Caso Práctico II

Minería de datos

Estimación y predicción con redes neuronales

José Ignacio Escribano

Móstoles, 12 de diciembre de 2015

Índice de figuras

Índice

1. Introducción

En este caso práctico, pondremos en práctica lo aprendido en teoría sobre redes neuronales, a través de dos librerías distintas de R que las implementan. Éstas son `nnet` y `neuralnet`.

2. Resolución de las cuestiones de evaluación

A continuación resolveremos las cuestiones planteadas en el caso práctico. En todas las cuestiones de evaluación consideraremos la función no lineal dada por $y = x \sin x$ en el intervalo $[-2\pi, 2\pi]$.

2.1. Cuestión 1

Representamos la función $y = x \sin x$ tomando 200 puntos en el intervalo $[-2\pi, 2\pi]$. La función queda como se muestra en la Figura 1.

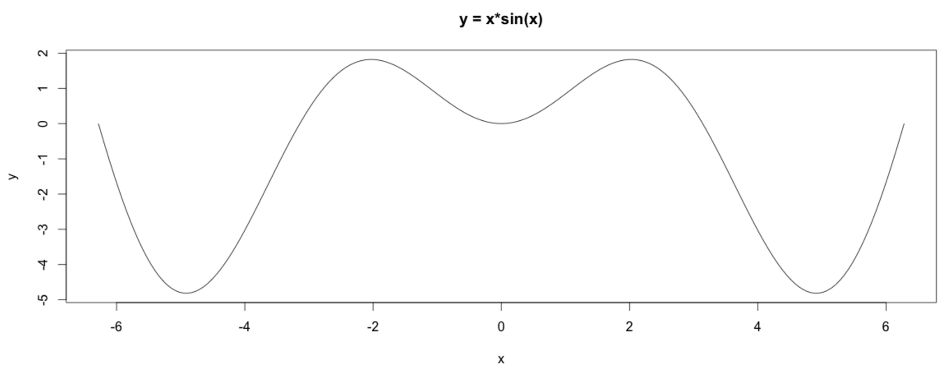


Figura 1: Función $y = x \sin x$ con 200 puntos

Si dibujamos la función sin el parámetro `type='l'`, tenemos que sólo se muestran los puntos $\{(x_i, x_i \sin x_i)\}$ con $i = 1, \ldots, 200$ en el plano. El parámetro `type='l'` lo que hace es interpolar los dintintos puntos para obtener una curva. La Figura 2 muestra este hecho.

Esta función sí podrá ser aproximada por una red neuronal, ya que aunque se trate de una función no lineal, el Teorema de Cybenko nos asegura que una red neuronal multicapa puede aproximar una función continua en subconjuntos compactos de $\mathbb{R}^n$, que es lo que intentamos aproximar.

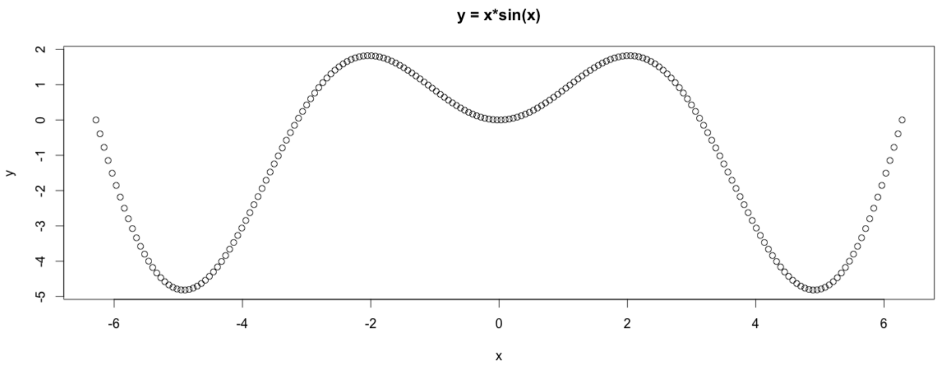


Figura 2: Función $y = x \sin x$ con 200 puntos sin el parámetro type='l'

2.2. Cuestión 2

Ajustamos un modelo lineal con los 200 puntos calculados anteriormente. El resumen que muestra R es el siguiente:

```
Call:
lm(formula = y ~ x)

Residuals:
       Min          1Q     Median          3Q         Max
-3.8188378 -2.2907973  0.9951677  2.0555346  2.8136221

Coefficients:
             Estimate     Std. Error  t value       Pr(>|t|)
(Intercept) -0.9946693  0.16374284208 -6.07458 0.0000000062568 ***
x           -0.0000000  0.04491295114  0.00000               1
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.315673 on 198 degrees of freedom
Multiple R-squared:  6.660172e-32,Adjusted R-squared:  -0.005050505
F-statistic: 1.318714e-29 on 1 and 198 DF,  p-value: 1
```

La recta de regresión es $y = -1$. La Figura 3 muestra la recta de regresión sobre la función. La Figura indica un ajuste bastante malo. Esto lo corroboramos con el coeficiente de determinación, R^2, que es prácticamente 0, por lo que no hay relación lineal entre los valores de x e y.

