nn-for-multiple-regression.R

fhern

2020-09-24

```r
# En este ejemplo se muestra como usar nn para regresion
# el ejemplo esta basado en http://uc-r.github.io/ann_regression
# Los datos del ejemplo se van a simular pero tambien estan disponibles
# en un repo de github.
```

```r
# Simulando los datos ----------------------------------------------------
# Vamos a usar datos simulados de un modelo
# y ~ N(mu=4 - 3 * x1 + 3 * x2, sigma=6)

gen_dat <- function(n) {
  x1 <- runif(n=n, min=-5, max=6)
  x2 <- runif(n=n, min=-4, max=5)
  media <- 4 - 3 * x1 + 3 * x2
  y <- rnorm(n=n, mean=media, sd=6)
  marco_datos <- data.frame(y=y, x1=x1, x2=x2)
  return(marco_datos)
}

set.seed(1974)
datos <- gen_dat(n=100)
head(datos)
```

```
##            y        x1        x2
## 1  6.1603128  1.168256  3.816337
## 2 25.5220883 -1.718673  4.853366
## 3 -4.4961652  2.066200 -1.375973
## 4 11.4844365 -3.586915  1.902173
## 5  0.6206632 -1.789701 -2.445758
## 6  7.3881806 -3.232255 -2.769691
```

```r
# Los datos simulados estan disponibles tambien en la url de abajo.
datos <- read.table("https://raw.githubusercontent.com/fhernanb/datos/master/datos_regresion_mult_redes
                    header=TRUE)
head(datos)
```

```
##            y        x1        x2
## 1  6.1603130  1.168256  3.816337
## 2 25.5220900 -1.718673  4.853366
## 3 -4.4961650  2.066200 -1.375973
## 4 11.4844400 -3.586915  1.902173
## 5  0.6206632 -1.789701 -2.445758
## 6  7.3881810 -3.232255 -2.769691
```

```r
# Visualizando los datos ------------------------------------------------
library(scatterplot3d)
scatterplot3d(x=datos$x1, y=datos$x2, z=datos$y,
              pch=16, cex.lab=1,
              highlight.3d=TRUE, type="h", xlab='x1',
              ylab='x2', zlab='y')


# Transformando los datos -----------------------------------------------

# Vamos a usar una transformacion al intervalo (0, 1).
# A usted le queda de tarea probar con una transformacion (-1, 1)

scale01 <- function(x){
  (x - min(x)) / (max(x) - min(x))
}

library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
datis <- datos %>% mutate_all(scale01) # scaled data

# Vamos a explorar la media y varianza de los datos sin/con transformacion
# pero vamos a crear una funcioncita para esto.
funcioncita <- function(x) c(min=min(x), med=mean(x), max=max(x))

apply(datos, MARGIN=2, FUN=funcioncita) # sin transf
```

```
##              y         x1         x2
## min -23.036150 -4.9121230 -3.8890720
## med   4.091121  0.2248992  0.2339341
## max  33.525820  5.7757390  4.9648280
```

```r
apply(datis, MARGIN=2, FUN=funcioncita) # con transf
```

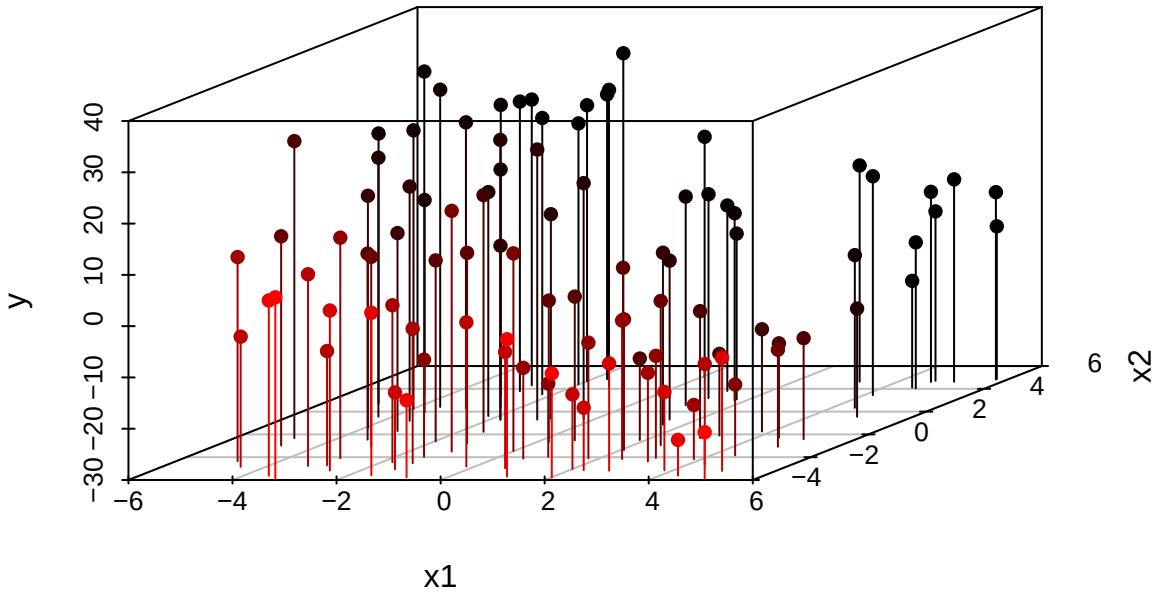
```
##             y        x1        x2
## min 0.0000000 0.0000000 0.0000000
## med 0.4796027 0.4806408 0.4656712
## max 1.0000000 1.0000000 1.0000000
```

