Data Science con R

Instituto de Estadística PUCV - Magister en Estadística

Dae-Jin Lee < dlee@bcamath.org >

```
install.packages("matrixStats")
install.packages("neuralnet")
install.packages("nnet")
install.packages("NeuralNetTools")
install.packages("quantmod")
install.packages("boot")
install.packages("plyr")
```

Introducción a las redes neuronales artificiales

En esta sección describimos una clase de métodos de aprendizaje que se desarrollaron por separado en diferentes campos: estadística e inteligencia artificial basadas en modelos esencialmente idénticos.

Las redes neuronales artificiales se inspiran en el comportamiento conocido del cerebro humano (principalmente el referido a las neuronas y sus conexiones), trata de crear modelos artificiales que solucionen problemas difíciles de resolver mediante técnicas algorítmicas convencionales.

La idea central es extraer las combinaciones lineales de las variables de entrada como características derivadas y, a continuación, modelar el objetivo como una función no lineal de estas características. El resultado es un poderoso método de aprendizaje, con amplias aplicaciones en muchos campos.

La neurona artificial pretende mimetizar las características más importantes de la neurona biólogica. En general, recibe las señales de entrada de las neuronas vecinas ponderadas por los pesos de las conexiones. La suma de estas señales ponderadas proporciona la entrada total o neta de la neurona y, mediante la aplicación de una función matemática - denominada función de salida - , sobre la entrada neta, se calcula un valor de salida, el cual es enviado a otras neuronas.

Tanto los valores de entrada a la neurona como su salida pueden ser señales excitatorias (cuando el valor es positivo) o inhibitorias (cuando el valor es negativo).

Si queremos restringir los valores de salida utilizamos una **función activación**.

La función de activación contrarresta el valor de salida y produce un valor dentro de un rango (que se basa en el tipo de función de activación).

Una red neuronal es un conjunto de capas (una capa tiene un conjunto de neuronas) apiladas secuencialmente.

Aquí tenemos tres capas

- **Input layer/Capa de entrada:** Un conjunto de neuronas de entrada donde cada neurona representa cada característica de nuestro conjunto de datos. Toma las entradas y las pasa a la siguiente capa.
- **Hidden layer/Capa oculta:** Un conjunto de (n) nº de neuronas donde cada neurona tiene un peso (parámetro) asignado a ella. Toma la entrada de la capa anterior y hace el producto del punto de entradas y pesos, aplica la función de la activación (como hemos visto arriba), produce el resultado y pasa los datos a la capa siguiente.

Nota: Podemos tener (n) nº de capas ocultas en el medio. .

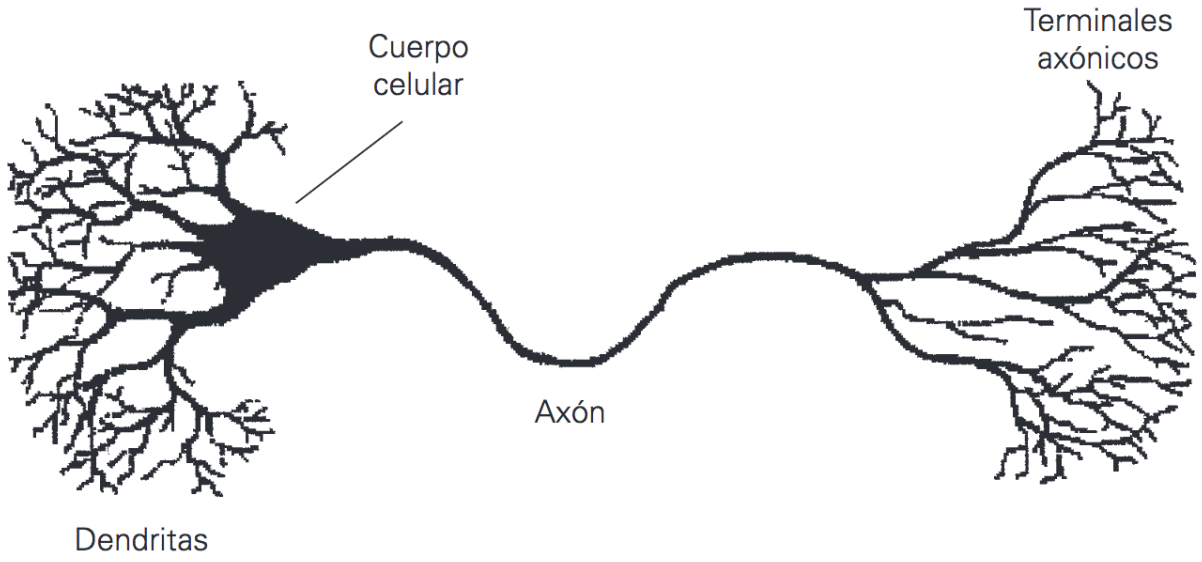


Figure 1: *Estructura general de una neurona biológica.*

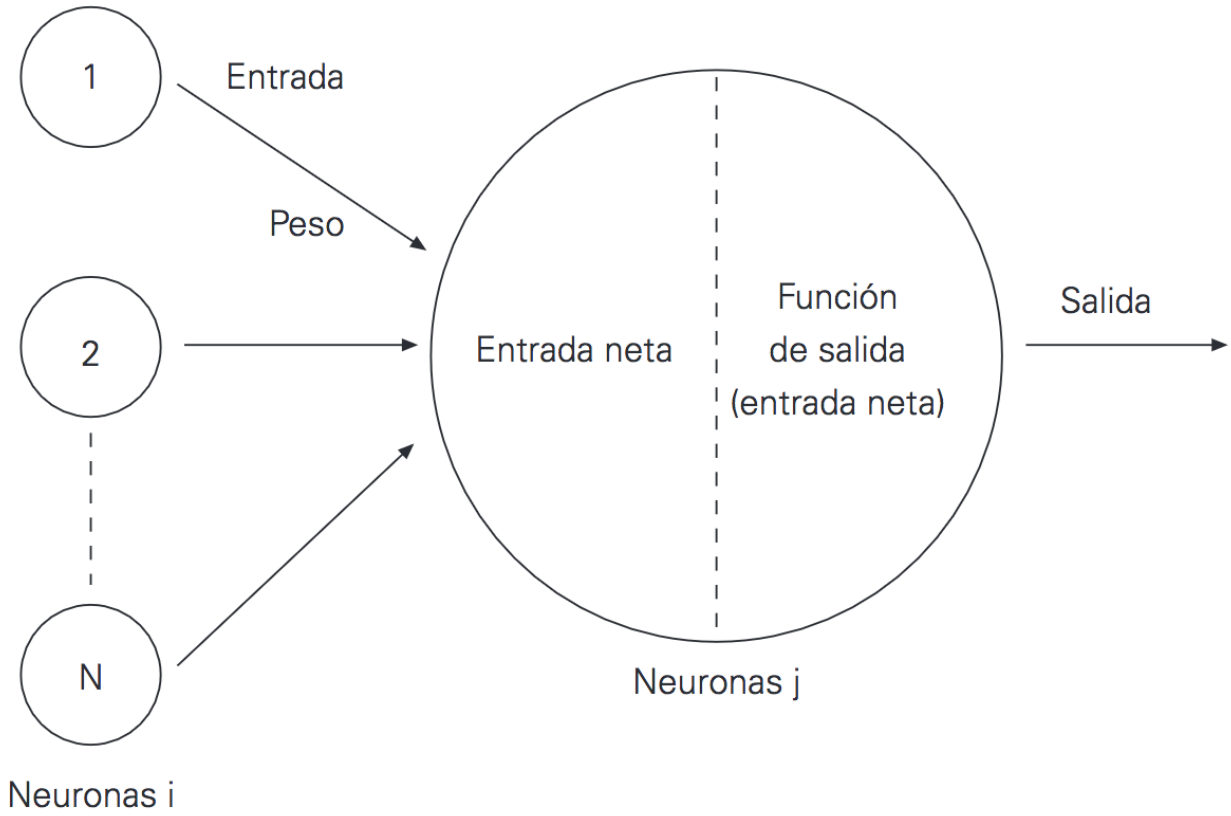


Figure 2: *Funcionamiento general de una neurona artificial.*

	Propagation
Sigmoid	$y_s = \frac{1}{1+e^{-x_s}}$
Tanh	$y_s = \tanh(x_s)$
ReLu	$y_s = \max(0, x_s)$

