# Métodos Quantitativos

Prof. Dr. A. L. Korzenowski

## Aula 07: Redes Neurais Artificiais

### Redes neurais

Em muitas situações, a relação funcional entre as covariáveis (também conhecidas como variáveis de entrada ou variáveis independentes) e as variáveis de resposta (também conhecidas como variáveis de saída ou variáveis dependentes) é de grande interesse.

As redes neurais artificiais podem ser aplicadas como aproximação a qualquer relação funcional complexa. Ao contrário dos modelos lineares generalizados, não é necessário que o tipo de relação entre variáveis dependentes e variáveis resposta seja, por exemplo, uma combinação linear. Isso faz das redes neurais artificiais uma valiosa ferramenta quantitativa. Esses modelos são, particularmente, extensões diretas dos modelos lineares generalizados e podem ser aplicados de maneira semelhante.

Dados observados são usados para treinar a rede neural, fazendo assim com que a rede neural "aprenda" uma aproximação da relação entre as variáveis independentes e dependentes de forma iterativa por meio da adaptação contínua dos seus parâmetros. De fato, usando a nomenclatura estatística, os parâmetros do modelo são estimados por meio da amostra.

### Perceptron Multi-Camadas

O pacote neuralnet se concentra nos modelos Perceptron Multi-Camadas (Multi-Layer Perceptrons - MLP), os quais são úteis na modelagem por meio de relações funcionais entre as variáveis. A estrutura subjacente de um MLP é um grafo orientado, isto é, consiste de vértices (neurônios) e arestas (sinapses). Os neurônios são organizados em camadas, que são normalmente ligadas por sinapses. No pacote neuralnet, uma sinapse só pode se conectar a camadas posteriores.

A camada de entrada é constituída por todas as covariáveis (um neurônio para cada covariável) separadas por neurônios (camadas ocultas) até as variáveis resposta. Essas camadas intermédias são denominadas camadas ocultas (ou variáveis latentes), por não serem diretamente observáveis. As camadas de entrada e as camadas ocultas incluem um neurônio constante, o qual estará associado ao intercepto em modelos lineares, ou seja, não é diretamente influenciado por qualquer covariável.

Um peso (parâmetro) está associado a cada uma das sinapses, representando o efeito correspondente do neurônio e de todos os dados passam pela rede neural como sinais. Os sinais são processados inicialmente pela função de integração combinando todos os sinais de entrada e, em seguida, pela função de ativação transformando os resultados do neurônio.

O modelo perceptron multicamada mais simples consiste de uma única camada de entrada com $n$ covariáveis e uma camada de saída com um único neurônio de saída. Esse modelo calcula a seguinte função:

$$o(\mathbf{x}) = f\left(w_0 + \sum_{i=1}^{n} w_i x_i\right) = f(w_0 + \mathbf{w}^T\mathbf{x}),$$

onde $w_0$ denota o intercepto, $\mathbf{w} = (w_1, \ldots, w_n)$ o vetor de todos os demais pesos (parâmetros), e $\mathbf{x} = (x_1, \ldots, x_n)$ o vetor de todas as covariáveis.

A função é matematicamente equivalente à estrutura padrão do modelo linear generalizado com função de ligação $f^{-1}(.)$. Portanto, todos os pesos calculados são, neste caso, equivalentes aos parâmetros da regressão via MLG. Para aumentar a flexibilidade da modelagem mais camadas ocultas podem ser incluídas, aumentando assim a "não-linearidade" do modelo. No entanto, Hornik et al. (1989) mostraram que uma única camada oculta é suficiente para modelar qualquer função contínua por partes.

