wineTestAgain.r

## File: WineTestAgain,r  
## Author: A. Breitzman  
## Date: 2/19/2017. Revised 5/23/2018  
## Description: Multiple tests of wine quality using multiple  
## hidden node/layer combinations

## The original idea was to build a white wine model and see how it performs   
## on the red wine set, but morphed into something else so we never got past  
## the white wine tests

library("neuralnet")  
library("plyr")  
setwd("C:/Users/User/Dropbox/Rowan/DM2/Lecture4")  
whiteWineData <-read.csv("winequality-white.csv",header = TRUE, sep = ";", stringsAsFactors = TRUE)  
redWineData <-read.csv("winequality-red.csv",header = TRUE, sep = ";", stringsAsFactors = TRUE)  
  
whiteWineData[1,]

## fixed.acidity volatile.acidity citric.acid residual.sugar chlorides  
## 1 7 0.27 0.36 20.7 0.045  
## free.sulfur.dioxide total.sulfur.dioxide density pH sulphates alcohol  
## 1 45 170 1.001 3 0.45 8.8  
## quality  
## 1 6

names(whiteWineData)

## [1] "fixed.acidity" "volatile.acidity" "citric.acid"   
## [4] "residual.sugar" "chlorides" "free.sulfur.dioxide"   
## [7] "total.sulfur.dioxide" "density" "pH"   
## [10] "sulphates" "alcohol" "quality"

# Normalizing white wine data  
# We don't have to do it one variable at a time  
# Here's how we might do min/max normalization on   
# fixedAcidity  
minFixedAcidity = min(whiteWineData$fixed.acidity)  
maxFixedAcidity = max(whiteWineData$fixed.acidity)  
temp = (whiteWineData$fixed.acidity - minFixedAcidity)/(maxFixedAcidity - minFixedAcidity)

# We can do all variables at once using the following trick  
maxs <- apply(whiteWineData, 2, max)  
mins <- apply(whiteWineData, 2, min)  
scaledWhite <- as.data.frame(scale(whiteWineData, center = mins, scale = maxs - mins))  
  
# Let's make sure this works by comparing temp above to scaledWhit$fixed.acidity  
cbind(temp,scaledWhite$fixed.acidity)[1:10,]

## temp   
## [1,] 0.3076923 0.3076923  
## [2,] 0.2403846 0.2403846  
## [3,] 0.4134615 0.4134615  
## [4,] 0.3269231 0.3269231  
## [5,] 0.3269231 0.3269231  
## [6,] 0.4134615 0.4134615  
## [7,] 0.2307692 0.2307692  
## [8,] 0.3076923 0.3076923  
## [9,] 0.2403846 0.2403846  
## [10,] 0.4134615 0.4134615

# perfect!  
  
# We'll scale the red wine data using the same min and max since  
# we need the same norms if we're going to use the white wine  
# predictor on the red wine set  
scaledRed <- as.data.frame(scale(redWineData, center = mins, scale = maxs - mins))  
  
names(scaledWhite)

## [1] "fixed.acidity" "volatile.acidity" "citric.acid"   
## [4] "residual.sugar" "chlorides" "free.sulfur.dioxide"   
## [7] "total.sulfur.dioxide" "density" "pH"   
## [10] "sulphates" "alcohol" "quality"

# change names for convenience. We can change some back later if we find some are more useful than others

names(scaledWhite) <- c("a1","a2","a3","a4","a5","a6","a7","a8","a9","a10","a11","Q")  
names(scaledRed) <- names(scaledWhite)  
formula = Q~a1+a2+a3+a4+a5+a6+a7+a8+a9+a10+a11

set.seed(2)  
train <- sample(1:nrow(scaledWhite),nrow(scaledWhite)\*(8/10))  
test<- -train  
  
trainingData<-scaledWhite[train,]  
testingData<-scaledWhite[test,]  
dim(trainingData)

## [1] 3918 12

dim(testingData)

## [1] 980 12

names(trainingData)

## [1] "a1" "a2" "a3" "a4" "a5" "a6" "a7" "a8" "a9" "a10" "a11"  
## [12] "Q"

# Make the actual neural net  
  
# Function that will add columns for doing some error testing  
computeResultColumns <- function(nnet,data1){  
 result <- compute(nnet, data1[,1:11])  
 data1$result <- sapply(result$net.result, function(b) {  
 return(b)  
 })  
 ## unscale originalQuality  
 data1$origQ <-apply(data1, MARGIN=1, function(x){  
 return(trunc(3+6\*x["Q"]))  
 })  
 data1$predQ <-apply(data1,MARGIN=1,function(x){  
 return(floor(3.5 +6\*x["result"]))  
 })  
 data1$error <- apply(data1, MARGIN=1, function(x) {  
 return(abs(x["origQ"] - x["predQ"]))  
 })  
 return(data1)  
}  
  