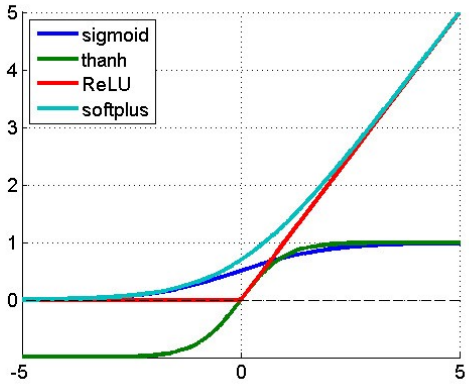
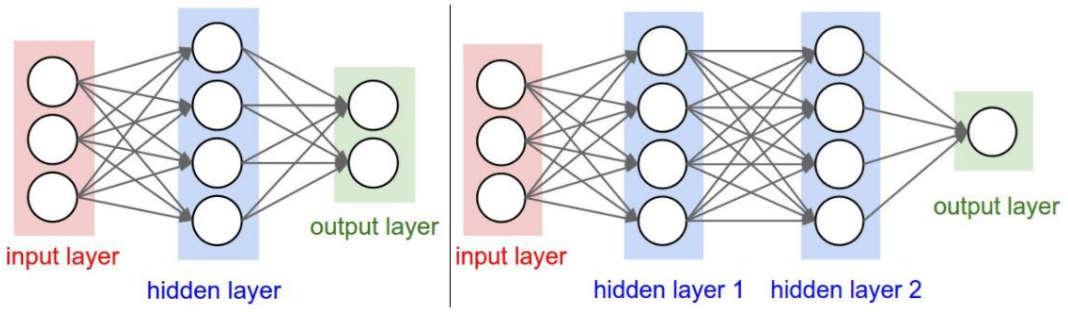


Figure 3: *Tipos de funciones de activación.*



Left: A 2-layer Neural Network (one hidden layer of 4 neurons (or units) and one output layer with 2 neurons), and three inputs.
Right: A 3-layer neural network with three inputs, two hidden layers of 4 neurons each and one output layer. Notice that in both cases there are connections (synapses) between neurons across layers, but not within a layer.

Figure 4: *Capas.*

- **Output layer/Capa de salida:** es la misma capa oculta excepto que da el resultado final (resultado/clase/valor).

¿Cómo definimos el número de neuronas en cada capa y toda la red?

Las neuronas de la capa de entrada se basan en el número de características del conjunto de datos.

Podemos definir tantas neuronas/capas como queramos (depende de los datos y el problema) pero sería bueno definir más que características y todas las capas ocultas tienen el mismo número de neuronas. Las neuronas de la capa de salida se basan en el tipo de problema y los resultados.

En el caso de un problema de regresión, entonces tendremos 1 neurona, para la clasificación binaria podemos tener 1 o 2 neuronas. y para la multiclasificación más de 2 neuronas.

El algoritmo de una red neuronal se puede resumir del siguiente modo:

1. Elegir la arquitectura de red (inicializar con pesos aleatorios)
2. Realizar un pase adelante (Forward propagation/Propagación adelante)
3. Calcular el error total (necesitamos minimizar este error)
4. Volver a propagar el error y Actualizar pesos (Backwar propagation/Propagación posterior)
5. Repita el proceso (2-4) para un número de iteraciones o hasta que el error sea mínimo.

Ejemplos de aplicación de las RNA

Servicios Bancarios y Financieros:

- **Medición del riesgo crediticio de los nuevos solicitantes:** El riesgo de crédito es el riesgo de incumplimiento por parte de los clientes a los que se aprueba una línea de crédito. Los bancos y las instituciones financieras construyen un modelo predictivo para evaluar la solvencia crediticia de los nuevos solicitantes. Se utilizan varias técnicas para construir el *scorecard* de riesgo crediticio o modelo predictivo. La red neuronal también se utiliza para clasificar a los clientes en buenos y malos. En este escenario, la variable objetivo es binaria y el tipo de problema es la clasificación.
  - **Predicción de la tasa de recuperación de préstamos:** Incluso después de usar el scorecard de crédito avanzado, hay un valor predeterminado. Una vez que los clientes han incumplido el pago de sus préstamos, las instituciones financieras intentan recuperar la mayor cantidad de dinero posible (mediante el cumplimiento de las normas legales y reglamentarias) de los clientes incumplidos. La tasa de recuperación ("complemento de la pérdida en caso de incumplimiento") es una fracción de los saldos pendientes recuperados de los clientes morosos. El modelo no paramétrico basado en redes neuronales se utiliza para pronosticar recuperaciones bancarias.
  - **Reserva para siniestros de seguros:** En el sector de los seguros, la estimación de las necesidades futuras de reservas de efectivo es crucial para optimizar las operaciones. El requerimiento de reserva de efectivo depende de los volúmenes futuros de reclamaciones y de la cantidad de reclamaciones. La Red Neural Artificial (ANN) podría ser usada para pronosticar reclamos futuros.

También se utiliza para la segmentación de tiendas minoristas, la previsión del volumen total de ventas y la predicción de los encuestados para el correo de fidelización.

Arquitecturas de las RNA

Las neuronas que componen una RNA se organizan de forma jerárquica formando capas. Una capa o nivel es un conjunto de neuronas cuyas entradas de información provienen de la misma fuente (que puede ser otra capa de neuronas) y cuyas salidas de información se dirigen al mismo destino (que puede ser otra capa de neuronas). En este sentido, se distinguen tres tipos de capas: la capa de entrada recibe la información del exterior; la o las capas ocultas son aquellas cuyas entradas y salidas se encuentran dentro del sistema y, por tanto, no tienen contacto con el exterior; por último, la capa de salida envía la respuesta de la red al exterior.

En función de la organización de las neuronas en la red formando capas o agrupaciones podemos encontrarnos con dos tipos de arquitecturas básicas: redes multicapa (multi-layer) y redes monocapa (single-layer).

Datos Boston housing

Vamos a utilizar el conjunto de datos de `Boston` en el paquete `MASS`. Recordemos que el conjunto de datos de `Boston` es una colección de datos sobre valores de vivienda en los suburbios de Boston. Nuestro objetivo es predecir el valor medio de las viviendas ocupadas por sus propietarios (‘medv) utilizando todas las demás variables continuas disponibles.

```
set.seed(500)
library(MASS)
data <- Boston
```

Primero tenemos que comprobar que no hay datos faltantes, de lo contrario tenemos que arreglar el conjunto de datos.

```
apply(data,2,function(x) sum(is.na(x)))
```

```
##    crim      zn   indus    chas     nox      rm     age     dis     rad
##       0       0       0       0       0       0       0       0       0
##     tax ptratio   black   lstat    medv
##       0       0       0       0       0
```

Hemos comprobado que no hay datos faltantes (`NA`). Procedemos dividiendo aleatoriamente los datos en un conjunto de entrenamiento y un conjunto de prueba, luego ajustamos un modelo de regresión lineal y lo probamos en el conjunto de prueba. Nótese que usamos la función `glm()` en lugar de la función `lm()` esto será útil más tarde al validar el modelo lineal.

```
index <- sample(1:nrow(data),round(0.75*nrow(data)))
train <- data[index,]
test <- data[-index,]
lm.fit <- glm(medv~., data=train)
summary(lm.fit)
```

```
##
## Call:
## glm(formula = medv ~ ., data = train)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -14.9143   -2.8607   -0.5244    1.5242   25.0004
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  43.469681   6.099347   7.127 5.50e-12 ***
## crim         -0.105439   0.057095  -1.847 0.065596 .
## zn            0.044347   0.015974   2.776 0.005782 **
## indus         0.024034   0.071107   0.338 0.735556
## chas          2.596028   1.089369   2.383 0.017679 *
## nox         -22.336623   4.572254  -4.885 1.55e-06 ***
## rm            3.538957   0.472374   7.492 5.15e-13 ***
## age           0.016976   0.015088   1.125 0.261291
## dis          -1.570970   0.235280  -6.677 9.07e-11 ***
## rad           0.400502   0.085475   4.686 3.94e-06 ***
## tax          -0.015165   0.004599  -3.297 0.001072 **
```

```
## ptratio      -1.147046   0.155702  -7.367 1.17e-12 ***
## black         0.010338   0.003077   3.360 0.000862 ***
## lstat        -0.524957   0.056899  -9.226  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 23.26491)
##
##     Null deviance: 33642  on 379  degrees of freedom
## Residual deviance:  8515  on 366  degrees of freedom
## AIC: 2290
##
## Number of Fisher Scoring iterations: 2
```

```
pr.lm <- predict(lm.fit,test)
MSE.lm <- sum((pr.lm - test$medv)^2)/nrow(test)
```

La función `sample(x,size)` simplemente produce un vector del tamaño especificado de muestras seleccionadas aleatoriamente desde el vector x. Por defecto el muestreo es sin reemplazo: el índice es esencialmente un vector aleatorio de muestras.