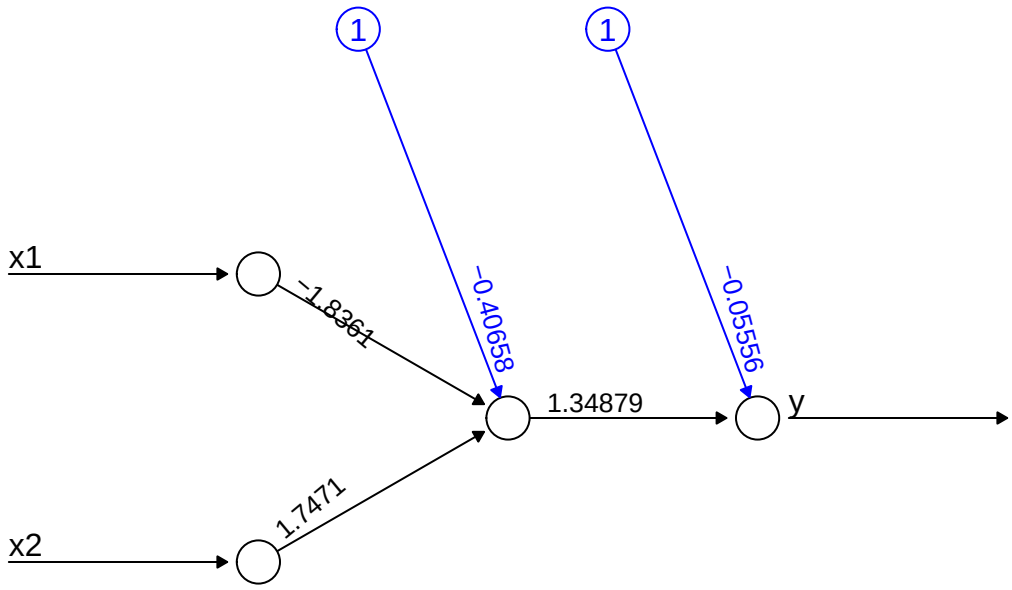
```r
# Ajustado el modelo con neuralnet ----------------------------------------

# Vamos a crear una red con 1 sola capa interna y 1 sola neurona
# funcion de activacion logistica

library(neuralnet)
```

```
##
## Attaching package: 'neuralnet'
```

```
## The following object is masked from 'package:dplyr':
##
##     compute
```

```r
mod1 <- neuralnet(y ~ x1 + x2, data=datis,
                  rep=1,
                  algorithm="rprop+",
                  err.fct="sse",
                  act.fct="logistic",
                  hidden=c(1))

# Dibujando la red entrenada
plot(mod1, rep = 'best')
```





Error: 0.545289 Steps: 692

```
# Para conocer la clase del objeto mod1
class(mod1)
```

```
## [1] "nn"
```

```
# Para ver los objetos dentro de mod1
names(mod1)
```

```
##  [1] "call"                "response"            "covariate"
##  [4] "model.list"          "err.fct"             "act.fct"
##  [7] "linear.output"       "data"                "exclude"
## [10] "net.result"          "weights"             "generalized.weights"
## [13] "startweights"        "result.matrix"
```

```
# Explorando los pesos para luego hacer operaciones con ellos
mod1$weights
```

```
## [[1]]
## [[1]][[1]]
##             [,1]
## [1,] -0.4065847
## [2,] -1.8361004
## [3,]  1.7470972
##
```

```
## [[1]][[2]]
##             [,1]
## [1,] -0.05556419
## [2,]  1.34878563
```

```r
# Haciendo predicciones manuales para la observacion k-esima
k <- 5
datis[k, ] # primera linea
```

```
##           y        x1        x2
## 5 0.4182459 0.2921465 0.1630145
```

```r
a <- mod1$weights[[1]][[1]][2, 1] * datis[k, 2] +
  mod1$weights[[1]][[1]][3, 1] * datis[k, 3] +
  mod1$weights[[1]][[1]][1, 1]

b <- exp(a) / (1 + exp(a))

b * mod1$weights[[1]][[2]][2, 1] + mod1$weights[[1]][[2]][1, 1]
```

```
## [1] 0.4045682
```

```r
# Haciendo las predicciones automaticamente
predict(mod1, newdata=datis[k ,]) # igual al manual
```

```
##        [,1]
## 5 0.4045682
```