  
trainingData<-scaledWhite[train,]  
testingData<-scaledWhite[test,]

for (i in 0:16){  
   
 nnet<- neuralnet(formula, testingData, hidden = i, threshold = 0.1)  
   
 testCopy <- computeResultColumns(nnet,testingData)  
 print("testErr")  
 print(c(i,sum(testCopy$error)/nrow(testCopy)))  
 print(count(testCopy,c("origQ","predQ")))  
   
 trainCopy <- computeResultColumns(nnet,trainingData)  
 print("trainErr")  
 print(c(i,sum(trainCopy$error)/nrow(trainCopy)))  
 print(count(trainCopy,c("origQ","predQ")))  
  
}

## 0 Hidden nodes .525 error on average  
## this means that on average if the wine is rated 8 we return between 7.47 ## and 8.53. This sounds better than it actually is, if we look at the  
## actual counts

## [1] "testErr"  
## [1] 0.0000000000 0.5255102041  
## origQ predQ freq  
## 1 3 4 1  
## 2 3 5 3  
## 3 3 6 2  
## 4 4 4 1  
## 5 4 5 20  
## 6 4 6 15  
## 7 5 4 1  
## 8 5 5 144  
## 9 5 6 145  
## 10 5 7 3  
## 11 6 4 1  
## 12 6 5 72  
## 13 6 6 336  
## 14 6 7 38  
## 15 7 5 6  
## 16 7 6 113  
## 17 7 7 42  
## 18 8 5 1  
## 19 8 6 21  
## 20 8 7 13  
## 21 9 6 1  
## 22 9 7 1

## Note that the training error is higher than the testing error  
## so there is definitely no over-fitting here  
## [1] "trainErr"  
## [1] 0.0000000000 0.5444104135  
## origQ predQ freq  
## 1 3 5 4  
## 2 3 6 9  
## 3 3 7 1  
## 4 4 4 4  
## 5 4 5 66  
## 6 4 6 56  
## 7 4 7 1  
## 8 5 4 10  
## 9 5 5 534  
## 10 5 6 603  
## 11 5 7 17  
## 12 6 4 2  
## 13 6 5 276  
## 14 6 6 1315  
## 15 6 7 158  
## 16 7 5 37  
## 17 7 6 510  
## 18 7 7 172  
## 19 8 5 8  
## 20 8 6 81  
## 21 8 7 51  
## 22 9 6 1  
## 23 9 7 2  
## [1] "testErr"  
## [1] 1.0000000000 0.5224489796  
## origQ predQ freq  
## 1 3 4 2  
## 2 3 5 2  
## 3 3 6 2  
## 4 4 4 2  
## 5 4 5 19  
## 6 4 6 15  
## 7 5 4 7  
## 8 5 5 128  
## 9 5 6 156  
## 10 5 7 2  
## 11 6 4 2  
## 12 6 5 63  
## 13 6 6 358  
## 14 6 7 24  
## 15 7 6 129  
## 16 7 7 32  
## 17 8 6 23  
## 18 8 7 12  
## 19 9 6 2  
## [1] "trainErr"  
## [1] 1.0000000000 0.5370086779  
## origQ predQ freq  
## 1 3 5 4  
## 2 3 6 9  
## 3 3 7 1  
## 4 4 3 2  
## 5 4 4 11  
## 6 4 5 60  
## 7 4 6 53  
## 8 4 7 1  
## 9 5 4 26  
## 10 5 5 479  
## 11 5 6 654  
## 12 5 7 5  
## 13 6 4 5  
## 14 6 5 243  
## 15 6 6 1410  
## 16 6 7 93  
## 17 7 5 17  
## 18 7 6 574  
## 19 7 7 128  
## 20 8 5 1  
## 21 8 6 100  
## 22 8 7 39  
## 23 9 6 2  
## 24 9 7 1  
## [1] "testErr"  
## [1] 2.0000000000 0.5234693878  
## origQ predQ freq  
## 1 3 4 1  
## 2 3 5 3  
## 3 3 6 2  
## 4 4 4 1  
## 5 4 5 24  
## 6 4 6 11  
## 7 5 5 146  
## 8 5 6 144  
## 9 5 7 3  
## 10 6 5 76  
## 11 6 6 333  
## 12 6 7 38  
## 13 7 5 9  
## 14 7 6 109  
## 15 7 7 43  
## 16 8 5 1  
## 17 8 6 20  
## 18 8 7 14  
## 19 9 6 2