Comprobemos que se incumplen las hipótesis del modelo: normalidad en los residuos y/o linealidad. Para ello, representamos en R, un Q-Q plot y un gráfico de residuos

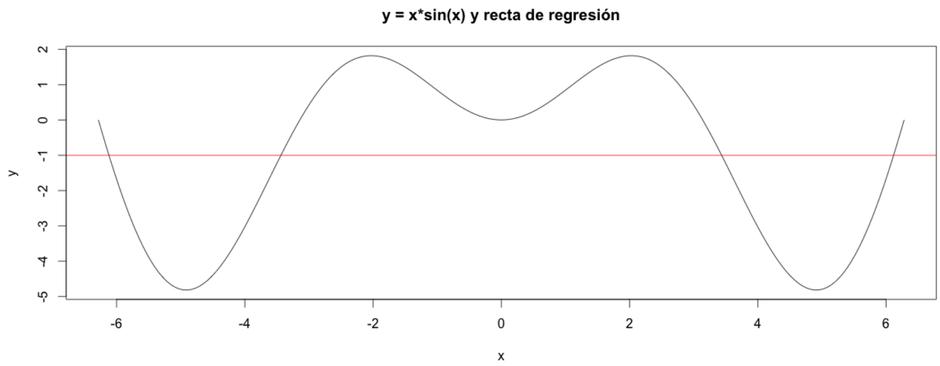
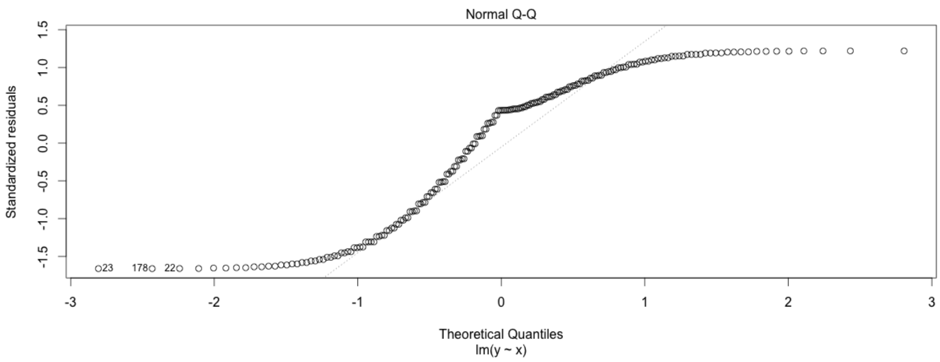
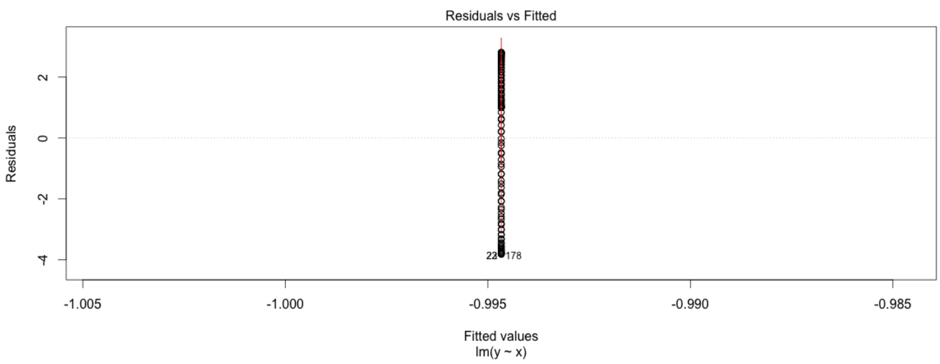


Figura 3: Recta de regresión de la función $y = x \sin x$

(Figura 4).



(a) Q-Q plot



(b) Residuos

Figura 4: Q-Q plot y gráfico de residuos del modelo lineal

Observamos que los residuos no están normalmente distribuidos, por lo que este modelo no es válido. No hay relación lineal entre los valores de x e y.

2.3. Cuestión 3

Procedemos a ajustar la función con una red neuronal. Representamos la función original junto con la función de ajuste de la red neuronal desde 1 hasta 10 neuronas (Figura 5).

