Exemplos de uso do pacote NeuralNet

Adriano VW

5/6/2021

Introdução

No R, usando o Rstudio, usamos o pacote **neuralnet** para ter a disposição vários modelos de Redes Neurais Artificiais. Também vamos usar a biblioteca **caret** para usar as funções de avaliação da classificação.

```
library(neuralnet)
library(caret)
```

Em seguida temos a definição do diretório de trabalho **setwd**, e a limpeza de todas as variáveis do ambiente **rm(list = ls())**. Como dados vamos usar o dataset **iris** o qual é dividido em 2/3 para treinamento e 1/3 para teste. Veja na ajuda como operam as funções **sample** e **nrow**.

```
rm(list = ls())
# Split data
train_idx <- sample(nrow(iris), 2/3 * nrow(iris))
iris_train <- iris[train_idx, ]
iris_test <- iris[-train_idx, ]
```

Classficação Binária

A principal função para criar a rede neural é a **neuralnet**. No exemplo aqui apresentado, coloquei vários dos parâmetros explicitamente na função, mas existem outros, para ver todos parâmetros e suas opções investigue a ajuda da função. Veja que é necessário informar uma fórmula, a qual denominamos **f**. Nesse exemplo estamos criando um modelo que tenta classificar a espécie **setosa**, a partir dos atributos **Petal.Length + Petal.Width**

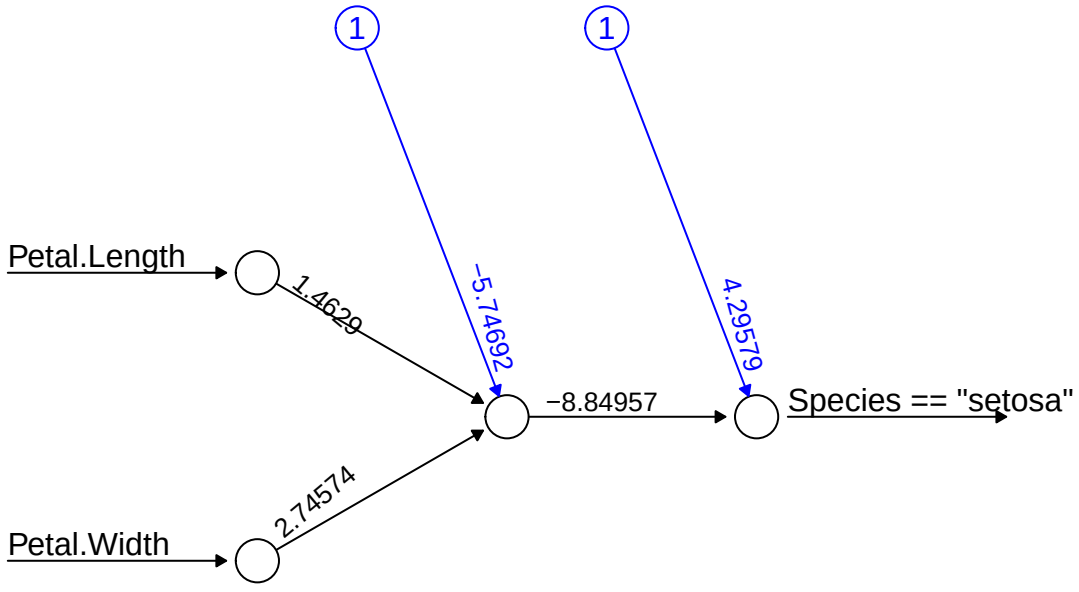
Para testar o modelo, usamos a função **predict**, a qual usa o modelo inferido **nn** e os dados de teste para avaliar a performance do modelo. Uma matriz de confusão e várias estatísticas são mostradas e por fim a rede inferida é plotada. Veja que escolhemos somente uma camada escondida, com somente um nó.

```
f<- as.formula("Species == \"setosa\" ~ Petal.Length + Petal.Width")
nn <- neuralnet(f,
                iris_train,
                hidden = c(1),
                threshold = 0.01,
                stepmax = 1e+05,
                learningrate=0.1,
                algorithm = "backprop",
                linear.output = FALSE)
pred <- predict(nn, iris_test)
result<-table(iris_test$Species == "setosa", pred[, 1] > 0.5)
confusionMatrix(result)
```

```
## Confusion Matrix and Statistics
```

```
##
##
##          FALSE TRUE
##   FALSE    33    0
##   TRUE      0   17
##
##                Accuracy : 1
##                  95% CI : (0.9289, 1)
##     No Information Rate : 0.66
##     P-Value [Acc > NIR] : 9.488e-10
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.00
##             Specificity : 1.00
##          Pos Pred Value : 1.00
##          Neg Pred Value : 1.00
##              Prevalence : 0.66
##          Detection Rate : 0.66
##    Detection Prevalence : 0.66
##       Balanced Accuracy : 1.00
##
##        'Positive' Class : FALSE
##
```

```
plot(nn,rep = "best")
```