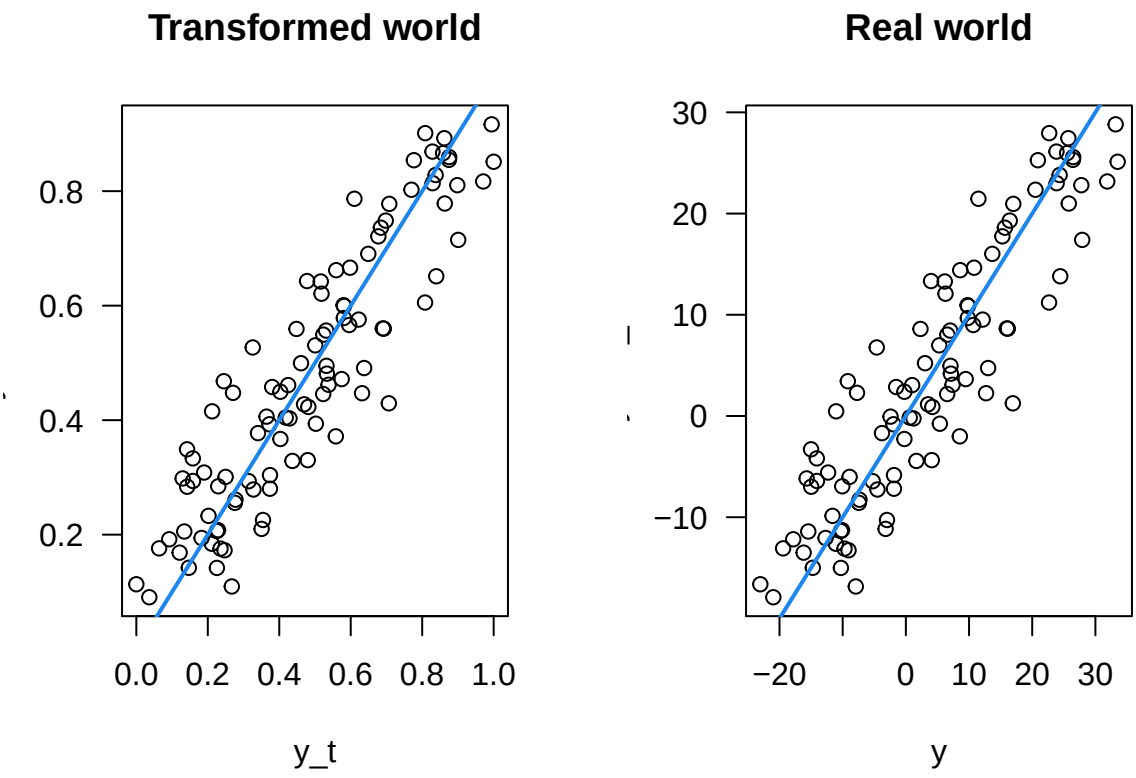
```r
# Creando un vector con todas las predicciones usando los datos transf
yhat1 <- predict(mod1, newdata=datis)

# Calculando el error
sum((datis$y - yhat1)^2) / 2
```

```
## [1] 0.5452892
```

```r
# Explorando las predicciones en el mundo transformado
par(mfrow=c(1, 2))

plot(x=datis$y, y=yhat1, las=1, xlab="y_t",
     main="Transformed world")
abline(a=0, b=1, col="dodgerblue2", lwd=2)
cor(x=datis[, 1], y=yhat1)
```

```
##           [,1]
## [1,] 0.9080662
```

```r
# Explorando las predicciones en el mundo normal (no transf)
# Debemos usar la transformada inversa
yhat1_nt <- yhat1 * (max(datos$y) - min(datos$y)) + min(datos$y)
plot(x=datos$y, y=yhat1_nt, las=1, xlab="y",
     main="Real world")
abline(a=0, b=1, col="dodgerblue2", lwd=2)
```




```
cor(x=datos$y, y=yhat1_nt)
```

```
##           [,1]
## [1,] 0.9080662
```

```
# Tarea: saque al menos UNA conclusion de este ejemplo.


# Ajustando el modelo con lm ----------------------------------------------

# Ahora vamos a ajustar el modelo usando lm

mod_lm <- lm(y ~ x1 + x2, data=datos)
yhat_lm <- predict(mod_lm)

# Calculando el MSE
sum((datos$y - yhat_lm)^2) / 2 # Depende de las unidade de Y
```

```
## [1] 1824.706
```

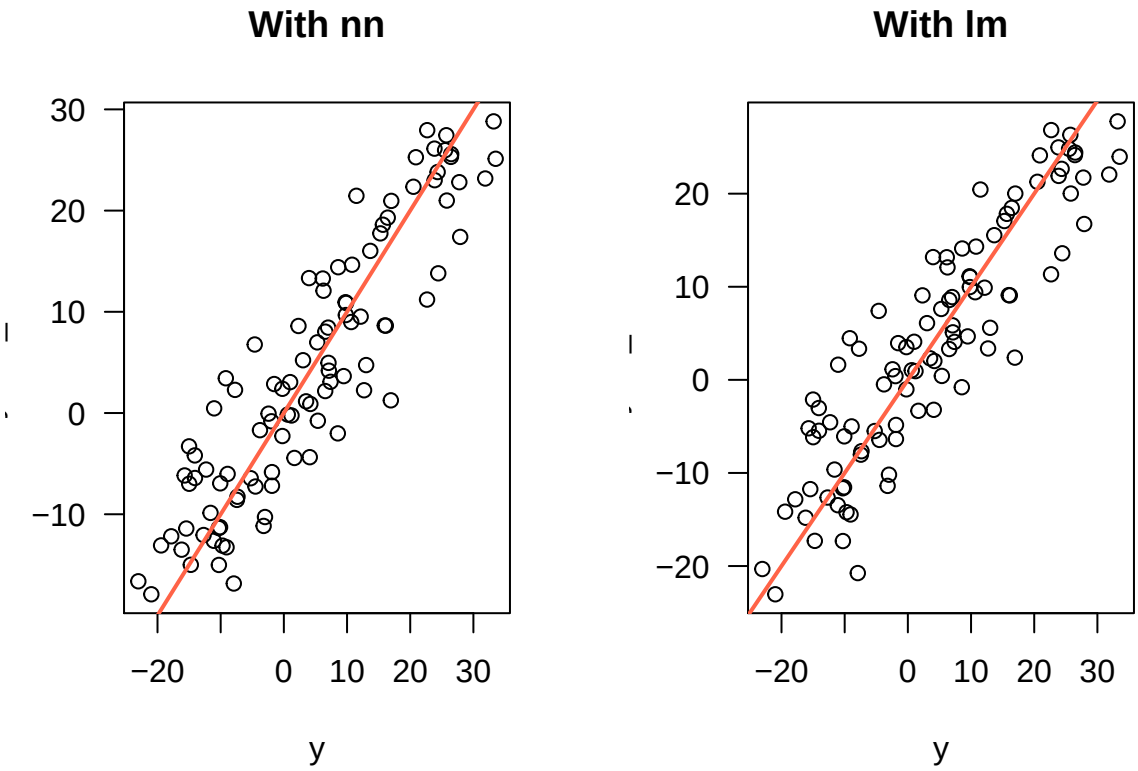
```
# Comparando modelo nn y lm -----------------------------------------------
cor(x=datos$y, y=yhat1_nt)
```

```
##           [,1]
## [1,] 0.9080662
```