Como se trata de un problema de regresión, vamos a utilizar el error cuadrático medio (ECM) como medida de lo lejos que están nuestras predicciones de los datos reales.

Ajuste de la red neuronal

Antes de entrenar una red neural, es necesario hacer algún paso previo. Como primer paso, vamos a abordar el preprocesamiento de datos.

- Es una buena práctica normalizar los datos antes de entrenar una red neuronal.
- Se pueden elegir diferentes métodos para escalar los datos (normalización-z, escala min-max, etc. . .).
- Elegiremos el método min-max y escalaremos los datos en el intervalo $[0, 1]$. Por lo general, la escala en los intervalos $[0, 1]$ ó $[-1, 1]$ tiende a dar mejores resultados.
- Por lo tanto, escalamos y dividimos los datos antes de seguir adelante:

```
maxs <- apply(data, 2, max)
mins <- apply(data, 2, min)

scaled <- as.data.frame(scale(data, center = mins, scale = maxs - mins))

train_ <- scaled[index,]
 test_ <- scaled[-index,]
```

No hay una regla fija en cuanto a cuántas capas y neuronas usar, aunque hay varias reglas generales más o menos aceptadas. Normalmente, si es necesario, una capa oculta es suficiente para un gran número de aplicaciones.

En cuanto al número de neuronas, debería estar entre el tamaño de la capa de entrada y el tamaño de la capa de salida, normalmente 2/3 del tamaño de entrada. No hay garantía de que ninguna de estas reglas se ajuste mejor a su modelo de modo que es recomendable probar con diferentes combinaciones.

Para este ejemplo, vamos a usar 2 capas ocultas con esta configuración: `13:5:3:1`. + La capa de entrada tiene 13 entradas,

- las dos capas ocultas tienen 5 y 3 neuronas y

- la capa de salida tiene, por supuesto, una sola salida ya que estamos haciendo regresión.

Vamos a entrar en la red:

```
library(neuralnet)
n <- names(train_)
f <- as.formula(paste("medv ~", paste(n[!n %in% "medv"], collapse = " + ")))
nn <- neuralnet(f,data=train_,hidden=c(5,3),linear.output=TRUE)
```

Nota:

- La expresión `y~.` no es aceptada por la función `neuralnet()`.
- El argumento oculto acepta un vector con el número de neuronas para cada capa oculta, mientras que el argumento `linear.output` se usa para especificar si queremos hacer una regresión `linear.output=TRUE` o clasificación `linear.output=FALSE`.

La librería `neuralnet` proporciona una buena herramienta para representar gráficamente el modelo con los pesos en cada conexión:

```
plot(nn)
```

Las líneas negras muestran las conexiones entre cada capa y los pesos de cada conexión, mientras que las líneas azules muestran el término de sesgo añadido en cada paso. El sesgo puede ser pensado como el intercepto de un modelo lineal.

La red neuronal es esencialmente una caja negra, por lo que no podemos decir mucho sobre el ajuste, los pesos y el modelo. Basta decir que el algoritmo de entrenamiento ha convergido y por lo tanto el modelo está listo para ser utilizado.

Predicción con una red neuronal

Ahora podemos tratar de predecir los valores para el equipo de prueba y calcular el ECM. Recuerda que la red producirá una predicción normalizada, por lo que necesitamos reducirla para hacer una comparación significativa (o simplemente una predicción simple).

```
pr.nn <- compute(nn,test_[,1:13])

pr.nn_ <- pr.nn$net.result*(max(data$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_)
MSE.nn
```

```
## [1] 15.75183702
```

Podemos ahora comparar los dos ECM:

```
print(paste(MSE.lm,MSE.nn))
```

```
## [1] "21.6297593507225 15.7518370200153"
```

Aparentemente la red neuronal está haciendo un mejor trabajo que el modelo lineal en la predicción de `medv`. Una vez más, hay que tener cuidado porque este resultado depende de la división de prueba de entrenamiento realizada anteriormente.

A continuación, vamos a realizar una rápida validación cruzada para tener más confianza en los resultados.