Error: 0.014502 Steps: 1057

Classificação multiclasse

Nesse segundo exemplo utilizamos os mesmos dados do exemplo anterior, porém nesse caso queremos classificar a espécie em uma das três possibilidades. Veje que na fórmula, dessa vez já escrita dentro da função **neuralnet**, temos três variáveis dependentes (resposta), *setosa, versicolor e virginica* e três variáveis independentes (explicativas).

Depois de obter o modelo, novamente utilizamos funções para fazer a predição usando os dados de teste e verificar as medidas de avaliação do modelo inferido. Note, que nos dois exemplos, os dados de teste não são utilizados para inferência do modelo. Também, como não estamos definindo o valor da semente aleatória da simulação, cada vez que o exemplo for executado, será encontrada um modelo, e consequentemente, uma avaliação diferente.

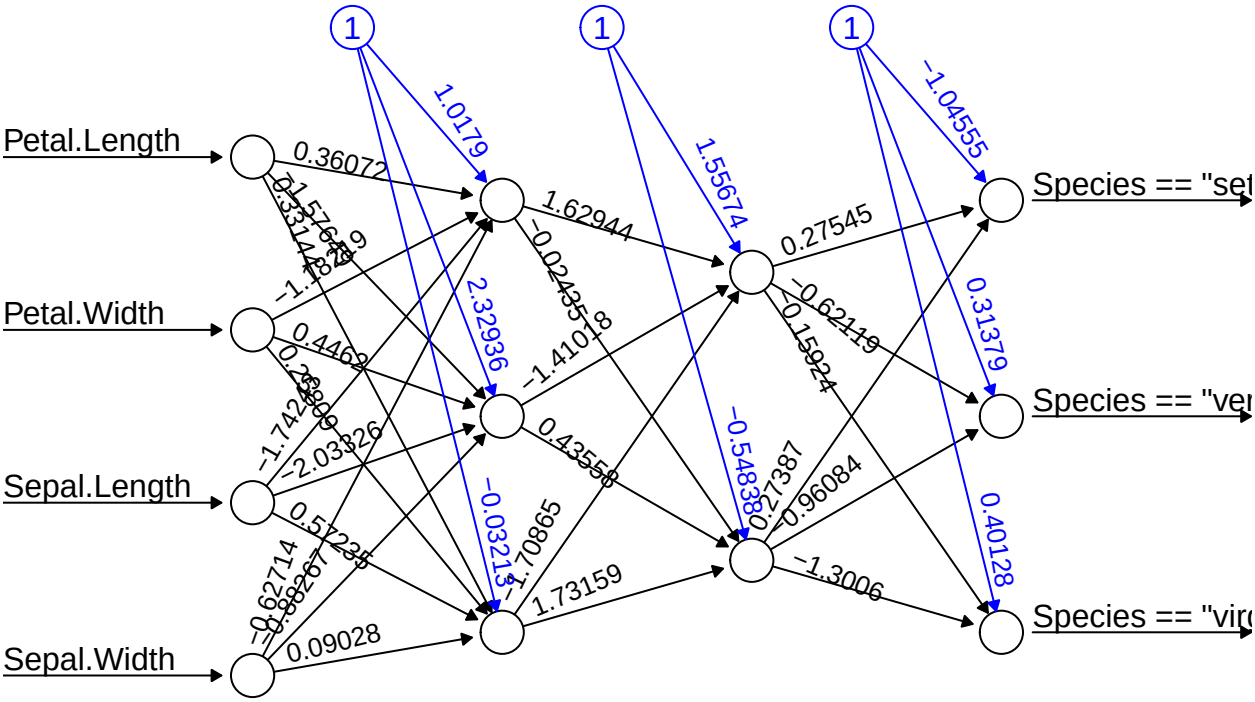
Veja como no parâmetro **hidden** são definidos os números de camadas escondidas e também o número de nós em cada camada. Experimente mudar esses números e veja como muda o erro do modelo inferido.

```
nn2 <- neuralnet((Species == "setosa") +
                 (Species == "versicolor") +
                 (Species == "virginica") ~
                  Petal.Length +
                  Petal.Width+Sepal.Length+
                  Sepal.Width,
                iris_train,
                hidden = c(3,2),
                threshold = 0.1,
                stepmax = 1e+05,
                learningrate=0.01,
                algorithm = "backprop",
                linear.output = FALSE)
pred2 <- predict(nn2, iris_test)

a<-apply(pred2, 1, which.max)
a[a==1]<-"setosa"
a[a==2]<-"versicolor"
a[a==3]<-"virginica"
a<-factor(a,levels = c("setosa","versicolor","virginica"))
result2<-table(iris_test$Species,a)
confusionMatrix(result2)
```

```
## Confusion Matrix and Statistics
##
##             a
##              setosa versicolor virginica
##   setosa          0          0        17
##   versicolor      0          0        17
##   virginica       0          0        16
##
## Overall Statistics
##
##                Accuracy : 0.32
##                  95% CI : (0.1952, 0.467)
##     No Information Rate : 1
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0
##
```

```
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                     NA                NA              0.32
## Specificity                   0.66              0.66                NA
## Pos Pred Value                  NA                NA                NA
## Neg Pred Value                  NA                NA                NA
## Prevalence                    0.00              0.00              1.00
## Detection Rate                0.00              0.00              0.32
## Detection Prevalence          0.34              0.34              0.32
## Balanced Accuracy               NA                NA                NA
```

```
plot(nn2,rep = "best")
```