## [1] "trainErr"  
## [1] 2.0000000000 0.5410923941  
## origQ predQ freq  
## 1 3 5 4  
## 2 3 6 9  
## 3 3 7 1  
## 4 4 4 3  
## 5 4 5 77  
## 6 4 6 46  
## 7 4 7 1  
## 8 5 4 3  
## 9 5 5 557  
## 10 5 6 583  
## 11 5 7 21  
## 12 6 5 298  
## 13 6 6 1288  
## 14 6 7 165  
## 15 7 5 37  
## 16 7 6 500  
## 17 7 7 182  
## 18 8 5 8  
## 19 8 6 81  
## 20 8 7 51  
## 21 9 6 1  
## 22 9 7 2  
## [1] "testErr"  
## [1] 3.0000000000 0.5132653061  
## origQ predQ freq  
## 1 3 4 2  
## 2 3 5 1  
## 3 3 6 3  
## 4 4 4 6  
## 5 4 5 14  
## 6 4 6 16  
## 7 5 4 4  
## 8 5 5 125  
## 9 5 6 162  
## 10 5 7 2  
## 11 6 4 2  
## 12 6 5 56  
## 13 6 6 363  
## 14 6 7 26  
## 15 7 5 2  
## 16 7 6 122  
## 17 7 7 37  
## 18 8 6 22  
## 19 8 7 13  
## 20 9 6 1  
## 21 9 7 1

## [1] "trainErr"  
## [1] 3.0000000000 0.5436447167  
## origQ predQ freq  
## 1 3 5 3  
## 2 3 6 11  
## 3 4 3 1  
## 4 4 4 13  
## 5 4 5 54  
## 6 4 6 58  
## 7 4 7 1  
## 8 5 4 28  
## 9 5 5 462  
## 10 5 6 667  
## 11 5 7 7  
## 12 6 4 7  
## 13 6 5 222  
## 14 6 6 1397  
## 15 6 7 125  
## 16 7 5 20  
## 17 7 6 558  
## 18 7 7 141  
## 19 8 5 1  
## 20 8 6 99  
## 21 8 7 40  
## 22 9 6 2  
## 23 9 7 1  
## [1] "testErr"  
## [1] 4.0000000000 0.5010204082  
## origQ predQ freq  
## 1 3 4 1  
## 2 3 5 2  
## 3 3 6 3  
## 4 4 4 4  
## 5 4 5 19  
## 6 4 6 13  
## 7 5 4 2  
## 8 5 5 143  
## 9 5 6 145  
## 10 5 7 3  
## 11 6 5 64  
## 12 6 6 344  
## 13 6 7 39  
## 14 7 6 112  
## 15 7 7 47  
## 16 7 8 2  
## 17 8 6 22  
## 18 8 7 13  
## 19 9 6 1  
## 20 9 7 1