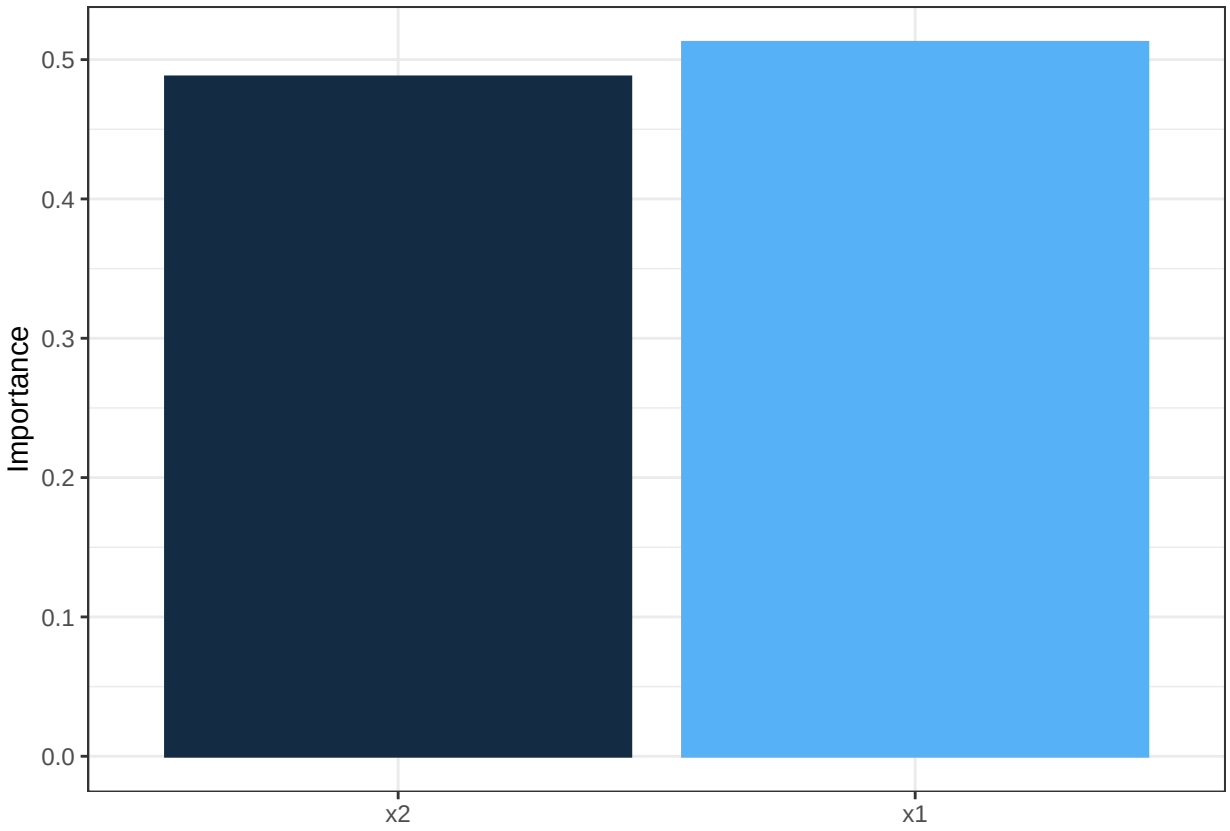
```r
cor(x=datos$y, y=yhat_lm)
```

```
## [1] 0.9036156
```