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

plot(test$medv,pr.nn_,col='red',main='Real vs predicted NN',pch=18,cex=0.7)
```

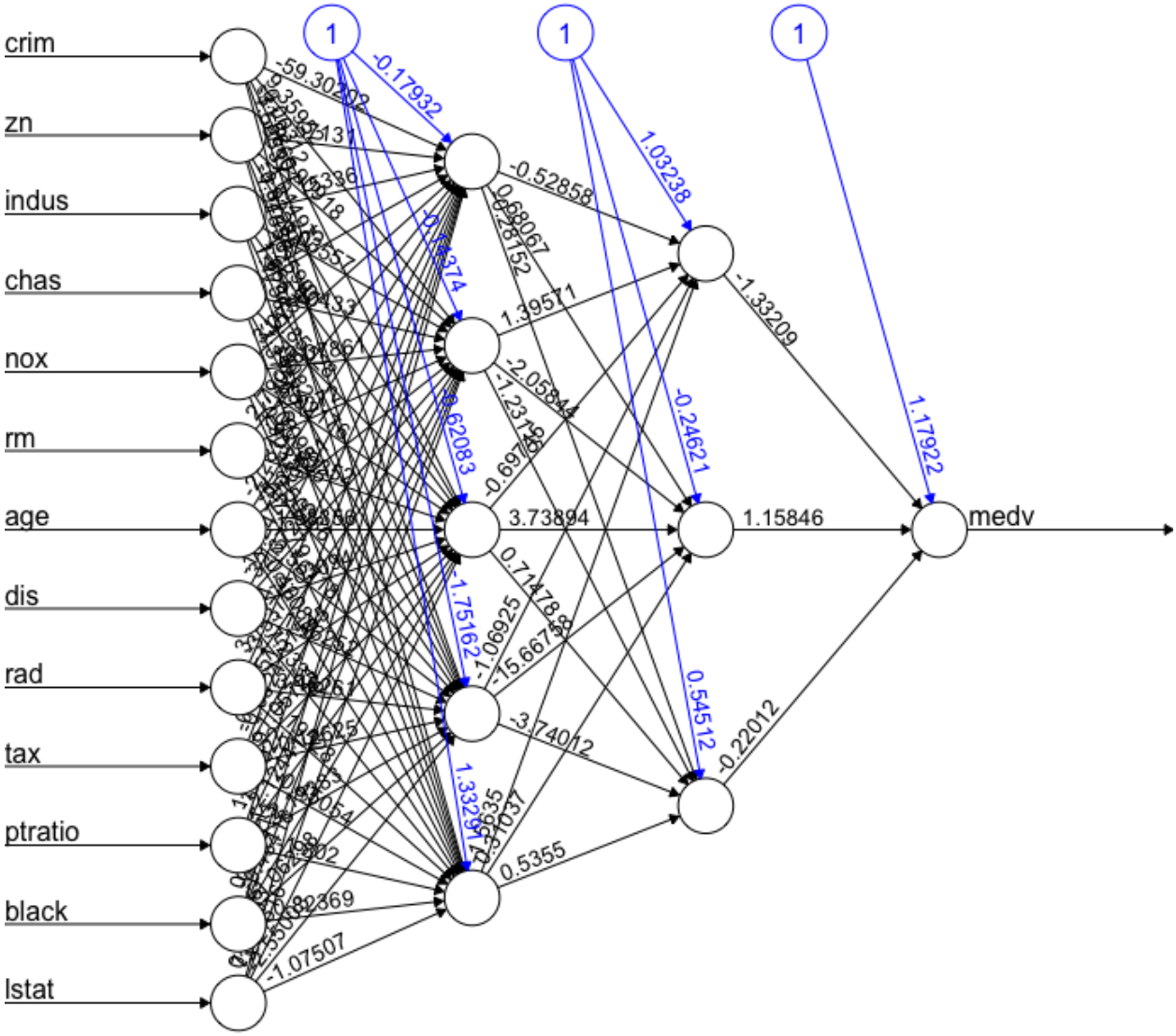
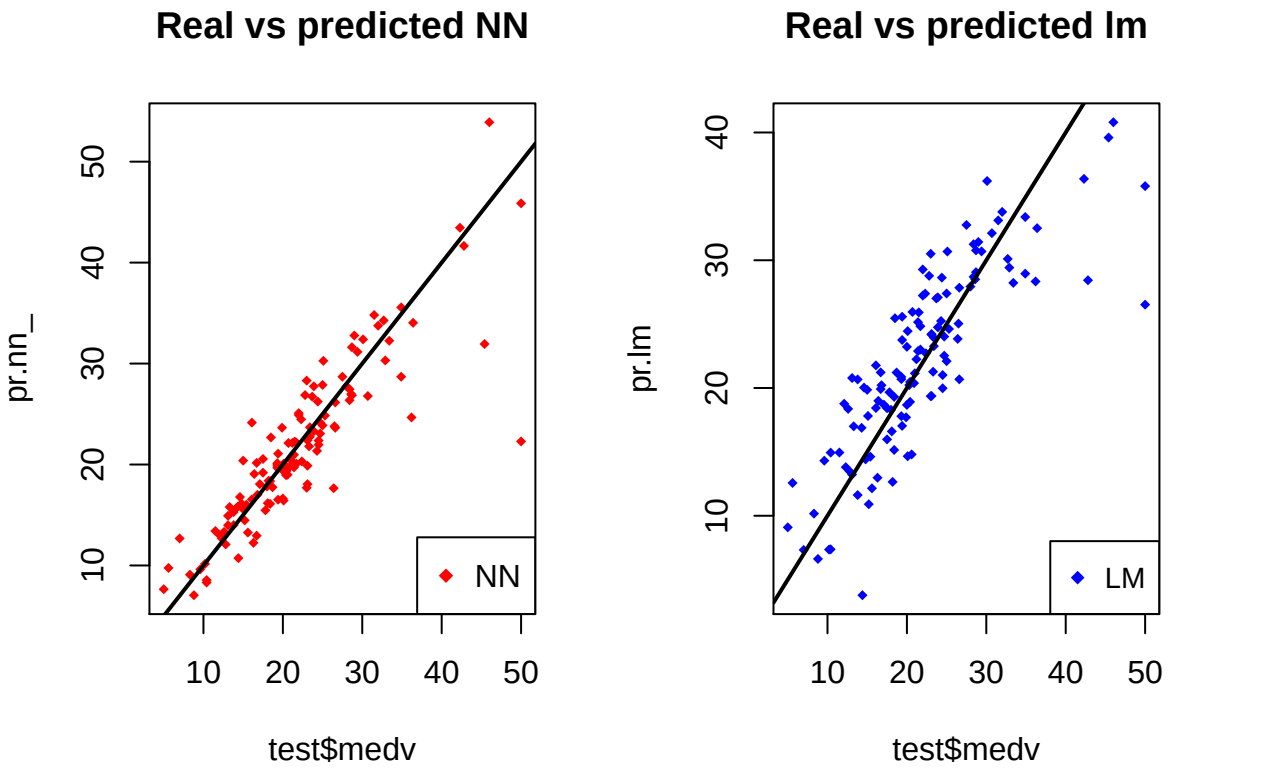

crim
zn
indus
chas
nox
rm
age
dis
rad
tax
ptratio
black
lstat
1
1
1
-0.17932
1.03238
-0.52858
-1.33209
1.17922
1.39571
-0.24621
-0.62083
3.73894
1.15846
medv
0.54512
-0.22012
-3.74012
0.5355
-1.07507

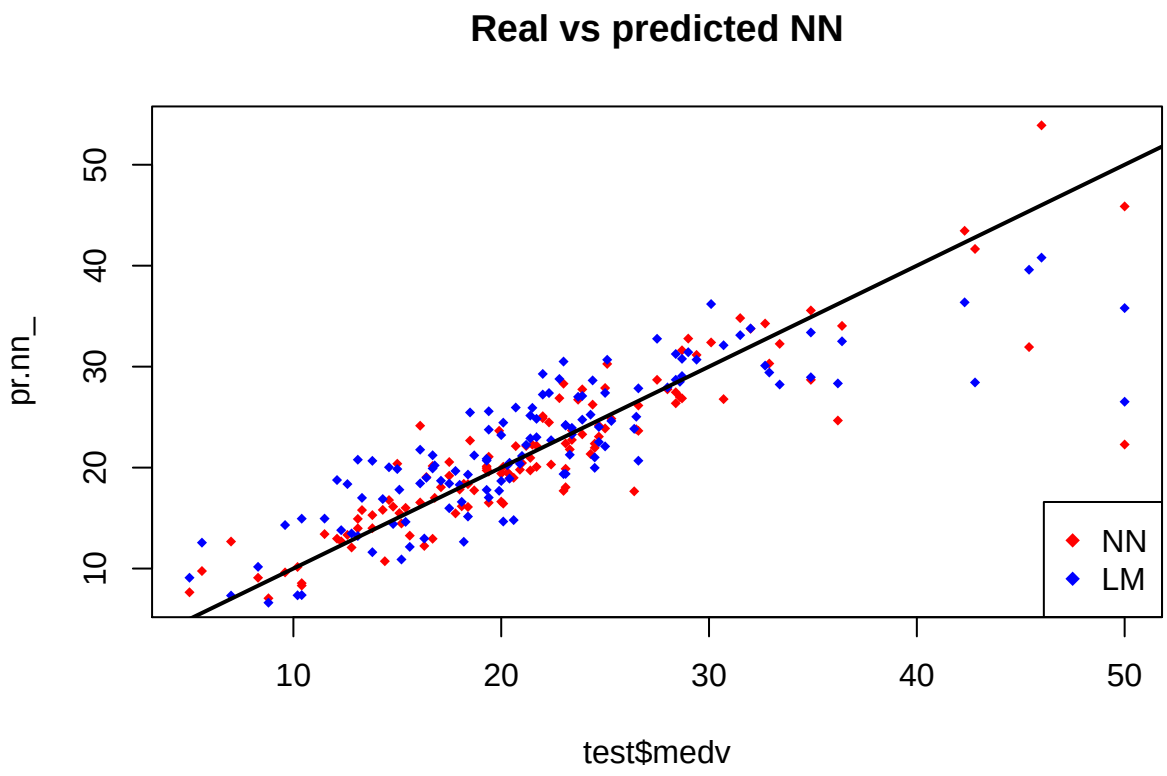
Figure 5:

```
abline(0,1,lwd=2)
legend('bottomright',legend='NN',pch=18,col='red')

plot(test$medv,pr.lm,col='blue',main='Real vs predicted lm',pch=18, cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend='LM',pch=18,col='blue', cex=.95)
```



Inspeccionando visualmente podemos ver que las predicciones hechas por la red neural están (en general) más concentradas alrededor de la línea (una alineación perfecta con la línea indicaría un ECM de 0 y por lo tanto una predicción perfecta ideal) que las hechas por el modelo lineal.