O modelo perceptron multicamada com uma camada oculta consistindo de $J$ neurônios calcula a seguinte função:

$$0(\mathbf{x}) = f\left(w_0 + \sum_{j=1}^{J} w_j f\left(w_{0j} + \sum_{i=0}^{n} w_{ij} x_i\right)\right) = f\left(w_0 + \sum_{j=1}^{J} w_j f(w_{0j} + \mathbf{w_j}^T \mathbf{x})\right)$$

onde $w_0$ denota o intercepto do neurônio de saída e $w_{0j}$ representa o intercepto do $j$-ésimo neurônio oculto. Além disso, $w_j$ denota o peso sináptico correspondente à sinapse começando no $j$-ésimo neurônio oculto e que conduz ao neurônio de saída. $\mathbf{w_j} = (w_{1j}, \ldots, w_{nj})$ o vetor de todos os pesos sinápticos correspondentes às sinapses que conduzem ao $j$-ésimo neurônio oculto, e $\mathbf{x} = (x_1, \ldots, x_n)$ é o vetor de todas as covariáveis.

Apesar das redes neurais serem extensões diretas dos MLG, os parâmetros não podem ser interpretados da mesma maneira.

De maneira formal, todos os neurônios ocultos e os neurônios de saída calculam a seguinte função: $f(g(z_0, z_1, \ldots, z_k)) = f(g(\mathbf{z}))$ a partir das saídas de todos os neurônios precedentes $z_0, z_1, \ldots, z_k$, onde $g : \mathbb{R}^{k+1} \to \mathbb{R}$ representa a função de integração e $f : \mathbb{R} \to \mathbb{R}$ é a função de ativação. O neurônio unitário $z_0$ é uma constante o qual está relacionado com o conceito de intercepto em modelos de regressão.

A função de integração é, muitas vezes, definida como $g(\mathbf{z}) = w_0 z_0 + \sum_{i=1}^{n} w_i z_i = w0 + \mathbf{w}^{'}\mathbf{z}$. A função de ativação $f$ é geralmente uma função não-linear, não-decrescente, limitada e também diferenciável tal como o função logística $f(u) = 1/(1 + \exp^{-u})$ ou a tangente hiperbólica.

Essa função deve ser escolhida em relação à variável de resposta, assim como nos modelos lineares generalizados. A função logística é, por exemplo, apropriada para variáveis resposta binárias, uma vez que mapeia a saída de cada neurônio para o intervalo $[0, 1]$. O pacote neuralnet usa a mesma função de integração, bem como função de ativação para todos os neurônios.

## Aprendizado supervisionado

As redes neurais são estimadas por meio de um processo de treinamento da rede que usa os dados para "aprender". Especificamente, o pacote neuralnet concentra-se em algoritmos de aprendizado supervisionado. Estes algoritmos de aprendizagem são caracterizados pela utilização de saídas (outputs, resultados ou ainda variáveis dependentes), as quais são comparadas com o valor predito pela rede neural, adaptando dinamicamente os valores dos parâmetros de modo que o "erro" seja minimizado.

Os parâmetros de uma rede neural são os seus pesos. Todos os pesos são geralmente iniciados com valores aleatórios provenientes de uma distribuição normal padrão. Durante um processo iterativo de formação da rede, as seguintes etapas são repetidas:

- A rede neural calcula uma saída de $o(\mathbf{x})$ para as entradas dadas $\mathbf{x}$ e para os parâmetros correntes (pesos atuais). Se o processo de formação ainda não estiver concluído, os resultados previstos serão diferentes da saída $\mathbf{y}$ observada.
- Uma função de erro, como a Soma dos Quadrados dos Erros (SSE - Sum of Squared Errors):

$$E = \frac{1}{2}\sum_{l=1}^{L}\sum_{h=1}^{H}(o_{lh} - y_{lh})^2.$$

- Todos os pesos são adaptados segundo alguma regra de aprendizagem definida a priori.

O processo termina se um critério pré-determinado é atingido, por exemplo, se todas as derivadas parciais da função de erro com respeito aos pesos ($\partial E/\partial \mathbf{w}$) são menores do que um dado limiar. Um algoritmo de aprendizagem amplamente utilizado é o algoritmo resilient backpropagation.

