# Neural Network Regression

*Karen Mazidi*

In this notebook we try a neural network on the Boston housing data. First we perform linear regression on the data.

### Linear regression

The correlation is 0.888 and the mse is 23.094.

```
library(MASS)
data("Boston")
df_boston <- Boston[]
set.seed(1234)
i <- sample(1:nrow(Boston), 0.75*nrow(Boston), replace=FALSE)
train <- df_boston[i,]
test <- df_boston[-i,]
lm1 <- lm(medv~., data=train)
summary(lm1)
```

```
##
## Call:
## lm(formula = medv ~ ., data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -12.6010  -2.8304  -0.4323   1.7485  25.6932
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  44.521498   6.048173   7.361 1.22e-12 ***
## crim         -0.089540   0.049306  -1.816  0.07019 .
## zn            0.047685   0.015656   3.046  0.00249 **
## indus         0.051049   0.072405   0.705  0.48123
## chas          2.128781   0.982105   2.168  0.03084 *
## nox         -18.906078   4.459331  -4.240 2.84e-05 ***
## rm            3.017641   0.485327   6.218 1.38e-09 ***
## age          -0.009964   0.015766  -0.632  0.52777
## dis          -1.494023   0.225626  -6.622 1.27e-10 ***
## rad           0.311572   0.079014   3.943 9.64e-05 ***
## tax          -0.013524   0.004354  -3.106  0.00205 **
## ptratio      -1.007642   0.154089  -6.539 2.09e-10 ***
## black         0.007522   0.003566   2.110  0.03558 *
## lstat        -0.528198   0.057159  -9.241  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.783 on 365 degrees of freedom
## Multiple R-squared:  0.7199, Adjusted R-squared:   0.71
## F-statistic: 72.17 on 13 and 365 DF,  p-value: < 2.2e-16
```

```
pred1 <- predict(lm1, newdata=test)
cor1 <- cor(pred1, test$medv)
mse1 <- mean((pred1-test$medv)^2)
```

### Data preprocessing

The neural network will perform better if the data is scaled. However, to analyze our results we will need to unscale it to get the original data back. We could use the unscale() function in package DMwR as demonstrated in Section 8.4, but instead we are going to scale manually. We write a function to normalize the data.

```
# function to normalize the data
normalize <- function(x){
  return ((x - min(x)) / (max(x) - min(x)))
}

# apply to all columns
train_scaled <- as.data.frame(lapply(train, normalize))
test_scaled <- as.data.frame(lapply(test, normalize))
```

### neural network

Unfortunately, neuralnet() can't handle regular R formulas so we build a string to hold the formula.

For the neuralnet() function we input the formula, the data and specify we want two hidden layers, the first with 6 hidden nodes and the second with 3. The linear.output parameter should be TRUE for regression and FALSE for classification.

How many nodes should we have in hidden layers? If there are too many it might cause overfitting but too few can cause underfitting. Different rules of thumb that are commonly discussed are:

- between 1 and the number of predictors
- 2/3 of the input layer size plus the size of the output layer
- less than twice the input layer size

These rules of thumbs suggest 1-13, 9, and $< 26$. Let's try 9 hidden nodes, 6 and 3.

The neural net algorithm initializes weights randomly (unless we specify initial weights). This random start will produce slightly different results each run. For this reason we set a seed to get reproducible results.

```
library(neuralnet)
n <- names(train)
f <- as.formula(paste("medv ~", paste(n[!n %in% "medv"], collapse = " + ")))
set.seed(1234)
nn1 <- neuralnet(f, data=train_scaled, hidden=c(6, 3),
                 linear.output = TRUE)
plot(nn1)
```

### results on test data

Correlation: 0.951 MSE: 15.479

This is an improvement over the linear regression model.

```r
pred2 <- compute(nn1, test_scaled[,1:13])
# scaled correlation
cor2_scaled <- cor(pred2$net.result, test_scaled$medv)
# unscaled original values correlation
pred2_unscaled <- pred2$net.result * (max(test$medv) - min(test$medv)) + min(test$medv)
cor2_unscaled <- cor(pred2_unscaled, test_scaled$medv)
mse2 <- mean((pred2_unscaled - test$medv)^2)
```