```
plot(test$medv,pr.nn_,col='red',main='Real vs predicted NN',pch=18,cex=0.7)
points(test$medv,pr.lm,col='blue',pch=18,cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend=c('NN','LM'),pch=18,col=c('red','blue'))
```



Evaluando la predicción mediante validación cruzada

La validación cruzada es otro paso muy importante en la construcción de modelos predictivos. Aunque existen diferentes tipos de métodos de validación cruzada, la idea básica es repetir el proceso siguiente varias veces:

División Entrenamiento-Prueba:

- Realizar la división entre muestra de entrenamiento-prueba.
- Ajustar el modelos al conjunto de entrenamiento.
- Comprobar el modelo en el conjunto de prueba.
- Calcular el error de predicción.
- Repetir el proceso K veces.

Luego, calculando el error promedio, podemos hacernos una idea de cómo le está yendo al modelo.

Vamos a implementar una validación cruzada usando un bucle para la red neuronal y la función `cv.glm()` en el paquete `boot` para el modelo lineal. Para $K = 10$.

```
library(boot)
set.seed(200)
lm.fit <- glm(medv~.,data=data)
cv.glm(data,lm.fit,K=10)$delta[1]
```

```
## [1] 23.83560156
```

```
set.seed(450)
cv.error <- NULL
k <- 10


for(i in 1:k){
```

```
    index <- sample(1:nrow(data),round(0.9*nrow(data)))
    train.cv <- scaled[index,]
    test.cv <- scaled[-index,]

    nn <- neuralnet(f,data=train.cv,hidden=c(5,2),linear.output=T)

    pr.nn <- compute(nn,test.cv[,1:13])
    pr.nn <- pr.nn$net.result*(max(data$medv)-min(data$medv))+min(data$medv)

    test.cv.r <- (test.cv$medv)*(max(data$medv)-min(data$medv))+min(data$medv)

    cv.error[i] <- sum((test.cv.r - pr.nn)^2)/nrow(test.cv)

}
```

Calculamos el ECM promedio y graficamos los resultados como una gráfica de caja.

```
mean(cv.error)
```

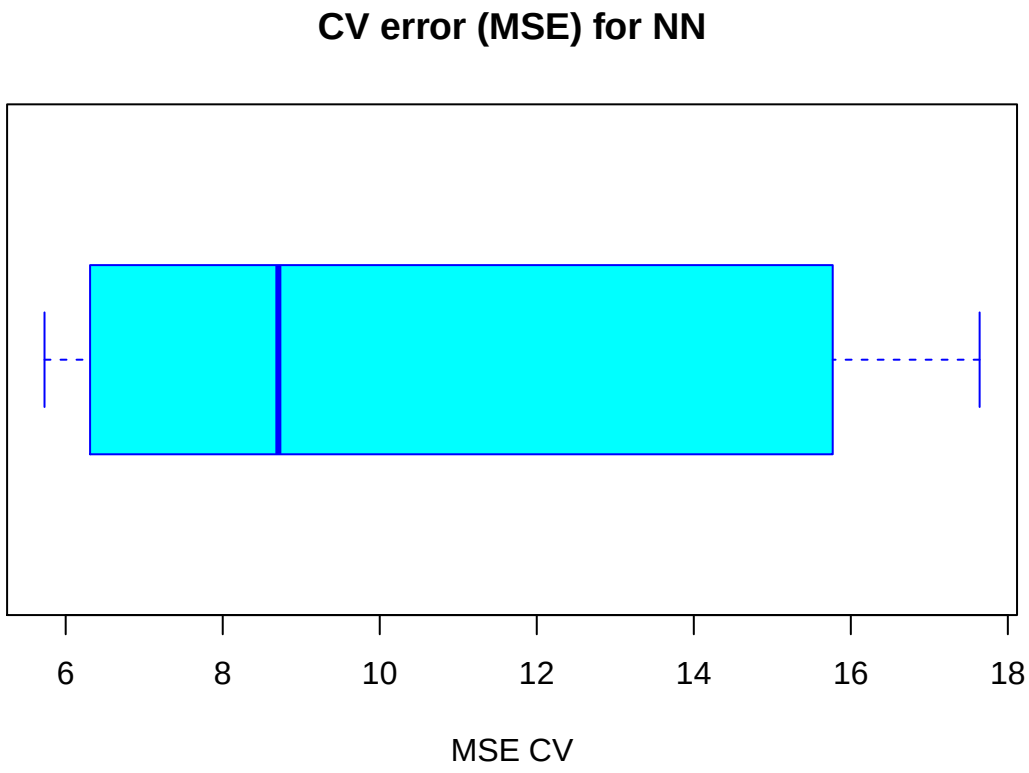
```
## [1] 10.32697995
```

```
cv.error
```

```
##  [1] 17.640652805  6.310575067 15.769518577  5.730130820 10.520947119
##  [6]  6.121160840  6.389967211  8.004786424 17.369282494  9.412778105
```

Boxplot

```
boxplot(cv.error,xlab='MSE CV',col='cyan',
        border='blue',names='CV error (MSE)',
        main='CV error (MSE) for NN',horizontal=TRUE)
```



Como se puede ver, la media del ECM de la red neural (10.33) es inferior al del modelo lineal, aunque parece

haber cierto grado de variación en laos ECMs de la validación cruzada. Esto puede depender de la división de los datos o de la inicialización aleatoria de los pesos en la red neuronal. Al ejecutar la simulación en diferentes momentos con diferentes semillas, puede obtener una estimación puntual más precisa para el ECM medio.

Comparación de modelos RNA

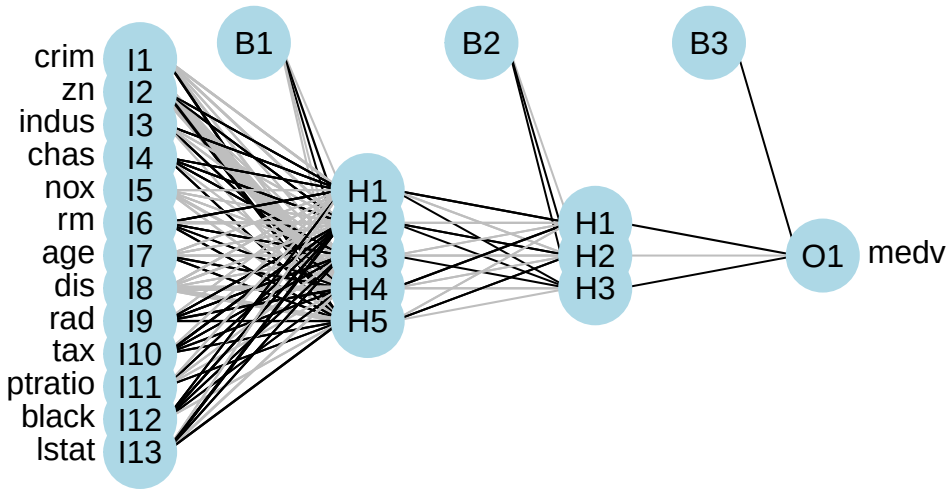
A continuación vamos a comparar 2 modelos:

1. 2 capas ocultas de 5 y 3.
2. 1 capa oculta de 8

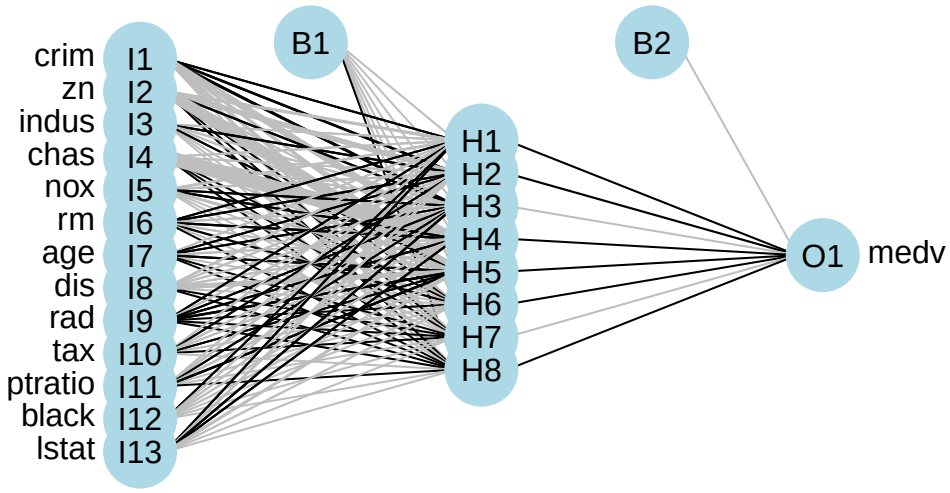
```
Boston.nn.5.3 <- neuralnet(f
                , data=train_
                , hidden=c(5,3)
                , linear.output=TRUE)

Boston.nn.8 <- neuralnet(f
                , data=train_
                , hidden=8
                , linear.output=TRUE)
```

Como alternativa a la función `plot.nn()`, la librería `NeuralNetTools` incluye funciones gráficas más elegantes. Este elegante gráfico resuelve el problema del desorden visual utilizando el grosor de la línea para representar la magnitud del peso y el color de la línea para representar el signo de peso (negro = positivo, gris = negativo).