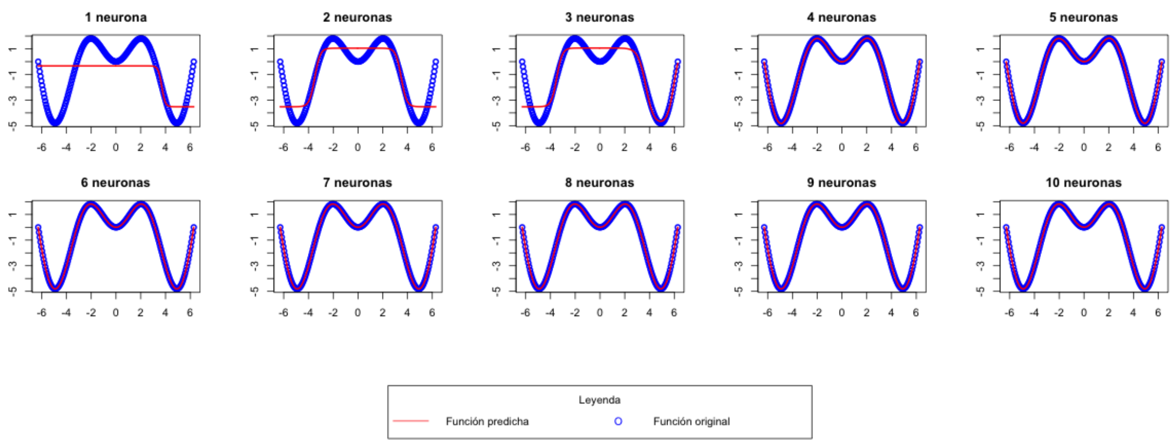


Figura 5: Predicción con distintas neuronas

Se observa que, a medida que aumentamos el número de neuronas, el ajuste de la función aprendida cada vez es mejor, aunque a partir de la cuarta neurona, apenas hay diferencias entre entre la función predicha y la original. Por tanto, con 4 neuronas es suficiente para predecir correctamente esta función.

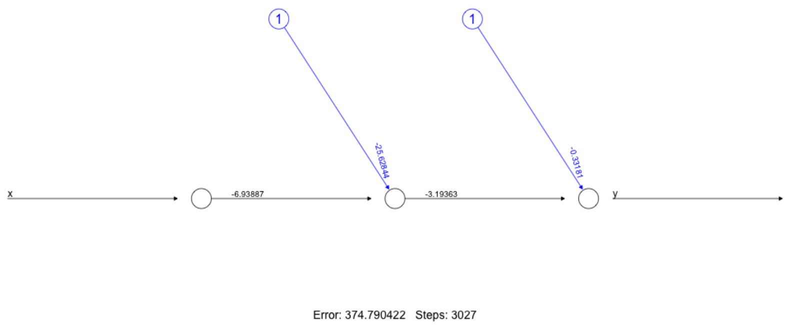
A modo de ejemplo, vemos la salida que produce R para la red con 4 neuronas.

```
# weights:  13
initial  value 1300.296952
iter  10 value 238.577267
iter  20 value 165.124118
iter  30 value 145.364713
iter  40 value 140.272616
iter  50 value 129.336562
iter  60 value 108.238405
iter  70 value 98.038065
iter  80 value 73.135425
iter  90 value 53.622029
...
```

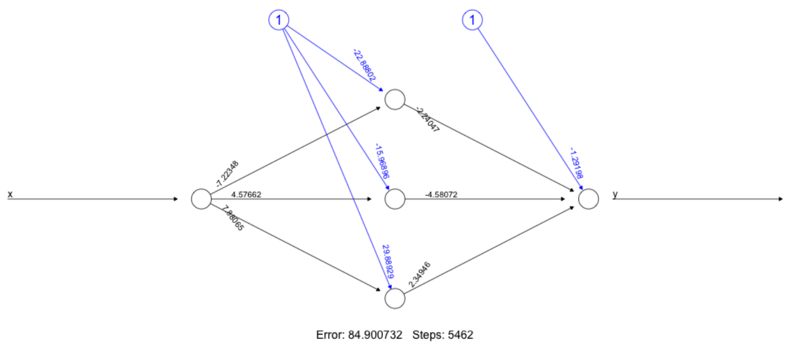
```
iter1200 value 0.336774
iter1210 value 0.335452
iter1220 value 0.327588
iter1230 value 0.326635
iter1240 value 0.318003
final  value 0.317111
converged
```

Es decir, converge tras 1240 iteraciones del algoritmo con un error de 0.317111.

Por último, representamos la red para 1 y 3 neuronas. Éstas se pueden ver en la Figura 6.