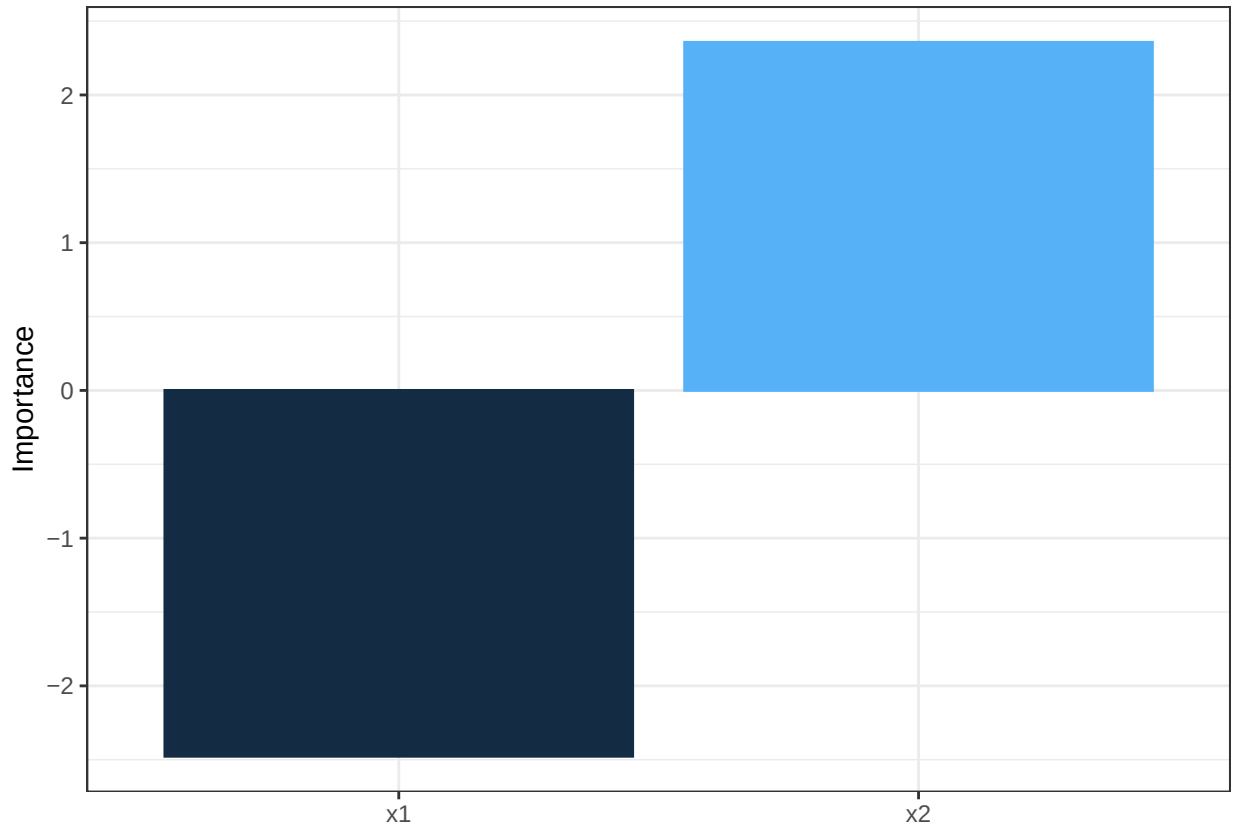
```r
par(mfrow=c(1, 2))

plot(x=datos$y, y=yhat1_nt, las=1, xlab="y", main="With nn")
abline(a=0, b=1, col="tomato", lwd=2)

plot(x=datos$y, y=yhat_lm, las=1, xlab="y", main="With lm")
abline(a=0, b=1, col="tomato", lwd=2)
```




```r
par(mfrow=c(1, 1))

# Tarea: saque otra conclusion de este ejemplo.

# Variable importance ----------------------------------------------------

# Para ver la importancia de las variables en la red usamos
library(NeuralNetTools)

garson(mod1) # Garson (1991). Interpreting neural network connection weights
```



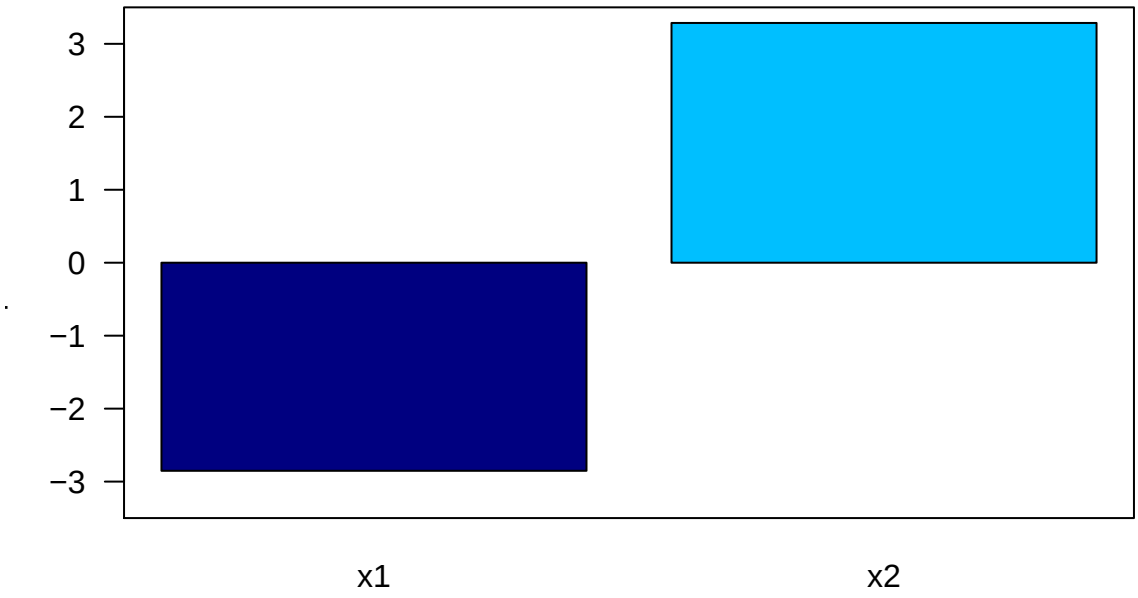

```
olden(mod1)  # Olden et al (2002). Illuminating the 'black-box'
```




```
# Tarea: Averiguar por que la altura de las barras son aprox 3 unidades

# Para ver la importancia de las variables en el modelo de regresion
summary(mod_lm)
```

```
##
## Call:
## lm(formula = y ~ x1 + x2, data = datos)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.6584  -3.5298   0.0918   3.5985  14.5483
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.9638     0.6171   6.424 4.95e-09 ***
## x1           -2.8522     0.1969 -14.488  < 2e-16 ***
## x2            3.2861     0.2191  14.997  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.134 on 97 degrees of freedom
## Multiple R-squared:  0.8165, Adjusted R-squared:  0.8127
## F-statistic: 215.8 on 2 and 97 DF,  p-value: < 2.2e-16
```

```r
# Para crear una figura similar a la de arriba pero para lm usamos
barplot(coef(mod_lm)[2:3], ylab="Importancia", las=1,
        ylim=c(-3.5, 3.5), col=c("navy", "deepskyblue"))
box()
```