## [1] "trainErr"  
## [1] 4.0000000000 0.5334354262  
## origQ predQ freq  
## 1 3 5 3  
## 2 3 6 10  
## 3 3 7 1  
## 4 4 4 10  
## 5 4 5 63  
## 6 4 6 53  
## 7 4 7 1  
## 8 5 4 20  
## 9 5 5 502  
## 10 5 6 623  
## 11 5 7 19  
## 12 6 4 3  
## 13 6 5 248  
## 14 6 6 1353  
## 15 6 7 146  
## 16 6 8 1  
## 17 7 5 14  
## 18 7 6 528  
## 19 7 7 175  
## 20 7 8 2  
## 21 8 5 1  
## 22 8 6 89  
## 23 8 7 49  
## 24 8 8 1  
## 25 9 6 1  
## 26 9 7 2  
## [1] "testErr"  
## [1] 5.0000000000 0.5132653061  
## origQ predQ freq  
## 1 3 4 2  
## 2 3 5 1  
## 3 3 6 3  
## 4 4 4 5  
## 5 4 5 22  
## 6 4 6 9  
## 7 5 4 5  
## 8 5 5 147  
## 9 5 6 139  
## 10 5 7 2  
## 11 6 5 69  
## 12 6 6 323  
## 13 6 7 55  
## 14 7 5 6  
## 15 7 6 106  
## 16 7 7 49  
## 17 8 6 21  
## 18 8 7 14  
## 19 9 7 2  
## [1] "trainErr"  
## [1] 5.0000000000 0.5296069423  
## origQ predQ freq  
## 1 3 4 1  
## 2 3 5 7  
## 3 3 6 6  
## 4 4 4 13  
## 5 4 5 78  
## 6 4 6 36  
## 7 5 4 17  
## 8 5 5 563  
## 9 5 6 566  
## 10 5 7 17  
## 11 5 8 1  
## 12 6 2 1  
## 13 6 4 8  
## 14 6 5 303  
## 15 6 6 1251  
## 16 6 7 188  
## 17 7 5 20  
## 18 7 6 486  
## 19 7 7 213  
## 20 8 5 4  
## 21 8 6 80  
## 22 8 7 56  
## 23 9 6 1  
## 24 9 7 2  
## [1] "testErr"  
## [1] 6.0000000000 0.5132653061  
## origQ predQ freq  
## 1 3 3 1  
## 2 3 4 1  
## 3 3 5 2  
## 4 3 6 2  
## 5 4 4 1  
## 6 4 5 20  
## 7 4 6 15  
## 8 5 4 3  
## 9 5 5 141  
## 10 5 6 147  
## 11 5 7 2  
## 12 6 4 2  
## 13 6 5 69  
## 14 6 6 337  
## 15 6 7 39  
## 16 7 5 1  
## 17 7 6 113  
## 18 7 7 47  
## 19 8 6 22  
## 20 8 7 13  
## 21 9 7 2  
## [1] "trainErr"  
## [1] 6.000000000 0.540071465  
## origQ predQ freq  
## 1 3 5 4  
## 2 3 6 9  
## 3 3 7 1  
## 4 4 4 7  
## 5 4 5 64  
## 6 4 6 55  
## 7 4 7 1  
## 8 5 4 15  
## 9 5 5 520  
## 10 5 6 615  
## 11 5 7 14  
## 12 6 4 4  
## 13 6 5 268  
## 14 6 6 1321  
## 15 6 7 158  
## 16 7 5 35  
## 17 7 6 503  
## 18 7 7 181  
## 19 8 5 1  
## 20 8 6 86  
## 21 8 7 53  
## 22 9 6 1  
## 23 9 7 2  
## [1] "testErr"  
## [1] 7.0000000000 0.4653061224  
## origQ predQ freq  
## 1 3 4 5  
## 2 3 5 1  
## 3 4 3 1  
## 4 4 4 12  
## 5 4 5 14  
## 6 4 6 9  
## 7 5 4 3  
## 8 5 5 149  
## 9 5 6 138  
## 10 5 7 3  
## 11 6 5 68  
## 12 6 6 333  
## 13 6 7 46  
## 14 7 5 2  
## 15 7 6 96  
## 16 7 7 63  
## 17 8 6 16  
## 18 8 7 19  
## 19 9 7 2  
## [1] "trainErr"  
## [1] 7.0000000000 0.5316488004  
## origQ predQ freq  
## 1 3 3 1  
## 2 3 4 4  
## 3 3 5 3  
## 4 3 6 6  
## 5 4 2 1  
## 6 4 3 4  
## 7 4 4 17  
## 8 4 5 69  
## 9 4 6 35  
## 10 4 7 1  
## 11 5 3 3  
## 12 5 4 38  
## 13 5 5 519  
## 14 5 6 576  
## 15 5 7 28  
## 16 6 -5 1  
## 17 6 3 2  
## 18 6 4 9  
## 19 6 5 292  
## 20 6 6 1243  
## 21 6 7 204  
## 22 7 5 16  
## 23 7 6 443  
## 24 7 7 260  
## 25 8 5 3  
## 26 8 6 72  
## 27 8 7 65  
## 28 9 6 1  
## 29 9 7 2  
## [1] "testErr"  
## [1] 8.0000000000 0.4857142857  
## origQ predQ freq  
## 1 3 4 1  
## 2 3 5 3  
## 3 3 6 2  
## 4 4 4 8  
## 5 4 5 21  
## 6 4 6 7  
## 7 5 4 4  
## 8 5 5 157  
## 9 5 6 130  
## 10 5 7 2  
## 11 6 5 79  
## 12 6 6 307  
## 13 6 7 61  
## 14 7 5 2  
## 15 7 6 91  
## 16 7 7 68  
## 17 8 6 16  
## 18 8 7 19  
## 19 9 7 2  
## [1] "trainErr"  
## [1] 8.0000000000 0.5056151097  
## origQ predQ freq  
## 1 3 4 1  
## 2 3 5 7  
## 3 3 6 6  
## 4 4 3 1  
## 5 4 4 16  
## 6 4 5 82  
## 7 4 6 28  
## 8 5 4 27  
## 9 5 5 609  
## 10 5 6 510  
## 11 5 7 18  
## 12 6 3 1  
## 13 6 4 6  
## 14 6 5 314  
## 15 6 6 1219  
## 16 6 7 211  
## 17 7 5 12  
## 18 7 6 443  
## 19 7 7 264  
## 20 8 5 3  
## 21 8 6 76  
## 22 8 7 61  
## 23 9 6 1  
## 24 9 7 2

