáveis do dataset, seguidos da separação dos dados de treinamento e teste tanto da entrada, quanto da saída. Para este exercício, foi definido que 70% dos dados escolhidos de forma aleatória do dataset seriam destinados para treinamento, e os outros 30% restantes para o teste do modelo. A seguir é apresentada a implementação feita do problema para este dataset:

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

normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}

# Carregando a base de dados
housing_data <- read.table("C:/Projetos/Trabalhos-Faculdade/Redes Neurais Artificiais/Exercicios/10/housing.data", quote="\"", comment.char="")
colnames(housing_data) <- c("CRIM", "ZN", "INDUS", "CHAS", "NOX", "RM", "AGE", "DIS", "RAD", "TAX", "PTRATIO", "B", "LSTAT", "MEDV")

# Separando os dados de entrada e saida do dataset
X <- housing_data[,1:13]
Y <- housing_data[, 14]

# Divisao dos dados em treinamento e teste
set.seed(123)
training_index <- sample(1:nrow(housing_data), 0.7 * nrow(housing_data))
test_index <- setdiff(1:nrow(housing_data), training_index)

X_training <- X[training_index,]
X_test <- X[test_index,]
Y_training <- Y[training_index]
Y_test <- Y[test_index]

# Normalizacao dos dados de entrada
X_training <- apply(X_training, 2, normalize)
X_test <- apply(X_test, 2, normalize)
```

Figura 1: Normalização e Separação dos dados de entrada e saída para treinamento e teste

```r
# Montagem da arquitetura
number_neurons <- c(10, 30, 50) # Numero de neuronios em cada camada oculta

for (n in number_neurons) {
  model <-  mlp (X_training, Y_training, size=c(n), maxit=20000, initFunc="Randomize_Weights", initFuncParams=c(-5, 5),
                 learnFunc="Std_Backpropagation", learnFuncParams=c(0.001, 0.5),
                 updateFunc="Topological_Order", updateFuncParams=c(0),
                 hiddenActFunc="Act_Logistic",
                 shufflePatterns=TRUE, linOut=TRUE)

  # Previsao do conjunto de teste
  Y_prediction <- predict(model, X_test)

  # Calculo do erro medio e do desvio padrao
  Error <- Y_prediction - Y_test
  Mean_Error <- mean(Error)
  SD_Error <- sd(Error)

  # Resultados
  cat("Numero de Neuronios: ", n, "\n")
  cat("Erro Medio: ", Mean_Error, "\n")
  cat("Desvio Padrao: ", SD_Error, "\n\n")
}
```

Figura 2: Aplicação da MLP para diferentes arquiteturas

Após a separação dos dados presentes na Figura 1, foi realizada a aplicação da MLP para diferentes arquiteturas, em que cada uma apresenta um determinado número de neurônios. Para este exercício, foi escolhido três valores distintos de neurônios para a camada intermediária, sendo eles de 10, 30 e 50 neurônios, com o objetivo de medir a precisão de classificação do parâmetro MEDV (coluna 14 do dataset). Abaixo é apresentado o resultado obtido para o erro médio de classificação, com o respectivo desvio padrão para cada arquitetura:

```
Numero de Neuronios:  10
Erro Medio:  -5.567201
Desvio Padrao:  5.208289

Numero de Neuronios:  30
Erro Medio:  -4.079485
Desvio Padrao:  4.938578

Numero de Neuronios:  50
Erro Medio:  -5.677581
Desvio Padrao:  5.695017
```

Figura 3: Erro Médio e Desvio Padrão obtido nas diferentes arquiteturas

Como pode ser visto, foi observado que o menor Erro Médio encontrado dentre as três arquiteturas utilizadas, foi aquela em que foi utilizada 30 neurônios, o que indica que ela possa ter uma maior capacidade de aprendizado e ser mais capaz de se ajustar aos dados de treinamento. Além disso, ela também apresentou um menor desvio padrão, sugerindo uma maior estabilidade para a classificação de dados desconhecidos.

Contudo, ao se comparar com a arquitetura seguinte de 50 neurônios, em que ocorre um aumento do erro e do desvio padrão, é observado um indício de início de overfitting devido ao aumento erro, e uma maior sensibilidade na variação dos dados a serem classificados, resultando em uma maior variabilidade de classificações.

2. Heart (Data)

No segundo dataset, foi realizado um processo similar de escalonamento inicial das variáveis do dataset, seguidos da separação dos dados de treinamento e teste tanto da entrada, quanto da saída. Para este segundo exemplo, também foi definido que 70% dos dados escolhidos de forma aleatória do dataset seriam destinados para treinamento, e os outros 30% restantes para o teste do modelo. A seguir é apresentada a implementação feita neste segundo problema:

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

normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}

# Carregando a base de dados
heart_data <- read.table("C:/Projetos/Trabalhos-Faculdade/Redes Neurais Artificiais/Exercicios/10/heart.dat", quote="\"", comment.char="")
colnames(heart_data) <- c("age", "sex", "cp", "trestbps", "chol", "fbs", "restecg", "thalach", "exang", "oldpeak", "slope", "ca", "thal", "target")

# Separando os dados de entrada e saida do dataset
X <- heart_data[,1:13]
Y <- heart_data[, 14]

# Normalizacao de dados de entrada
X <- apply(X, 2, normalize)

# Divisao dos dados em treinamento e teste
set.seed(123)
training_index <- sample(1:nrow(heart_data), 0.7 * nrow(heart_data))
test_index <- setdiff(1:nrow(heart_data), training_index)

X_training <- X[training_index,]
X_test <- X[test_index,]
Y_training <- Y[training_index]
Y_test <- Y[test_index]

# Montagem da arquitetura
number_neurons <- c(10, 30, 50) # Numero de neuronios em cada camada oculta

for (n in number_neurons) {
  model <-  mlp (X_training, Y_training, size=c(n), maxit=20000, initFunc="Randomize_Weights", initFuncParams=c(-5, 5),
                 learnFunc="Std_Backpropagation", learnFuncParams=c(0.001, 0.5),
                 updateFunc="Topological_Order", updateFuncParams=c(0),
                 hiddenActFunc="Act_Logistic",
                 shufflePatterns=TRUE, linOut=TRUE)

  # Previsao do conjunto de teste
  Y_prediction <- predict(model, X_test)

  # Calculo do erro medio e do desvio padrao
  Error <- Y_prediction - Y_test
  Mean_Error <- mean(Error)
  SD_Error <- sd(Error)

  # Resultados
  cat("Numero de Neuronios: ", n, "\n")
  cat("Erro Medio: ", Mean_Error, "\n")
  cat("Desvio Padrao: ", SD_Error, "\n\n")
}
```

Figura 4: Implementação da MLP para o segundo exemplo

Após a separação dos dados presentes no início da Figura 4, foi também realizada a aplicação da MLP para diferentes arquiteturas, em que cada uma apresenta um determinado número de neurônios. Escolhidos também 10, 30 e 50 neurônios para a camada oculta, o objetivo de segundo exemplo, é de medir a precisão de classificação do parâmetro target, (coluna 14 do dataset). Abaixo, é apresentado o resultado obtido para o erro médio de classificação, com o respectivo desvio padrão para cada arquitetura:

```
Numero de Neuronios:  10
Erro Medio:  0.004878394
Desvio Padrao:  0.4859598

Numero de Neuronios:  30
Erro Medio:  -0.03288969
Desvio Padrao:  0.5273399

Numero de Neuronios:  50
Erro Medio:  -0.08266069
Desvio Padrao:  1.151709
```

Figura 5: Erro Médio e Desvio Padrão obtido nas diferentes arquiteturas

Diferentemente do primeiro exemplo, a arquitetura que apresentou o menor número de neurônios na camada oculta (n = 10), apresentou o desempenho mais satisfatório, como pode ser visto logo acima, na Figura 5. Nas arquiteturas seguintes, foi observado um aumento constante e gradual, tanto do Erro Médio quanto do Desvio Padrão, dando fortes indícios de um início de Overfitting do Modelo.