(a) 1 neurona



(b) 3 neuronas

Figura 6: Representación de la red para 1 y 3 neuronas

Observamos que, en el caso de una sola neurona tenemos un error de 374.79 tras 3027 iteraciones del algoritmo. En el caso de tres neuronas, tenemos un error de 54.90 tras 5462 iteraciones del algoritmo. De nuevo, vemos que aumentando el número de neuronas de la capa oculta, reducimos el error del ajuste de la función. Notar que se obtienen errores sensiblemente diferentes según se emplee una librería u otra. Esto puede ser debido a la implementación o la inicialización de los pesos del algoritmo utilizado.

3. Conclusiones

En este caso práctico, hemos puesto en práctica las redes neuronales para aprender y predecir valores de una función. También hemos visto que es indiferente que la función sea o no lineal, puesto que una red neuronal multicapa será capaz de aprenderla gracias al teorema de Cybenko (sobre un subconjunto compacto de $\mathbb{R}^n$).

En R existen distintas librerías que implementan redes neuronales, por lo que hay que tener cuidado a la hora de aplicar una librería u otra ya que pueden variar sensiblemente los resultados según una u otra. Pero conociendo las ventajas de cada librería podremos conseguir grandes resultados gracias a la gran capacidad de cálculo de R.

4. Código R utilizado

El código utilizado para la resolución de este caso práctico, separado por cuestiones, se puede ver a continuación:

```
#--------------------------------------------------------------
# Cuestiones de evaluación
#--------------------------------------------------------------

#--------------------------------------------------------------
# Cuestión 1
#--------------------------------------------------------------

# Dibujamos la función y = x*sen(x) en [-2π, 2π]
# con 200 puntos
points = 200
x <- seq(-2*pi, 2*pi, length = points)
y <- x*sin(x)
plot(x,y, type='l', main = 'y = x*sin(x)')

#--------------------------------------------------------------
# Cuestión 2
#--------------------------------------------------------------

# Ajustamos los datos según un modelo linear
linear.model <- lm(y~x)
summary(linear.model)

# Representamos los datos junto a la recta de regresión
plot(x,y, type='l', main = 'y = x*sin(x) y recta de regresión')
abline(h=-1, col=2)

# Comprobamos las hipótesis del modelo
plot(linear.model,1)
plot(linear.model,2)

#--------------------------------------------------------------
# Cuestión 3
#--------------------------------------------------------------

# Cargamos la librería
library(nnet)
```

```
# Definimos función para ejecutar varias veces (hasta un
# máximo de 10) el comando nnet y devuelve el ajuste  que
# tenga el error menor tenga
min.nn <- function(x,y, iter=10, neurals=5){
  min_value <- 10e6 # Infinito
  actual_fit <- NULL
  for(i in seq(1, iter)){
    fit.nn <- nnet(x,y, rang=0.1, size=neurals, linout=T,
        maxit=10000)
    cat(fit.nn$value, "\n")
    if(fit.nn$value < min_value){
      min_value <- fit.nn$value
      actual_fit <- fit.nn
    }
  }
  return(actual_fit)
}

# Comparamos la función calculada con la función original
# según el número de neuronas en la capa oculta
# (desde 1 hasta 10)
nf<-layout(matrix(c(1,2,3,4,5,6,7,8,9,10,11,11,11,11,11),
    ncol=5, byrow=TRUE))
par(mai= rep(0.5, 4))
predictions <- vector()
for(neurals in seq(1,10)){

  # Curva ajustada con el menor error entre varias ejecuciones
  # de la función nnet
  min <- min.nn(x, y, 10, neurals)

  # Predicción de los valores a partir de la red calculada
  predict.nn<-predict(min, as.matrix(x))

  # Representamos la función predicha junto con la función
  # original
  if(neurals == 1)
    neurals_plot <-  "neurona"
  else
    neurals_plot <- "neuronas"

  plot(x,y, col=4, lwd=2, main=paste(neurals, neurals_plot))
  points(x, predict.nn, type='l', col=2, lwd=2)
```

```
}

par(mai=c(0,0,0,0))
plot.new()
legend(x="center", ncol=2,
    legend=c("Función predicha", "Función original"),
    title="Leyenda",
    lty = c(1, NA),
    pch = c(NA, 'O'),
    col=c('red', 'blue'))

# Cargamos la librería neuralnet
library(neuralnet)

# Representamos la red con 1 y 3 neuronas

par(mfrow=c(1,2))
for(neurals in c(1,3)){
  curve.nn <- neuralnet(y~x, cbind(x,y), hidden=neurals,
      rep=1, stepmax=10e6)
  plot.nn(curve.nn)
}
```