## [1] "testErr"  
## [1] 9.0000000000 0.4591836735  
## origQ predQ freq  
## 1 3 4 2  
## 2 3 5 3  
## 3 3 6 1  
## 4 4 3 1  
## 5 4 4 4  
## 6 4 5 23  
## 7 4 6 8  
## 8 5 4 2  
## 9 5 5 168  
## 10 5 6 120  
## 11 5 7 3  
## 12 6 4 1  
## 13 6 5 64  
## 14 6 6 325  
## 15 6 7 57  
## 16 7 5 2  
## 17 7 6 90  
## 18 7 7 69  
## 19 8 6 15  
## 20 8 7 20  
## 21 9 7 2  
## [1] "trainErr"  
## [1] 9.0000000000 0.5068912711  
## origQ predQ freq  
## 1 3 4 3  
## 2 3 5 5  
## 3 3 6 5  
## 4 3 7 1  
## 5 4 3 4  
## 6 4 4 12  
## 7 4 5 78  
## 8 4 6 31  
## 9 4 7 2  
## 10 5 4 26  
## 11 5 5 597  
## 12 5 6 516  
## 13 5 7 25  
## 14 6 2 1  
## 15 6 4 6  
## 16 6 5 322  
## 17 6 6 1211  
## 18 6 7 211  
## 19 7 5 18  
## 20 7 6 409  
## 21 7 7 292  
## 22 8 5 3  
## 23 8 6 65  
## 24 8 7 72  
## 25 9 6 1  
## 26 9 7 2  
## [1] "testErr"  
## [1] 10.0000000000 0.5112244898  
## origQ predQ freq  
## 1 3 4 1  
## 2 3 5 4  
## 3 3 6 1  
## 4 4 4 3  
## 5 4 5 26  
## 6 4 6 7  
## 7 5 4 4  
## 8 5 5 145  
## 9 5 6 142  
## 10 5 7 2  
## 11 6 4 1  
## 12 6 5 71  
## 13 6 6 327  
## 14 6 7 48  
## 15 7 5 8  
## 16 7 6 103  
## 17 7 7 50  
## 18 8 6 20  
## 19 8 7 15  
## 20 9 7 2  
## [1] "trainErr"  
## [1] 10.0000000000 0.5339458908  
## origQ predQ freq  
## 1 3 2 1  
## 2 3 4 2  
## 3 3 5 5  
## 4 3 6 6  
## 5 4 4 18  
## 6 4 5 78  
## 7 4 6 30  
## 8 4 7 1  
## 9 5 4 16  
## 10 5 5 540  
## 11 5 6 588  
## 12 5 7 20  
## 13 6 3 1  
## 14 6 4 3  
## 15 6 5 303  
## 16 6 6 1261  
## 17 6 7 182  
## 18 6 8 1  
## 19 7 5 24  
## 20 7 6 496  
## 21 7 7 199  
## 22 8 5 4  
## 23 8 6 81  
## 24 8 7 55  
## 25 9 6 1  
## 26 9 7 2  
## [1] "testErr"  
## [1] 11.0000000000 0.5010204082  
## origQ predQ freq  
## 1 3 4 4  
## 2 3 5 1  
## 3 3 6 1  
## 4 4 4 5  
## 5 4 5 23  
## 6 4 6 8  
## 7 5 4 6  
## 8 5 5 138  
## 9 5 6 147  
## 10 5 7 2  
## 11 6 5 64  
## 12 6 6 328  
## 13 6 7 55  
## 14 7 5 4  
## 15 7 6 98  
## 16 7 7 59  
## 17 8 6 22  
## 18 8 7 13  
## 19 9 7 2  
## [1] "trainErr"  
## [1] 11.0000000000 0.5201633486  
## origQ predQ freq  
## 1 3 1 2  
## 2 3 4 1  
## 3 3 5 5  
## 4 3 6 6  
## 5 4 3 2  
## 6 4 4 20  
## 7 4 5 70  
## 8 4 6 34  
## 9 4 7 1  
## 10 5 4 23  
## 11 