```
library(NeuralNetTools)
plotnet(Boston.nn.5.3)
```




```
plotnet(Boston.nn.8)
```



Datos cereales

Los datos contienen información sobre varias variables de diferentes marcas de cereales.

Enlace

El objetivo es predecir la valoración de las variables de los cereales como calorías, proteínas, grasas, etc.

Un valor de -1 para nutrientes indica que falta una observación.

- **Número de casos:** 77
- **Nombres de variables:**
  - `name`: Name of cereal
  - `mfr`: Manufacturer of cereal where A = American Home Food Products; G = General Mills; K = Kelloggs; N = Nabisco; P = Post; Q = Quaker Oats; R = Ralston Purina
  - `type`: cold or hot
  - `calories`: calories per serving
  - `protein`: grams of protein
  - 'fat: grams of fat
  - `sodium`: milligrams of sodium
  - `fiber`: grams of dietary fiber
  - `carbo`: grams of complex carbohydrates
  - `sugars`: grams of sugars
  - `potass`: milligrams of potassium
  - `vitamins`: vitamins and minerals - 0, 25, or 100, indicating the typical percentage of FDA recommended shelf: display shelf (1, 2, or 3, counting from the floor)
  - 'weight: weight in ounces of one serving
  - `cups`: number of cups in one serving
  - `rating`: a rating of the cereals

```
# Read the Data
data = read.csv("http://idaejin.github.io/bcam-courses/pucv/data/cereals.csv", header=T)

# Random sampling
samplesize = 0.60 * nrow(data)
set.seed(80)
index = sample( seq_len ( nrow ( data ) ), size = samplesize )
```

```
# Create training and test set
datatrain = data[ index, ]
 datatest = data[ -index, ]
```

Ahora ajustaremos una red neuronal en a los datos. Utilizamos la librería `neuralnet` para el análisis.

El primer paso es escalar el conjunto de datos de cereales. El escalamiento de los datos es esencial porque, de lo contrario, una variable puede tener un gran impacto en la variable de predicción sólo debido a su escala. El uso sin escala puede conducir a resultados sin sentido.

Las técnicas comunes para escalar los datos son: *normalización mín-máx*, *normalización del Z-score*, *mediana* y *MAD* (mean absolute deviation), y estimadores tan-h. La normalización mín-máx transforma los datos en un rango común, eliminando así el efecto de escala de todas las variables.

A diferencia de la normalización del Z-score y el método de la mediana y MAD, el método min-max retiene la distribución original de las variables. Usamos la normalización min-max para escalar los datos.

```
## Scale data for neural network

max = apply(data , 2 , max)
min = apply(data, 2 , min)
scaled = as.data.frame(scale(data, center = min, scale = max - min))
```

```
## Fit neural network
library(neuralnet)
# creating training and test set
trainNN = scaled[index , ]
 testNN = scaled[-index , ]

# fit neural network
set.seed(2)
NN = neuralnet(rating ~ calories + protein + fat + sodium + fiber, trainNN, hidden = 3 , linear.output =

# plot neural network
plot(NN)
```

El modelo tiene 3 neuronas en su capa oculta. Las líneas negras muestran las conexiones con los pesos. Los pesos se calculan utilizando el *algoritmo de backward progapation*. La línea azul es la que muestra el término de sesgo.

Predecimos la puntuación usando el modelo de red neuronal. Recordar que la puntación pronosticada será escalada y debe transformarse para poder hacer una comparación con la calificación real. También comparamos la puntación pronosticada con la puntación real. El RMSE para el modelo de red neural es 6.05.

```
## Prediction using neural network

predict_testNN = compute(NN, testNN[,c(1:5)])
predict_testNN = (predict_testNN$net.result * (max(data$rating) - min(data$rating))) + min(data$rating)

plot(datatest$rating, predict_testNN, col='blue', pch=16, ylab = "predicted rating NN", xlab = "real ra

abline(0,1)
```

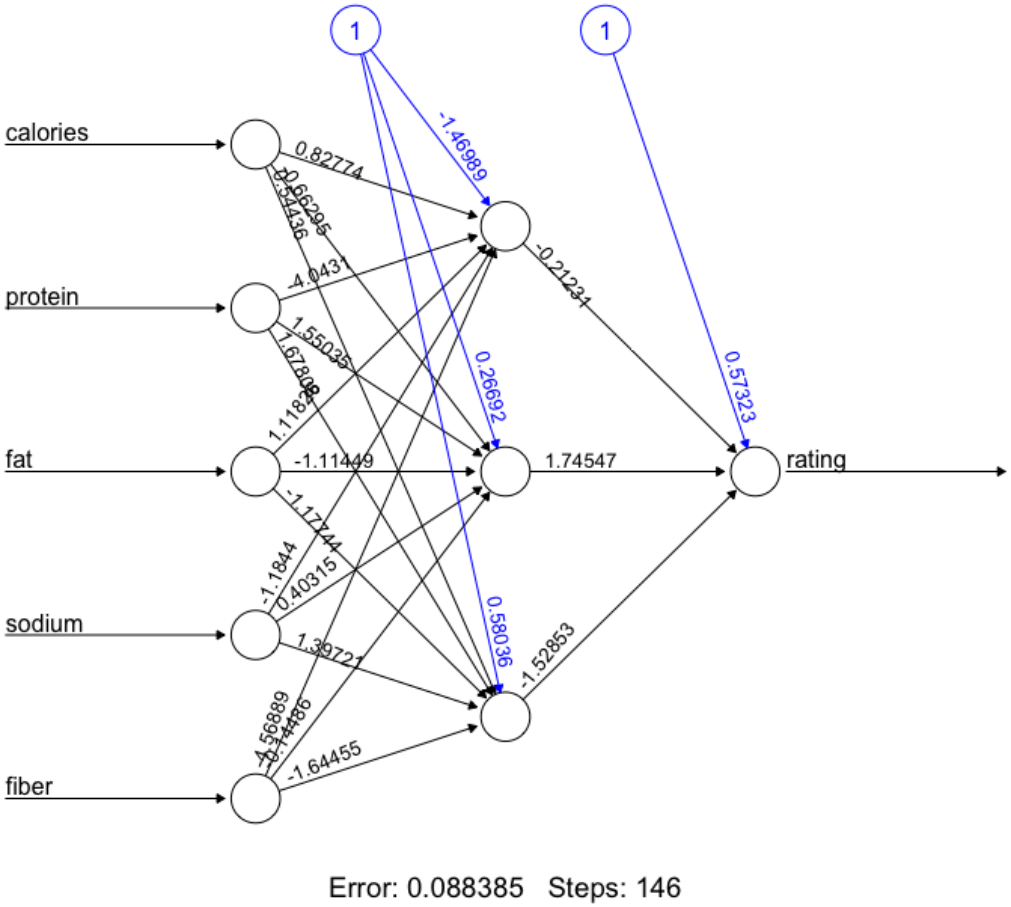
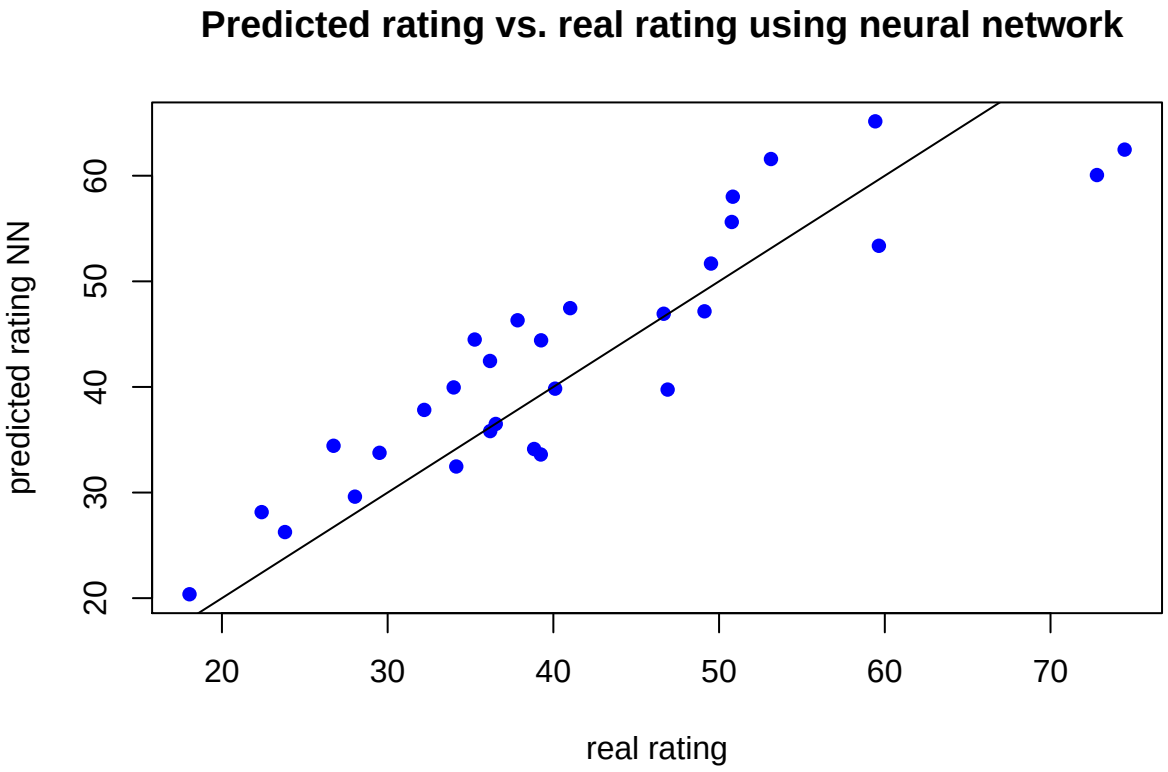

1
1
calories
protein
fat
sodium
fiber
0.82774
-0.66295
-0.54436
-4.0431
1.55035
1.67808
1.11824
-1.11449
-1.17244
-1.1844
0.40315
1.39721
-0.56889
-0.74486
-1.64455
-1.46989
-0.21234
0.26692
0.57323
1.74547
0.58036
-1.52853
rating
Error: 0.088385 Steps: 146

Figure 6:




```
# Calculate Root Mean Square Error (RMSE)
RMSE.NN = (sum((datatest$rating - predict_testNN)^2) / nrow(datatest)) ^ 0.5
```

Validación cruzada de una red neuronal

Hemos evaluado nuestro método de red neural usando *RMSE*, que es un método de evaluación basado en los residuos del modelo. El principal problema de los métodos de evaluación residuales es que no nos informa sobre el comportamiento de nuestro modelo cuando se introducen nuevos datos.

Intentamos resolver el problema de los *nuevos datos* dividiendo nuestros datos en entrenamiento y test, construyendo el modelo sobre la muestra de entrenamiento y evaluando el modelo mediante el cálculo de RMSE para el conjunto o muestra de test. La división de la prueba de entrenamiento no es más que la forma más simple de método de validación cruzada conocido como método de retención (*holdout method*). Una limitación del método de retención es la varianza de la métrica de evaluación del rendimiento, en nuestro caso RMSE, que puede ser alta en función de los elementos asignados la muestra de entrenamiento y al conjunto de test.

La segunda técnica de validación cruzada más común es la validación cruzada *k-fold*. Este método puede ser visto como un método de retención recurrente. Los datos completos se dividen en subconjuntos k iguales y cada vez que se asigna un subconjunto como conjunto de prueba, mientras que otros se utilizan para entrenar el modelo. Cada punto de datos tiene la oportunidad de estar en el conjunto de entrenamiento y en el conjunto de test, por lo que este método reduce la dependencia del rendimiento de la división del entrenamiento de prueba y reduce la varianza de las métricas de rendimiento. El caso extremo de la validación cruzada k se producirá cuando k es igual al número de puntos de datos. Significaría que el modelo predictivo se entrena sobre todos los puntos de datos excepto un punto de datos, que toma el papel de un conjunto de pruebas. Este método de dejar un punto de datos como conjunto de pruebas se conoce como *leave-one-out cross validation*.

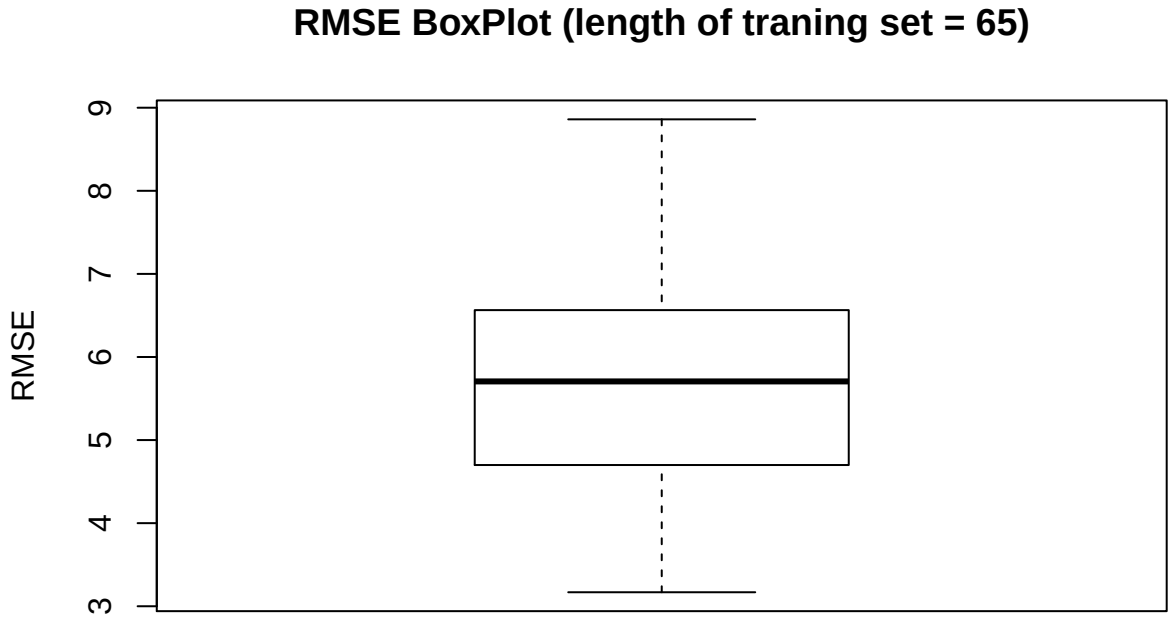
Ahora vamos a realizar la validación cruzada de k-fold en el modelo de red neuronal que construimos en la sección anterior. El número de elementos en el conjunto de entrenamiento, j, varía de 10 a 65 y por cada j, se extraen 100 muestras del conjunto de datos. El resto de los elementos en cada caso se asignan al conjunto de test. El modelo se entrena en cada uno de los 5600 conjuntos de datos de entrenamiento y luego se prueba en

los conjuntos de pruebas correspondientes. Calculamos el RMSE de cada uno de los equipos de prueba. Los valores RMSE para cada uno de los conjuntos se almacenan en una matriz de tamaño 100×56. Este método asegura que nuestros resultados estén libres de cualquier sesgo de la muestra y comprueba la robustez de nuestro modelo.