**Nota**: source_url('https://gist.githubusercontent.com/fawda123/7471137/raw/

466c1474d0a505ff044412703516c34f1a4684a5/nnet_plot_update.r')

## Uma aplicação de Regressão

Vamos revisitar o problema de previsão das milhas por galão, modelo que ajustamos na nossa aula de Modelos Lineares Generalizados. O consumo do veículo em *mpg* será predita pelas variáveis $hp + wt + vs$. Em seguida vamos dividir a base de dados em duas partes (treinamento e validação). Estimaremos um modelo de redes neurais com 1 camada com 4 neurônios e admitindo o número máximo de 1000 iterações com *threshold* igual a 0,1. Uma vez obtidos os valores para os parâmetros do modelo de redes neurais podemos analisar os resultados. O próximo passo é realizar a previsão para os dados de validação. Vamos ao sacrifício...

```
#Limpa o Workspace
rm(list=ls())

#Habilita o pacote car
if (!require(car)) install.packages('car')
require(car)
attach(mtcars)

dados<-mtcars
dados <- na.omit(dados)

indice <- sample(1:nrow(dados), trunc(nrow(dados)*0.7),
                 replace=FALSE)

#Dados para estimação
dados.Treina<-dados[indice,c(1,4,6,8)]

#Dados para validação
dados.Valida<-dados[-indice,c(1,4,6,8)]

# Habilita o pacote nnet
if (!require(nnet)) install.packages("nnet")
library(nnet)

nn <- nnet(mpg ~ hp + wt + vs, data=dados.Treina, size=4, linout=T,
       rang = 0.1, decay = 5e-2, maxit = 1000)
```

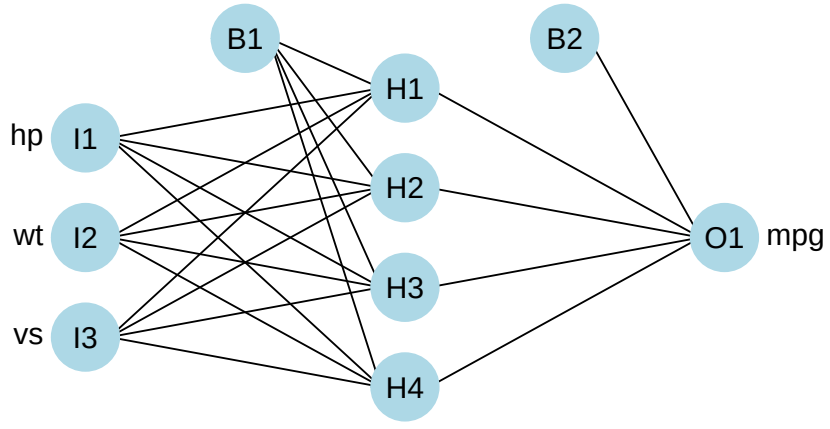
```
## # weights:  21
## initial  value 9806.488541
## iter  10 value 185.638859
```

```
## iter  20 value 164.381291
## iter  30 value 140.597162
## iter  40 value 139.805494
## final  value 139.802918
## converged
```

```
#Apresenta os valores para os pesos
nn$wts
```