5 5 525  
## 12 5 6 598  
## 13 5 7 18  
## 14 6 4 6  
## 15 6 5 271  
## 16 6 6 1301  
## 17 6 7 173  
## 18 7 5 20  
## 19 7 6 472  
## 20 7 7 225  
## 21 7 8 2  
## 22 8 5 3  
## 23 8 6 81  
## 24 8 7 56  
## 25 9 6 2  
## 26 9 7 1  
## [1] "testErr"  
## [1] 12.0000000000 0.4928571429  
## origQ predQ freq  
## 1 3 4 3  
## 2 3 5 2  
## 3 3 6 1  
## 4 4 4 7  
## 5 4 5 21  
## 6 4 6 8  
## 7 5 4 2  
## 8 5 5 147  
## 9 5 6 142  
## 10 5 7 2  
## 11 6 4 1  
## 12 6 5 66  
## 13 6 6 326  
## 14 6 7 54  
## 15 7 5 3  
## 16 7 6 101  
## 17 7 7 57  
## 18 8 6 20  
## 19 8 7 15  
## 20 9 7 2  
## [1] "trainErr"  
## [1] 12.0000000000 0.5158244002  
## origQ predQ freq  
## 1 3 4 2  
## 2 3 5 6  
## 3 3 6 6  
## 4 4 4 16  
## 5 4 5 80  
## 6 4 6 30  
## 7 4 7 1  
## 8 5 4 20  
## 9 5 5 549  
## 10 5 6 579  
## 11 5 7 16  
## 12 6 1 1  
## 13 6 4 4  
## 14 6 5 296  
## 15 6 6 1260  
## 16 6 7 190  
## 17 7 5 13  
## 18 7 6 462  
## 19 7 7 243  
## 20 7 8 1  
## 21 8 5 4  
## 22 8 6 72  
## 23 8 7 64  
## 24 9 6 1  
## 25 9 7 2  
## [1] "testErr"  
## [1] 13.0000000000 0.5081632653  
## origQ predQ freq  
## 1 3 4 2  
## 2 3 5 3  
## 3 3 6 1  
## 4 4 4 6  
## 5 4 5 21  
## 6 4 6 9  
## 7 5 4 6  
## 8 5 5 140  
## 9 5 6 145  
## 10 5 7 2  
## 11 6 4 1  
## 12 6 5 63  
## 13 6 6 334  
## 14 6 7 49  
## 15 7 5 5  
## 16 7 6 107  
## 17 7 7 49  
## 18 8 5 1  
## 19 8 6 20  
## 20 8 7 14  
## 21 9 6 1  
## 22 9 7 1  
## [1] "trainErr"  
## [1] 13.0000000000 0.5398162328  
## origQ predQ freq  
## 1 3 4 2  
## 2 3 5 6  
## 3 3 6 6  
## 4 4 4 12  
## 5 4 5 80  
## 6 4 6 34  
## 7 4 7 1  
## 8 5 4 15  
## 9 5 5 544  
## 10 5 6 583  
## 11 5 7 22  
## 12 6 4 5  
## 13 6 5 315  
## 14 6 6 1235  
## 15 6 7 196  
## 16 7 5 20  
## 17 7 6 489  
## 18 7 7 209  
## 19 7 8 1  
## 20 8 5 10  
## 21 8 6 72  
## 22 8 7 58  
## 23 9 6 1  
## 24 9 7 2  
## [1] "testErr"  
## [1] 14.0000000000 0.5030612245  
## origQ predQ freq  
## 1 3 5 5  
## 2 3 6 1  
## 3 4 4 4  
## 4 4 5 23  
## 5 4 6 9  
## 6 5 4 4  
## 7 5 5 144  
## 8 5 6 143  
## 9 5 7 2  
## 10 6 5 71  
## 11 6 6 328  
## 12 6 7 48  
## 13 7 5 5  
## 14 7 6 101  
## 15 7 7 55  
## 16 8 6 19  
## 17 8 7 16  
## 18 9 7 2  
## [1] "trainErr"  