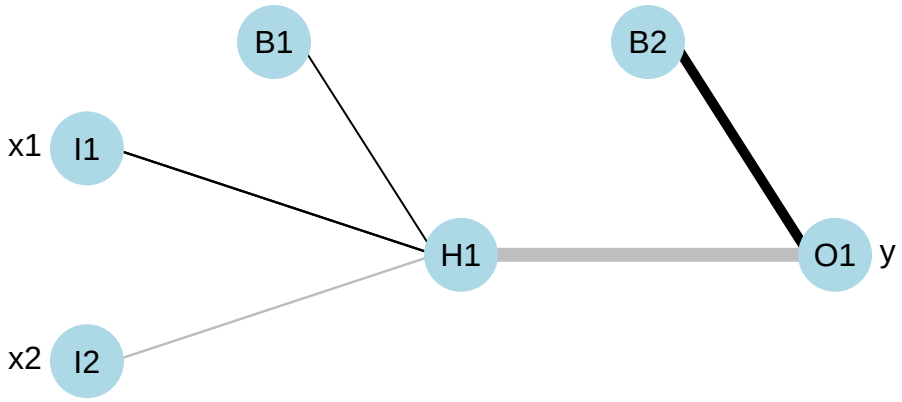



```r
# Ajustado el modelo con nnet ---------------------------------------------

# Vamos a crear una red con 1 sola capa interna y 1 sola neurona
# funcion de activacion logistica

library(nnet)
mod2 <- nnet(y ~ x1 + x2, data=datis,
             size=c(1),
             softmax=FALSE,
             maxit=1000)
```

```
## # weights:  5
## initial  value 6.640059
## iter  10 value 1.202272
## iter  20 value 1.136917
## iter  30 value 1.117777
## iter  40 value 1.112262
## iter  50 value 1.110837
## iter  60 value 1.109406
## iter  70 value 1.108376
```

```
## iter  80 value 1.107606
## iter  90 value 1.106862
## iter 100 value 1.106381
## iter 110 value 1.106009
## iter 120 value 1.105718
## iter 130 value 1.105379
## iter 140 value 1.105124
## iter 150 value 1.104523
## iter 160 value 1.104194
## iter 170 value 1.103737
## iter 180 value 1.103377
## iter 190 value 1.103222
## iter 200 value 1.103046
## iter 210 value 1.102869
## iter 220 value 1.102698
## iter 230 value 1.102614
## iter 240 value 1.102525
## iter 250 value 1.102461
## iter 260 value 1.102399
## iter 270 value 1.102353
## iter 280 value 1.102289
## iter 290 value 1.102175
## iter 300 value 1.102100
## iter 310 value 1.102017
## iter 320 value 1.101958
## iter 330 value 1.101867
## iter 340 value 1.101685
## iter 350 value 1.101643
## iter 360 value 1.101594
## iter 370 value 1.101572
## iter 380 value 1.101527
## iter 390 value 1.101493
## iter 400 value 1.101453
## iter 410 value 1.101408
## final  value 1.101376
## converged
```

```r
# Dibujando la red entrenada
NeuralNetTools::plotnet(mod2)
```




```
# Para conocer la clase del objeto mod2
class(mod2)
```

```
## [1] "nnet.formula" "nnet"
```

```
# Para ver los objetos dentro de mod2
names(mod2)
```

```
##  [1] "n"             "nunits"        "nconn"         "conn"
##  [5] "nsunits"       "decay"         "entropy"       "softmax"
##  [9] "censored"      "value"         "wts"           "convergence"
## [13] "fitted.values" "residuals"     "call"          "terms"
## [17] "coefnames"     "xlevels"
```

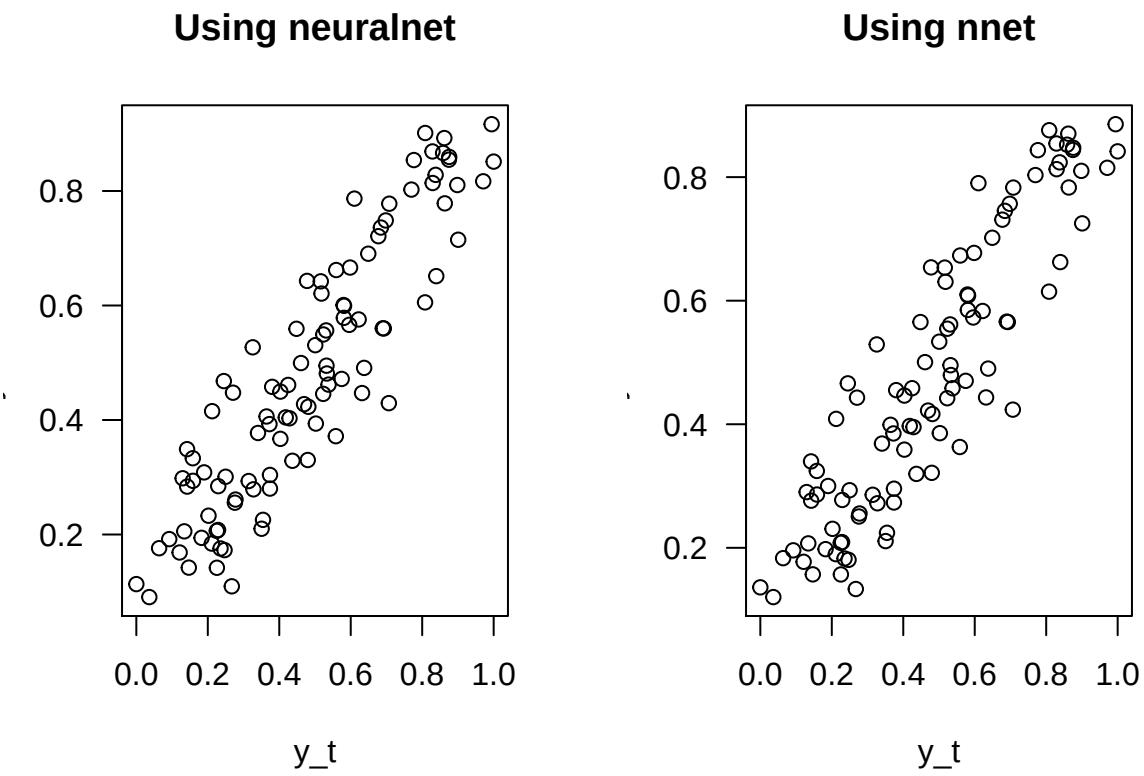
```
# Para ver los pesos dentro de la red
mod2$wts
```

```
## [1]   0.8047488   1.0098590  -0.9633683   8.2206473 -12.0030363
```