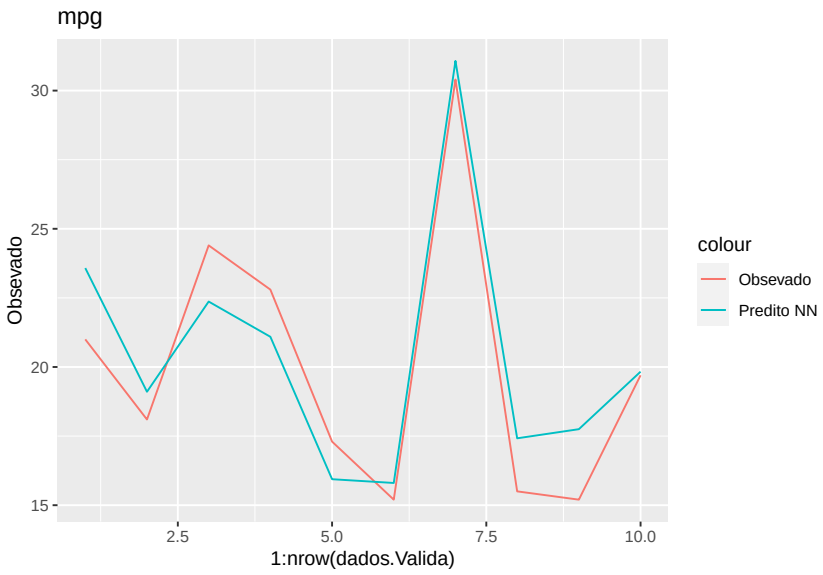
```
##  [1]  0.0024245017  0.1834869017  0.0071779186  0.0013069784  0.0010181055
##  [6]  0.2272568405  0.0077646202  0.0012107519 -0.0017272749  0.1392608296
## [11]  0.0011367000 -0.0004681009  3.4969763716 -0.0083615738 -1.0299542675
## [16] -0.0290520422  3.1704628652  3.1762874841  3.3113700633  3.1721726013
## [21] 22.8848238484
```

```
#Faz o gráfico do modelo
#import the function from Github
if (!require(devtools)) install.packages("devtools")
library(devtools)
source_url('https://gist.githubusercontent.com/fawda123/7471137/raw/466c1474d0a505ff044412703516c34f1a4
#plot  model
plot(nn, nid=F)

#Faz a previsão
previsao.nn<-predict(nn,dados.Valida[,2:4])


#Monta a base com as previsões
previsao.todos<-as.data.frame(cbind(dados.Valida[,1], previsao.nn))
colnames(previsao.todos)<-c("Obsevado","Predito.NN")

#Faz o gráfico
if (!require(ggplot2)) install.packages('ggplot2')
```




```
library("ggplot2")
ggplot(previsao.todos, aes(1:nrow(dados.Valida)), ) +
  geom_line(aes(y = Obsevado, colour = "Obsevado")) +
```

```
  geom_line(aes(y = Predito.NN, colour = "Predito NN")) +
  ggtitle("mpg")
```




```
#Encontra os erros
erro.nn<-previsao.todos[,1]-previsao.todos[,2]
RMSE <- sqrt(sum(erro.nn^2))
RMSE
```

```
## [1] 5.252281
```

## Uma aplicação em séries temporais

Vamos baixar dados de uma ação na bolsa, ajustar um modelo Autoregressivo de ordem 5, isto é,

$$y_t = \phi_0 + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \phi_3 y_{t-3} + \phi_4 y_{t-4} + \phi_5 y_{t-5}.$$