## [1] 14.0000000000 0.5372639102  
## origQ predQ freq  
## 1 3 4 2  
## 2 3 5 5  
## 3 3 6 7  
## 4 4 3 1  
## 5 4 4 20  
## 6 4 5 68  
## 7 4 6 38  
## 8 5 4 26  
## 9 5 5 523  
## 10 5 6 588  
## 11 5 7 27  
## 12 6 4 8  
## 13 6 5 294  
## 14 6 6 1252  
## 15 6 7 196  
## 16 6 8 1  
## 17 7 5 18  
## 18 7 6 486  
## 19 7 7 214  
## 20 7 8 1  
## 21 8 5 3  
## 22 8 6 76  
## 23 8 7 60  
## 24 8 8 1  
## 25 9 6 1  
## 26 9 7 2  
## [1] "testErr"  
## [1] 15.0000000000 0.4551020408  
## origQ predQ freq  
## 1 3 3 1  
## 2 3 4 2  
## 3 3 5 3  
## 4 4 4 12  
## 5 4 5 16  
## 6 4 6 8  
## 7 5 4 5  
## 8 5 5 164  
## 9 5 6 121  
## 10 5 7 3  
## 11 6 4 1  
## 12 6 5 81  
## 13 6 6 310  
## 14 6 7 55  
## 15 7 5 2  
## 16 7 6 79  
## 17 7 7 79  
## 18 7 8 1  
## 19 8 6 12  
## 20 8 7 23  
## 21 9 6 1  
## 22 9 7 1  
## [1] "trainErr"  
## [1] 15.0000000000 0.5168453292  
## origQ predQ freq  
## 1 3 3 1  
## 2 3 5 5  
## 3 3 6 6  
## 4 3 7 1  
## 5 3 9 1  
## 6 4 3 1  
## 7 4 4 22  
## 8 4 5 76  
## 9 4 6 26  
## 10 4 7 2  
## 11 5 4 31  
## 12 5 5 620  
## 13 5 6 482  
## 14 5 7 28  
## 15 5 8 3  
## 16 6 2 1  
## 17 6 4 13  
## 18 6 5 337  
## 19 6 6 1161  
## 20 6 7 234  
## 21 6 8 5  
## 22 7 5 22  
## 23 7 6 399  
## 24 7 7 290  
## 25 7 8 8  
## 26 8 5 3  
## 27 8 6 59  
## 28 8 7 78  
## 29 9 6 1  
## 30 9 7 2  
## [1] "testErr"  
## [1] 16.0000000000 0.5153061224  
## origQ predQ freq  
## 1 3 4 2  
## 2 3 5 2  
## 3 3 6 2  
## 4 4 4 1  
## 5 4 5 24  
## 6 4 6 10  
## 7 4 7 1  
## 8 5 4 1  
## 9 5 5 139  
## 10 5 6 149  
## 11 5 7 4  
## 12 6 4 2  
## 13 6 5 72  
## 14 6 6 331  
## 15 6 7 42  
## 16 7 5 1  
## 17 7 6 109  
## 18 7 7 50  
## 19 7 8 1  
## 20 8 6 19  
## 21 8 7 15  
## 22 8 8 1  
## 23 9 6 1  
## 24 9 7 1  
## [1] "trainErr"  
## [1] 16.0000000000 0.5296069423  
## origQ predQ freq  
## 1 3 5 3  
## 2 3 6 11  
## 3 4 4 5  
## 4 4 5 68  
## 5 4 6 53  
## 6 4 7 1  
## 7 5 4 11  
## 8 5 5 553  
## 9 5 6 576  
## 10 5 7 23  
## 11 5 8 1  
## 12 6 4 4  
## 13 6 5 279  
## 14 6 6 1287  
## 15 6 7 180  
## 16 6 8 1  
## 17 7 5 17  
## 18 7 6 497  
## 19 7 7 203  
## 20 7 8 2  
## 21 8 5 1  
## 22 8 6 73  
## 23 8 7 65  
## 24 8 8 1  
## 25 9 6 1  
## 26 9 7 2