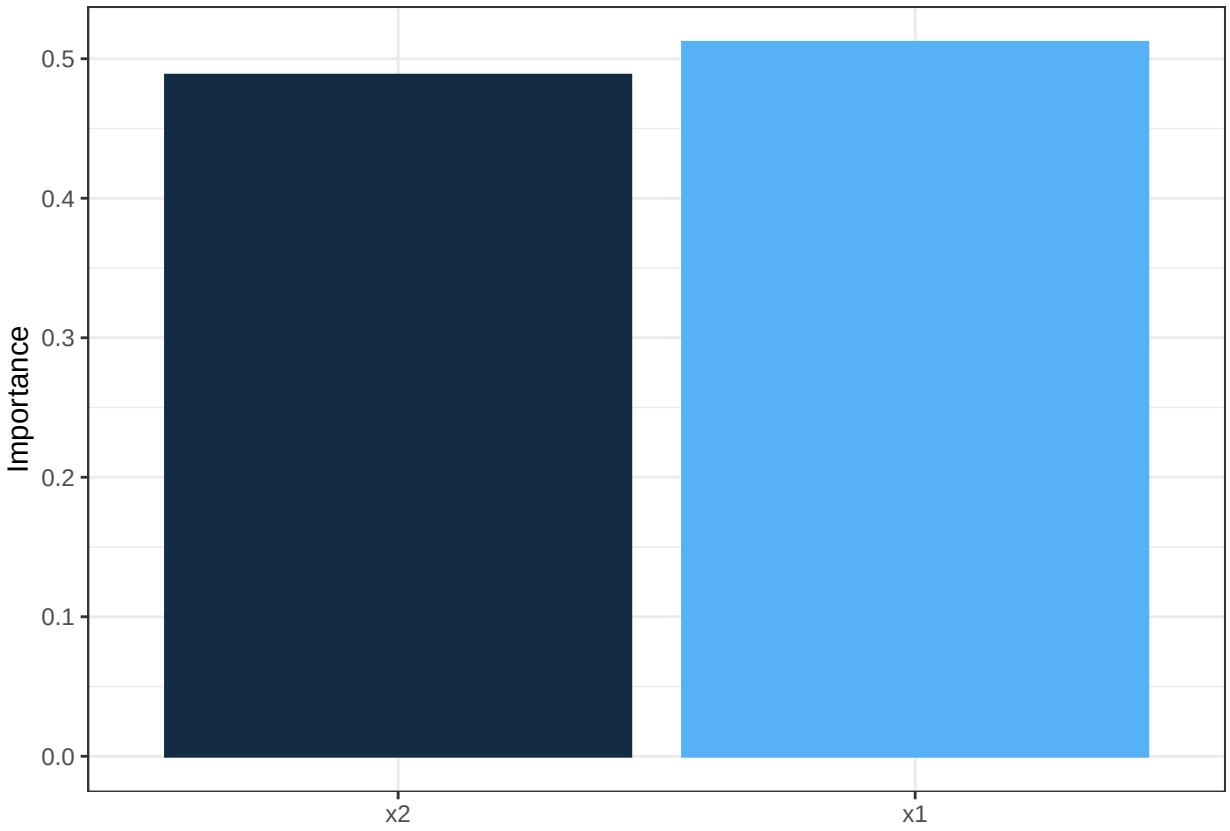
```
# Tarea: por que no son igualitos los pesos de ambas redes?

# Creando un vector con todas las predicciones usando los datos transf
yhat2 <- predict(mod2, newdata=datis)

# Calculando el error
sum((datis$y - yhat2)^2) / 2
```

```
## [1] 0.550688
```