Nesse caso, o valor da ação no tempo $t$ seria predita pelos 5 valores (diários) que a antecedem. Em seguida vamos dividir a série temporal em duas partes (treinamento e teste). Estimaremos um modelo de redes neurais com 2 camadas com 7 e 3 neurônios e admitindo o número máximo de 100000 iterações com *threshold* igual a 1. Uma vez obtidos os valores para os parâmetros do modelo de redes neurais podemos analisar os resultados. O próximo passo é realizar a previsão para os dados de validação.

```
#Limpa o Workspace
rm(list=ls())

#Habilita o pacote quantmod
if (!require(quantmod)) install.packages('quantmod')
library(quantmod)

#Início do período de interesse
inicio = as.Date("2015-01-01")
```

```r
#Fim do período de interesse
fim = as.Date("2020-02-28")

#Obtêm os dados da ITUB4
getSymbols("ITUB4.SA", src="yahoo",from=inicio,to=fim)
```

```
## [1] "ITUB4.SA"
```

```r
#Dados para o neuralnet
t0<-as.numeric(Cl(ITUB4.SA))            #Cinco dias antes
t0<-t0[-((length(t0)-4):length(t0))]

t1<-as.numeric(Cl(ITUB4.SA)) [-1]       #Quatro dias antes
t1<-t1[-((length(t1)-3):length(t1))]

t2<-as.numeric(Cl(ITUB4.SA)) [-c(1,2)]  #Três dias antes
t2<-t2[-((length(t2)-2):length(t2))]

t3<-as.numeric(Cl(ITUB4.SA)) [-(1:3)]   #Dois dias antes
t3<-t3[-((length(t3)-1):length(t3))]

t4<-as.numeric(Cl(ITUB4.SA)) [-(1:4)]   #Um dia antes
t4<-t4[-length(t0)]

t5<-as.numeric(Cl(ITUB4.SA)) [-(1:5)]   #Variável dependente

#Cria a base
dados<-cbind(t1,t2,t3,t4,t5)
dados<-na.omit(dados)

#Dados para estimação
dados.Treina<-dados[1:trunc(nrow(dados)*.7),]

#Dados para validação
dados.Valida<-dados[(trunc(nrow(dados)*.7)+1):nrow(dados),]

# Habilita o pacote nnet
if (!require(nnet)) install.packages("nnet")
library(nnet)

nn <- nnet(t5 ~ t4 + t3 + t2 + t1, data=dados.Treina, size=7, linout=T,
       rang = 0.1, decay = 5e-2, maxit = 1000)
```

```
## # weights:  43
## initial  value 512409.904975
## iter  10 value 25563.204324
## iter  20 value 25483.090167
## iter  30 value 20413.026199
## iter  40 value 1089.547946
## iter  50 value 430.180864
## iter  60 value 298.983364
## iter  70 value 230.105706
## iter  80 value 222.686991
```

```
## iter  90 value 213.887809
## iter 100 value 208.299231
## iter 110 value 204.880256
## iter 120 value 195.151044
## iter 130 value 188.906346
## iter 140 value 185.697827
## iter 150 value 184.351051
## iter 160 value 182.572880
## iter 170 value 181.981912
## iter 180 value 181.636043
## iter 190 value 181.322627
## iter 200 value 180.890242
## iter 210 value 180.676886
## iter 220 value 180.135645
## iter 230 value 179.702529
## iter 240 value 178.425217
## iter 250 value 176.708694
## iter 260 value 176.291845
## iter 270 value 175.966430
## iter 280 value 175.794700
## iter 290 value 175.750782
## iter 300 value 175.704461
## iter 310 value 175.620332
## iter 320 value 175.604057
## iter 330 value 175.485854
## iter 340 value 175.300001
## iter 350 value 175.274399
## iter 360 value 175.255316
## iter 370 value 175.218199
## iter 380 value 174.751931
## iter 390 value 174.482235
## iter 400 value 174.385005
## iter 410 value 174.341838
## iter 420 value 174.227949
## iter 430 value 174.142617
## iter 440 value 174.121467
## iter 450 value 174.094408
## iter 460 value 174.041321
## iter 470 value 173.998706
## iter 480 value 173.987152
## iter 490 value 173.981460
## iter 500 value 173.979002
## iter 510 value 173.968347
## iter 520 value 173.963769
## iter 530 value 173.961376
## final  value 173.961146
## converged
```

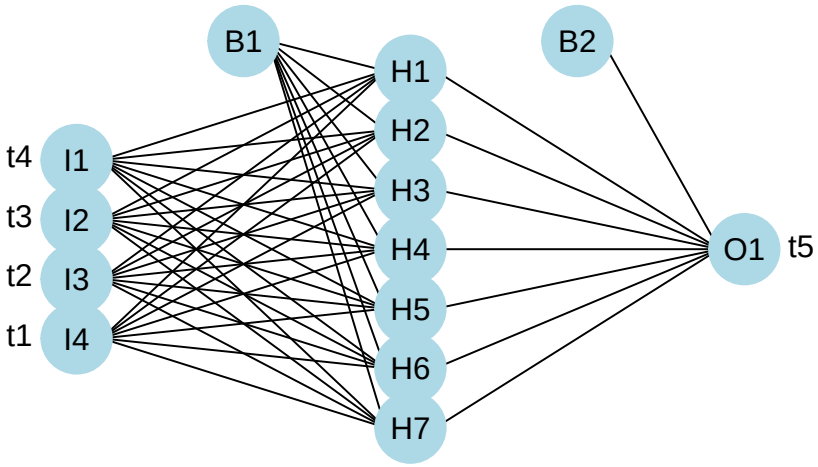
```
#Apresenta os valores para os pesos
nn$wts
```

```
##  [1] -4.04092022 -0.44343265  0.58391915  0.13184614 -0.07489589 -3.83086943
##  [7]  0.55284994 -0.13244742 -0.37762290  0.16774196 -0.36452330 -0.13842037
## [13]  2.35250605 -1.50229910 -0.61121506 -2.11306042  0.99169137 -0.69107027
```