Error: 33.350131 Steps: 81

Regressão

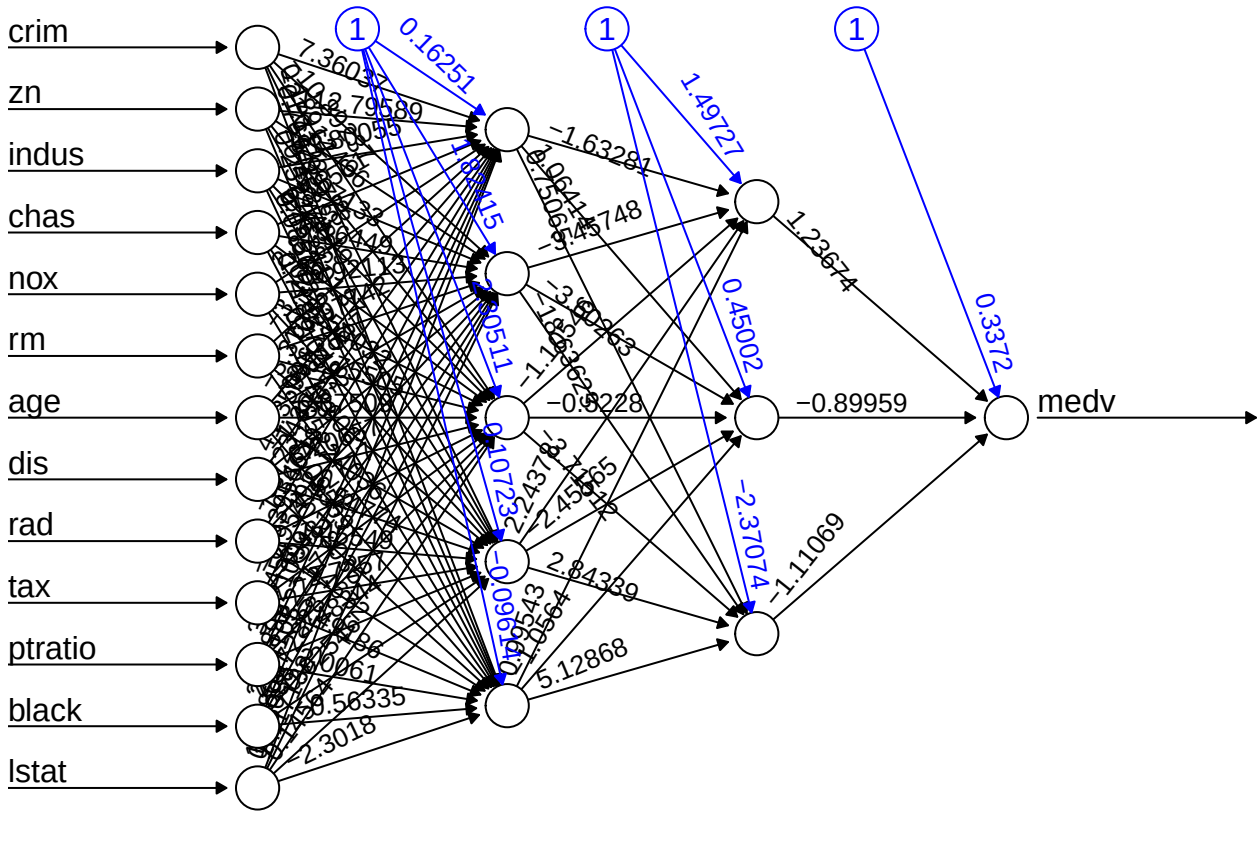
As Redes Neurais Artificiais também podem ser empregadas em problemas de regressão, onde a variável de resposta é contínua. No nosso exemplo vamos usar os dados **Boston** do pacote **MASS**. Uma recomendação importante para o treinamento de Redes Neurais Artificiais é que os valores de cada variável tenham valores na mesma escala. Para tanto nesse problema os valores são padronizados para o intervalo $[0, 1]$. Depois os dados são separados em treinamento e teste e a rede neural é treinada.

```
rm(list = ls())
library(MASS)

# Boston dataset from MASS
data <- Boston
```

```
# # Normalize the data
maxs <- apply(data, 2, max)
mins <- apply(data, 2, min)
scaled <- as.data.frame(scale(data, center = mins,
                              scale = maxs - mins))

# Split the data into training and testing set
index <- sample(1:nrow(data), round(2/3 * nrow(data)))
train_ <- scaled[index,]
test_ <- scaled[-index,]
nn <- neuralnet(medv ~ crim + zn + indus + chas + nox
                + rm + age + dis + rad + tax +
                  ptratio + black + lstat,
                data = train_, hidden = c(5, 3),
                linear.output = TRUE)
```

A rede inferida é a seguinte:

```
plot(nn,rep = "best")
```



A predição é realizada como nos casos de classificação, mas os valores obtidos precisam ser reescalados para os valores originais. Os valores obtidos e os de teste, já reescalados, são então utilizados para calcular o *mean squared error*.

```
#Predict on test data
pr.nn <- predict(nn, test_[,1:13])
# Compute mean squared error
```

```r
pr.nn_ <- pr.nn * (max(data$medv) - min(data$medv)) + min(data$medv)
test.r <- (test_$medv) * (max(data$medv) - min(data$medv)) +
  min(data$medv)
MSE.nn <- sum((test.r - pr.nn_)^2) / nrow(test_)
```

Uma figura com a dispersão entre os valores de teste e os valores preditos é também apresentada.

```r
par(mfrow=c(1,1),pty="s")
plot(test_$medv, as.vector(pr.nn), col = "red",
     main = 'Real vs Predicted')
abline(0, 1, lwd = 2)
```