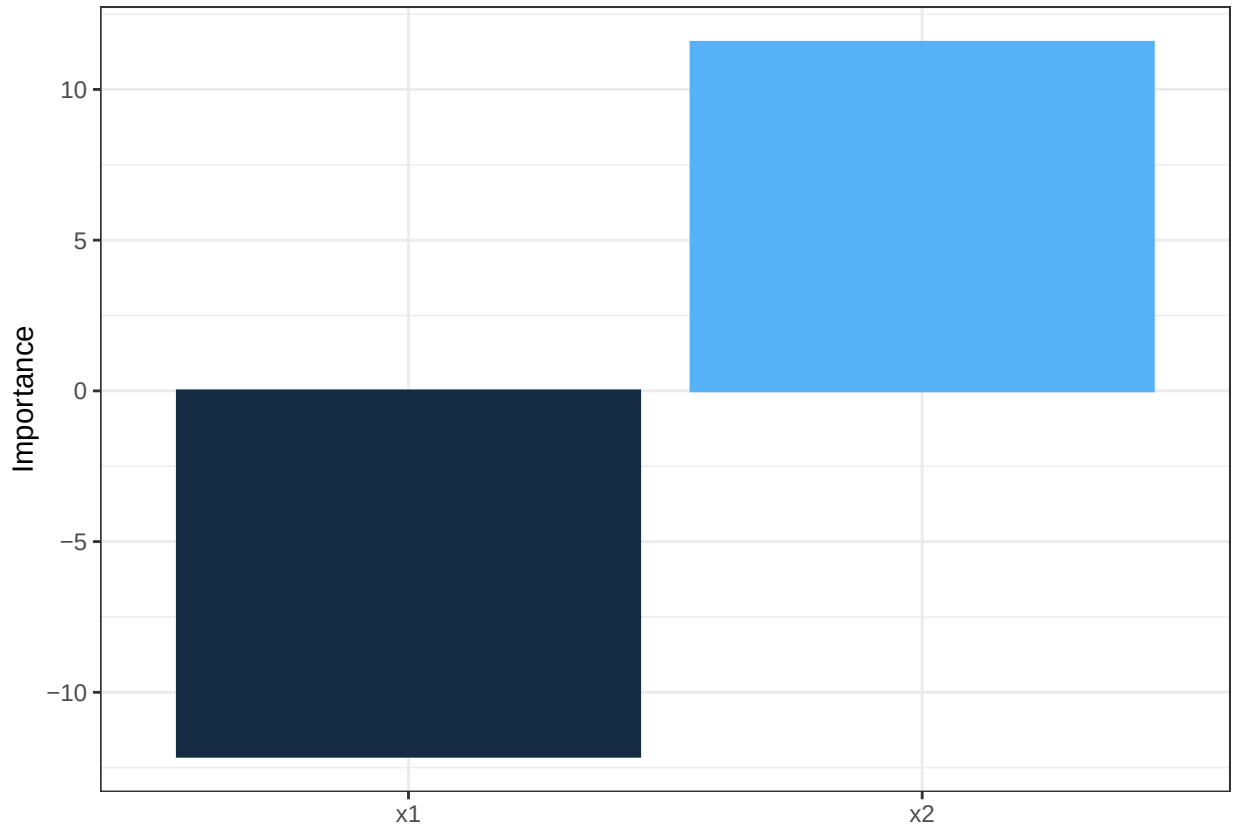
```r
# Explorando las predicciones con ambas redes
par(mfrow=c(1, 2))

plot(x=datis$y, y=yhat1, las=1, xlab="y_t",
     main="Using neuralnet")

plot(x=datis$y, y=yhat2, las=1, xlab="y_t",
     main="Using nnet")
```




```r
# Tarea: Saque una conclusion del ejercicio.

# Para ver la importancia de las variables en la red usamos
library(NeuralNetTools)
garson(mod2) # Garson (1991). Interpreting neural network connection weights
```



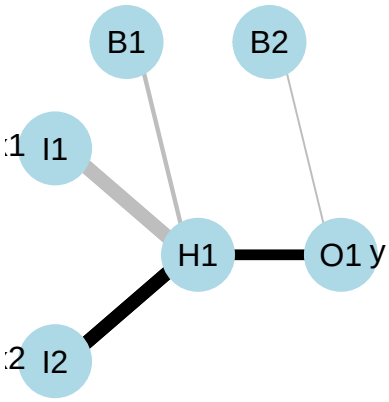

```
olden(mod2)  # Olden et al (2002). Illuminating the 'black-box'
```

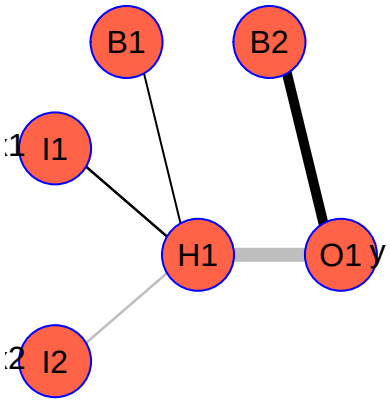



```
# Comparando los MSE ------------------------------------------------------

mse_neuralnet <- mean((datis$y - yhat1)^2)
mse_nn <- mean((datis$y - yhat2)^2)

cbind(mse_neuralnet, mse_nn)
```

```
##      mse_neuralnet     mse_nn
## [1,]    0.01090578 0.01101376
```

```
# Tarea: volver a ajustar mod1 y mod2 pero modificando los otros
# argumentos de las funciones y cambiando la ARQUITECTURA de la red
# para conseguir modelos con ERRORES menores a los mostrados aqui.
# Le apuesto que usted logra disminuir aun mas los MSE.


# Explorando las utilidades del paquete NeuralNetTools --------------------

par(mfrow=c(1, 2))
plotnet(mod1)
plotnet(mod2, circle_col="tomato", bord_col="blue", prune_col="red")
```


B1
B2
I1
H1
O1
y
I2


B1
B2
I1
H1
O1
y
I2