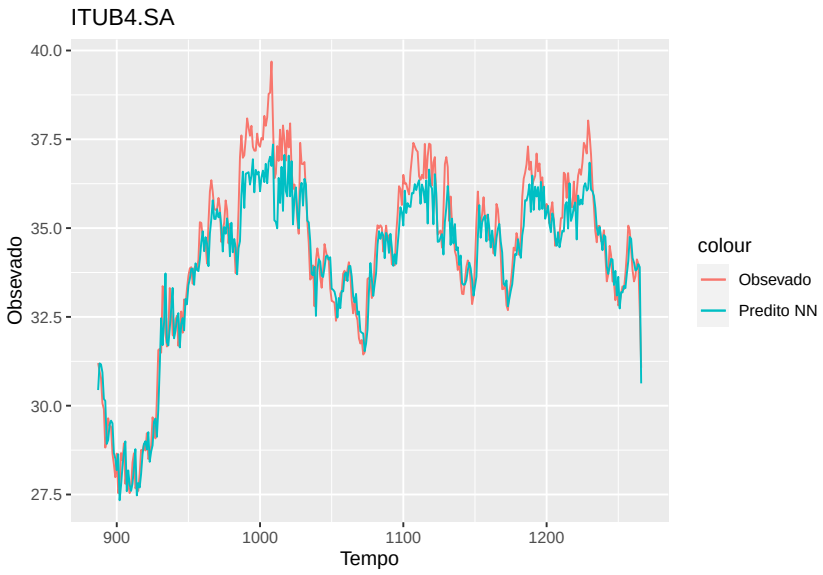
```
## [19] -0.28787978  0.07596010 -1.98402846 -0.19001721 -0.02658900  0.66917072
## [25] -0.27853048 -8.96357416  0.03126836  0.48432300 -0.05041008 -0.18295913
## [31] -1.61767585 -0.27291942 -1.64048547  1.28041451  0.64889574  3.34482025
## [37]  6.48117003  6.30799824  2.81497644  5.41556502  4.99100375  9.37037169
## [43]  3.71184356
```

```r
#Faz o gráfico do modelo
#import the function from Github
if (!require(devtools)) install.packages("devtools")
library(devtools)
source_url('https://gist.githubusercontent.com/fawda123/7471137/raw/466c1474d0a505ff044412703516c34f1a4
```

```r
#plot  model
plot(nn, nid=F)
```




```r
#Faz a previsão
previsao.nn<-predict(nn,dados.Valida[,1:4])

#Monta a base com as previsões
Tempo<-seq((trunc(nrow(dados)*.7)+1),nrow(dados))
previsao.todos<-as.data.frame(cbind(Tempo, dados.Valida[,5], previsao.nn))
colnames(previsao.todos)<-c("Tempo","Obsevado","Predito.NN")

#Faz o gráfico
if (!require(ggplot2)) install.packages('ggplot2')
library("ggplot2")
ggplot(previsao.todos, aes(Tempo)) +
  geom_line(aes(y = Obsevado, colour = "Obsevado")) +
  geom_line(aes(y = Predito.NN, colour = "Predito NN")) +
  ggtitle("ITUB4.SA")
```




```
#Encontra os erros
erro.nn<-previsao.todos[,2]-previsao.todos[,3]
RMSE <- sqrt(sum(erro.nn^2))
RMSE
```

```
## [1] 15.31343
```

## Uma aplicação de classificação binária

Aqui vamos revisitar o exemplo do Seguro Residencial e tentar estimar a suspeita de fraude utilizando Rede Neural Artificial.

Para tanto vamos dividir a Base em Seguro.Treino e Seguro.Valida, vamos configurar a rede neural e treiná-la e após utilizar uma matriz de confusão para validar os resultados da classificação efetuada.