```
## Cross validation of neural network model

# Load libraries
library(boot)
library(plyr)

# Initialize variables
set.seed(50)
k = 100
RMSE.NN = NULL

List = list( )

# Fit neural network model within nested for loop
for(j in 10:65){
    for (i in 1:k) {
        index = sample(1:nrow(data),j )

        trainNN = scaled[index,]
        testNN = scaled[-index,]
        datatest = data[-index,]

        NN = neuralnet(rating ~ calories + protein + fat + sodium + fiber, trainNN, hidden = 3, linear.
        predict_testNN = compute(NN,testNN[,c(1:5)])
        predict_testNN = (predict_testNN$net.result*(max(data$rating)-min(data$rating)))+min(data$ratin

        RMSE.NN [i]<- (sum((datatest$rating - predict_testNN)^2)/nrow(datatest))^0.5
    }
    List[[j]] = RMSE.NN
}

Matrix.RMSE = do.call(cbind, List)
```

Se puede acceder a los valores RMSE utilizando la variable `Matrix.RMSE`. El tamaño de la matriz es grande, por lo que intentaremos dar sentido a los datos a través de visualizaciones. Primero, prepararemos una gráfica para una de las columnas de `Matrix.RMSE`, donde el conjunto de entrenamiento tiene una longitud igual a 65. Se pueden preparar estos gráficos de boxplots para cada una de las muestras de entrenamiento (de 10 a 65). El guión R es el siguiente.

```
## Prepare boxplot
boxplot(Matrix.RMSE[,56], ylab = "RMSE", main = "RMSE BoxPlot (length of traning set = 65)")
```



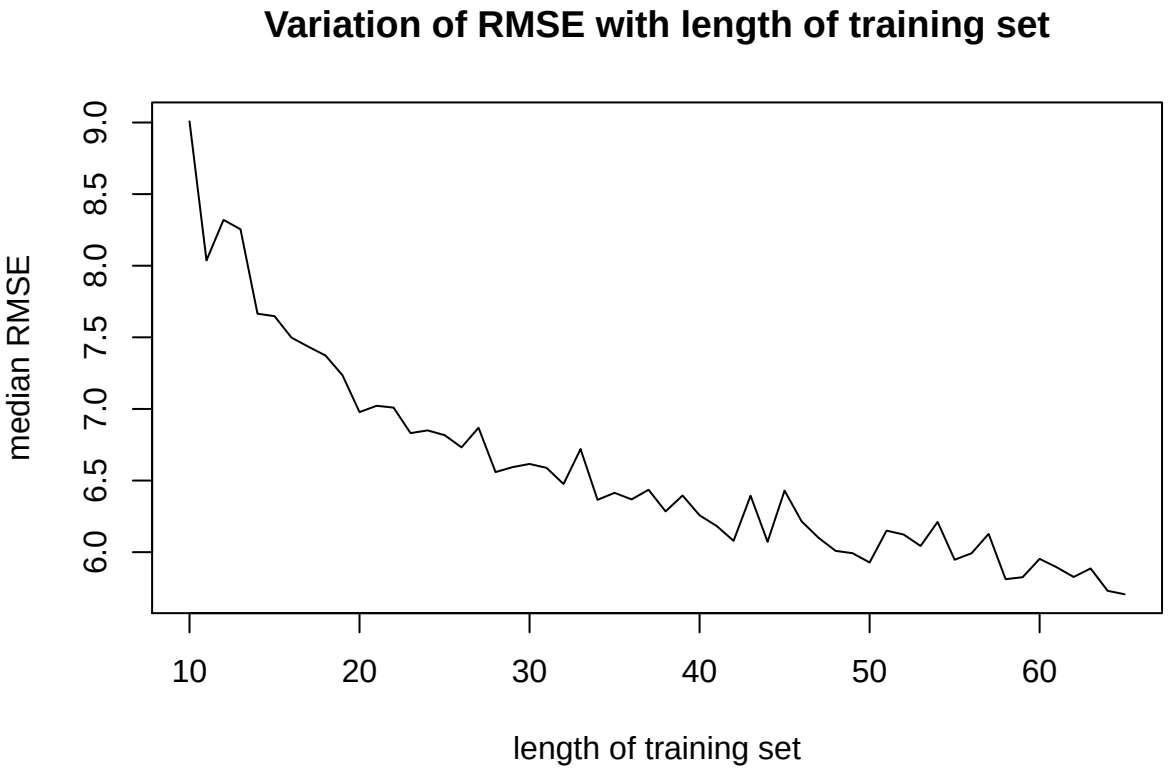
El boxplot muestra que la mediana de RMSE a través de 100 muestras cuando la duración del conjunto de entrenamiento está fijada en 65 es de $5,70$. En la siguiente visualización estudiamos la variación de RMSE con la duración del conjunto de entrenamiento. Calculamos la mediana de RMSE para cada una de las longitudes del conjunto de entrenamiento y las graficamos.

```
## Variation of median RMSE

library(matrixStats)
```

```
##
## Attaching package: 'matrixStats'
```

```
## The following object is masked from 'package:plyr':
##
##     count
```

```
med = colMedians(Matrix.RMSE)

X = seq(10,65)

plot (med~X, type = "l", xlab = "length of training set", ylab = "median RMSE", main = "Variation of RM
```



librería `nnet`

Usemos el conjunto de datos del iris como ejemplo:

```
data("iris")
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
?iris
```

Podemos utilizar la función `nnet`

```
library(nnet)

set.seed(123)
samp <- c(sample(1:50,25), sample(51:100,25), sample(101:150,25))

ird <- data.frame(rbind(iris3[,,1], iris3[,,2], iris3[,,3]),
                  species = factor(c(rep("s",50), rep("c", 50), rep("v", 50))))

ir.nn <- nnet(species ~ ., data = ird, subset = samp, size = 2, rang = 0.1,
              decay = 5e-4, maxit = 200)
```

```
## # weights:  19
```

```
## initial  value 82.414224
## iter  10 value 35.511771
## iter  20 value 35.008180
## iter  30 value 34.878930
## iter  40 value 15.346315
## iter  50 value 5.597691
## iter  60 value 4.247841
## iter  70 value 1.529529
## iter  80 value 0.874907
## iter  90 value 0.851403
## iter 100 value 0.836149
## iter 110 value 0.831229
## iter 120 value 0.827126
## iter 130 value 0.824769
## iter 140 value 0.824074
## iter 150 value 0.824009
## iter 160 value 0.823935
## iter 170 value 0.823908
## iter 180 value 0.823898
## iter 190 value 0.823896
## iter 200 value 0.823895
## final  value 0.823895
## stopped after 200 iterations
```

```
table(ird$species[-samp], predict(ir.nn, ird[-samp,], type = "class"))
```

```
##
##      c  s  v
##   c 23  0  2
##   s  0 25  0
##   v  3  0 22
```

Librería `caret`

La librería `caret` (*classification and regression training*) contiene funciones para la construcción de modelos para problemas complejos de regresión y clasificación.

```
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'lattice'
```

```
## The following object is masked from 'package:boot':
##
##     melanoma
```

```
## Loading required package: ggplot2
```

```
inTrain <- createDataPartition(y=iris$Species, p=0.75, list=FALSE)   # We wish 75% for the trainset

train.set <- iris[inTrain,]
test.set  <- iris[-inTrain,]
nrow(train.set)/nrow(test.set) # should be around 3
```

```
## [1] 3.166666667
```

```r
model <- train(Species ~ ., train.set, method='nnet', trace = FALSE) # train
# we also add parameter 'preProc = c("center", "scale"))' at train() for centering and scaling the data
prediction <- predict(model, test.set[-5])                           # predict
table(prediction, test.set$Species)                                  # compare
```

```
##
## prediction   setosa versicolor virginica
##   setosa         12          0         0
##   versicolor      0         12         0
##   virginica       0          0        12
```

```r
# predict can also return the probability for each class:
prediction <- predict(model, test.set[-5], type="prob")
head(prediction)
```

```
##          setosa    versicolor       virginica
## 3  0.9747517873 0.02492969192 0.0003185207993
## 9  0.9662119438 0.03336402212 0.0004240340819
## 11 0.9779224825 0.02179832238 0.0002791951089
## 21 0.9725265340 0.02712741813 0.0003460478429
## 29 0.9765412952 0.02316237012 0.0002963346866
## 34 0.9805063742 0.01924654678 0.0002470790013
```