A Neural Network using Neuralnet

Kayanna Chandler

August 4, 2016

```r
# We are going to be using the Boston dataset in the MASS package

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

# The Boston dataset contains data about the housing values in the suburbs of Boston.
# The goal is to predict the median value (medv) of occupied homes

# checking for gaps in the data
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
```

```r
# since there are no gaps, we can proceed. If there were gaps or other problems,
# we would have to fix the dataset. Otherwise our network would perform poorly.

# We randomly split the data into a train set and a test set.

# random splitting of data
index <- sample(1:nrow(data), round(0.75*nrow(data)))
train <- data[index,]
test <- data[-index,]

# Here we fit a linear regression model and test it on the test set.
# Since we are dealing with a regression problem, we are going to use
# the mean squared error (MSE) as a measure of how much our predictions
# are far away from the real data.

# Linear regression on training data
linreg.fit <- glm(medv~., data=train)
summary(linreg.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
```

```r
# Prediction of testing data based on learning
predict.linreg <- predict(linreg.fit, test)
mean_squared_error.linreg <- sum((predict.linreg - test$medv)^2)/nrow(test)

# As a first step, we are going to address data preprocessing.
# It is good practice to normalize your data before training a neural network.
# If this step is ommited, your neural models may become useless.

# normalizing data
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,]

# Now that our data is ready, we create a neural netwrok with the configuration 13:5:3:1.
# this means there will be a 13-node input layer (corresponding to the 13 variables in
# Boston dataset) two hidden layers, one with 5 neurons and one with 3 nuerons, and
# an output layer of one neuron.

library(grid)
library(neuralnet)

nnames <- names(train_)
f <- as.formula(paste("medv ~", paste(nnames[!nnames %in%
"medv"], collapse = " + ")))
```

```r
nn <- neuralnet(f, data=train_, hidden=c(5,3), linear.output=T)

# The hidden argument accepts a vector with the number of neurons for each hidden layer,
# while the argument linear.output is used to specify whether we want to do regression
# Currently, there are no hard and fast rules for the configuration of Neural Networks.

# We can visualize the network by plottong it:
#plot(nn)

# This is the graphical representation of the model with the weights on each
# connection. The black lines show the connections between each layer and
# the weights on each connection while the blue lines show the bias term
# added in each step. The bias can be thought as the intercept of a linear model.

# Predict medv using test set and our neural network
predict.nn <- compute(nn, test_[,1:13])
predict.nn_ <- predict.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)

# Now we can try to predict the values for the test set and calculate the MSE.
# Remember that the net will output a normalized prediction,
# so we need to scale it back to make a meaningful comparison.

# We then compare the two MSEs.
mean_squared_error.nn <- sum((test.r - predict.nn_)^2)/nrow(test_)
print(paste(mean_squared_error.linreg, mean_squared_error.nn))
```

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

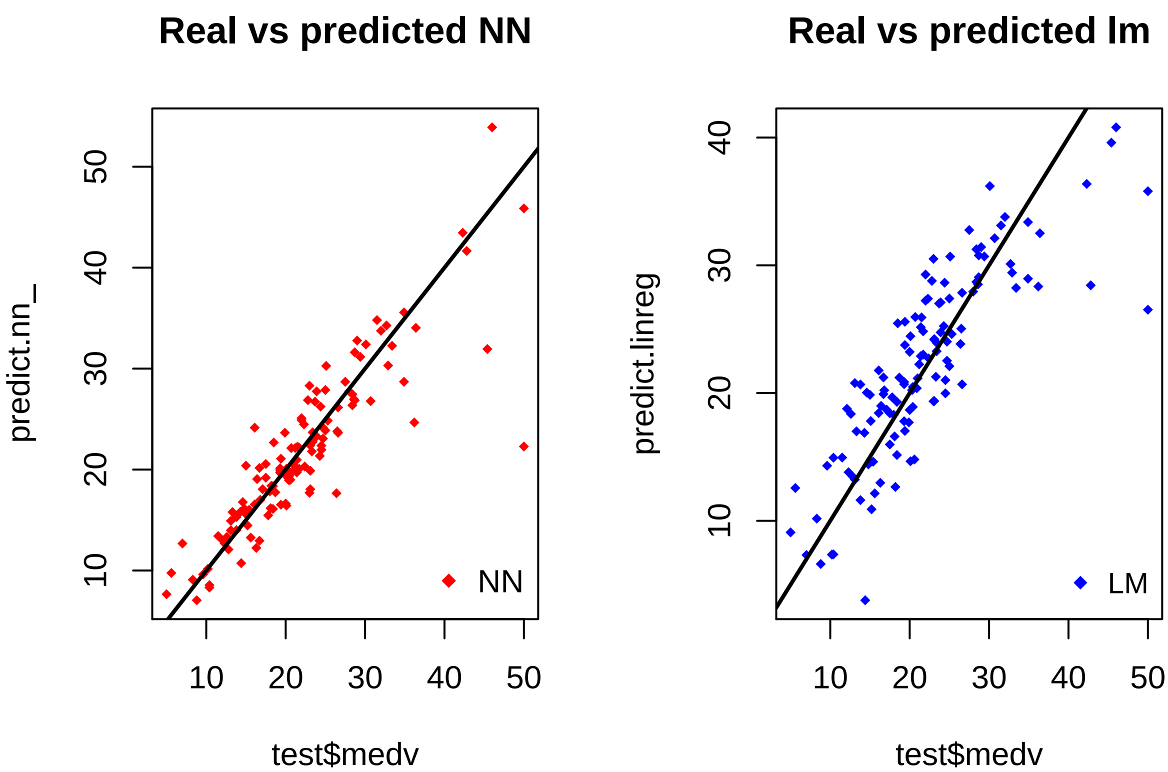
```r
# Apparently, the net is doing a better work than the linear model at predicting medv.

# A visual comparison of the performance of the network and the linear model

par(mfrow=c(1,2))

plot(test$medv,predict.nn_,col='red',main='Real vs predicted NN',pch=18,cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend='NN',pch=18,col='red', bty='n')

plot(test$medv,predict.linreg,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', bty='n', cex=.95)
```




```
# These graphs confirm that our neural network
# is performing better than the linear regression model.
```