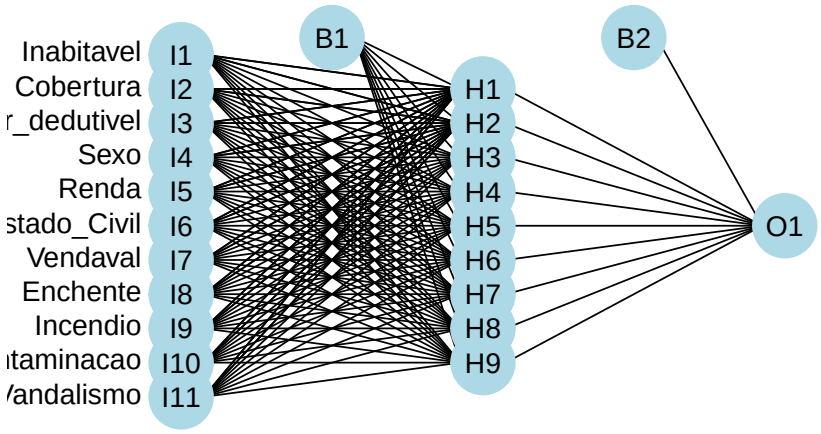
```
#Limpa o Workspace
rm(list=ls())

# Habilita o pacote readxl
if (!require(readxl)) install.packages('readxl')
library(readxl)

# Carrega a base de dados
Seguro_Residencial<-read_excel("Seguro_Residencial.xlsx")

for(i in 1:5){
  Seguro_Residencial<-cbind(Seguro_Residencial, Seguro_Residencial$Tipo==i)
}
names(Seguro_Residencial)[20:24]<-c("Vendaval","Enchente","Incendio",
                                    "Contaminacao","Vandalismo")
Seguro_Residencial <- Seguro_Residencial[,c(6,4,8,9,11,16,17,20:24)]

indice <- sample(1:nrow(Seguro_Residencial), trunc(nrow(Seguro_Residencial)*0.7),
                 replace=FALSE)
```

```r
#Dados para estimação
dados.Treina<-Seguro_Residencial[indice,]

#Dados para validação
dados.Valida<-Seguro_Residencial[-indice,]

# Habilita o pacote nnet
if (!require(nnet)) install.packages("nnet")
library(nnet)

nn <- nnet(dados.Treina[,2:ncol(dados.Treina)], dados.Treina[,1],
           size=9, linout=T, rang = 0.1, decay = 5e-2, maxit = 100000)
```

```
## # weights:  118
## initial  value 312.837848
## iter  10 value 292.701421
## iter  20 value 287.654659
## iter  30 value 287.388450
## iter  40 value 287.179022
## iter  50 value 286.945610
## iter  60 value 286.526212
## iter  70 value 286.121227
## iter  80 value 285.090968
## iter  90 value 284.294195
## iter 100 value 282.795178
## iter 110 value 282.654900
## iter 120 value 282.457866
## iter 130 value 282.362553
## iter 140 value 281.816925
## iter 150 value 281.069424
## iter 160 value 280.851733
## iter 170 value 280.815203
## iter 180 value 280.653950
## iter 190 value 279.861203
## iter 200 value 279.467143
## iter 210 value 279.447060
## iter 220 value 279.412457
## iter 230 value 279.025702
## iter 240 value 278.779904
## iter 250 value 278.735570
## iter 260 value 278.711872
## iter 270 value 278.503856
## iter 280 value 278.491053
## final  value 278.490793
## converged
```

```r
# Habilita o pacote RSNNS
#if (!require(RSNNS)) install.packages("RSNNS")
#library(RSNNS)
#nn <- mlp(dados.Treina[,2:ncol(dados.Treina)], dados.Treina[,1],
#           size=15, linout=T, maxit = 1000)
```

```r
#Faz o gráfico do modelo
#import the function from Github
if (!require(devtools)) install.packages("devtools")
library(devtools)
source_url('https://gist.githubusercontent.com/fawda123/7471137/raw/466c1474d0a505ff044412703516c34f1a4
#plot  model
plot.nnet(nn, nid=F)
```




```r
#Faz a previsão
previsao.nn<-predict(nn,dados.Valida[,2:ncol(dados.Valida)])

#Converte previsão em resposta binária
pred <- ifelse(previsao.nn > 0.5, 1, 0)

#Constrói a matriz da confusão
table(dados.Valida$Fraudulento, pred)
```

```
##    pred
##        0    1
##   0 1176    7
##   1  141    1
```