## Let's try some examples with 2 hidden layers  
  
for (i in 2:16){  
   
 nnet<- neuralnet(formula, testingData, hidden = c(i,ceiling(i/2)), threshold = 0.1)  
   
 testCopy <- computeAvgError(nnet,testingData)  
 print("testErr")  
 print(c(i,sum(testCopy$error)/nrow(testCopy)))  
 print(count(testCopy,c("origQ","predQ")))  
   
 trainCopy <- computeAvgError(nnet,trainingData)  
 print("trainErr")  
 print(c(i,sum(trainCopy$error)/nrow(trainCopy)))  
 print(count(trainCopy,c("origQ","predQ")))  
   
}

## [1] "testErr"  
## [1] 2.0000000000 0.5255102041  
## origQ predQ freq  
## 1 3 5 4  
## 2 3 6 2  
## 3 4 5 24  
## 4 4 6 12  
## 5 5 5 154  
## 6 5 6 136  
## 7 5 7 3  
## 8 6 5 83  
## 9 6 6 319  
## 10 6 7 45  
## 11 7 5 11  
## 12 7 6 100  
## 13 7 7 50  
## 14 8 5 1  
## 15 8 6 20  
## 16 8 7 14  
## 17 9 7 2  
## [1] "trainErr"  
## [1] 2.0000000000 0.5433894844  
## origQ predQ freq  
## 1 3 5 4  
## 2 3 6 10  
## 3 4 5 81  
## 4 4 6 44  
## 5 4 7 2  
## 6 5 5 584  
## 7 5 6 563  
## 8 5 7 17  
## 9 6 5 325  
## 10 6 6 1238  
## 11 6 7 188  
## 12 7 5 45  
## 13 7 6 472  
## 14 7 7 202  
## 15 8 5 8  
## 16 8 6 81  
## 17 8 7 51  
## 18 9 6 1  
## 19 9 7 2  
## [1] "testErr"  
## [1] 3.0000000000 0.5081632653  
## origQ predQ freq  
## 1 3 4 1  
## 2 3 5 3  
## 3 3 6 2  
## 4 4 4 1  
## 5 4 5 24  
## 6 4 6 11  
## 7 5 5 144  
## 8 5 6 146  
## 9 5 7 3  
## 10 6 5 71  
## 11 6 6 334  
## 12 6 7 42  
## 13 7 5 1  
## 14 7 6 111  
## 15 7 7 49  
## 16 8 5 1  
## 17 8 6 20  
## 18 8 7 14  
## 19 9 7 2  
## [1] "trainErr"  
## [1] 3.0000000000 0.5354772843  
## origQ predQ freq  
## 1 3 5 4  
## 2 3 6 10  
## 3 4 4 3  
## 4 4 5 76  
## 5 4 6 46  
## 6 4 7 2  
## 7 5 4 1  
## 8 5 5 549  
## 9 5 6 598  
## 10 5 7 16  
## 11 6 5 292  
## 12 6 6 1274  
## 13 6 7 185  
## 14 7 5 20  
## 15 7 6 499  
## 16 7 7 200  
## 17 8 5 1  
## 18 8 6 90  
## 19 8 7 49  
## 20 9 6 1  
## 21 9 7 2  
## [1] "testErr"  
## [1] 4.0000000000 0.5132653061  
## origQ predQ freq  
## 1 3 4 1  
## 2 3 5 3  
## 3 3 6 2  
## 4 4 4 3  
## 5 4 5 21  
## 6 4 6 12  
## 7 5 4 2  
## 8 5 5 140  
## 9 5 6 148  
## 10 5 7 3  
## 11 6 4 2  
## 12 6 5 63  
## 13 6 6 345  
## 14 6 7 37  
## 15 7 5 5  
## 16 7 6 114  
## 17 7 7 42  
## 18 8 6 22  
## 19 8 7 13  
## 20 9 7 2  
## [1] "trainErr"  
## [1] 4.0000000000 0.5403266973  
## origQ predQ freq  
## 1 3 5 3  
## 2 3 6 11  
## 3 4 4 8  
## 4 4 5 61  
## 5 4 6 57  
## 6 4 7 1  
## 7 5 4 13  
## 8 5 5 521  
## 9 5 6 614  
## 10 5 7 16  
## 11 6 4 4  
## 12 6 5 261  
## 13 6 6 1335  
## 14 6 7 151  
## 15 7 5 39  
## 16 7 6 505  
## 17 7 7 175  
## 18 8 5 1  
## 19 8 6 89  
## 20 8 7 50  
## 21 9 6 1  
## 22 9 7 2  
## [1] "testErr"  
## [1] 5.0000000000 0.5132653061  
## origQ predQ freq  
## 1 3 4 1  
## 2 3 5 3  
## 3 3 6 2  
## 4 4 4 1  
## 5 4 5 23  
## 6 4 6 12  
## 7 5 5 143  
## 8 5 6 147  
## 9 5 7 3  
## 10 6 4 1  
## 11 6 5 70  
## 12 6 6 338  
## 13 6 7 38  
## 14 7 5 2  
## 15 7 6 115  
## 16 7 7 44  
## 17 8 6 21  
## 18 8 7 14  
## 19 9 6 1  
## 20 9 7 1  
## [1] "trainErr"  
## [1] 5.0000000000 0.5321592649  
## origQ predQ freq  
## 1 3 5 4  
## 2 3 6 10  
## 3 4 4 5  
## 4 4 5 74  
## 5 4 6 47  
## 6 4 7 1  
## 7 5 4 6  
## 8 5 5 540  
## 9 5 6 601  
## 10 5 7 17  
## 11 6 4 1  
## 12 6 5 277  
## 13 6 6 1321  
## 14 6 7 152  
## 15 7 5 25  
## 16 7 6 514  
## 17 7 7 180  
## 18 8 5 3  
## 19 8 6 87  
## 20 8 7 50  
## 21 9 6 1  
## 22 9 7 2  
## [1] "testErr"  
## [1] 6.0000000000 0.5153061224  
## origQ predQ freq  
## 1 3 4 1  
## 2 3 5 2  
## 3 3 6 3  
## 4 4 4 5  
## 5 4 5 15  
## 6 4 6 16  
## 7 5 4 4  
## 8 5 5 134  
## 9 5 6 152  
## 10 5 7 3  
## 11 6 4 1  
## 12 6 5 62  
## 13 6 6 347  
## 14 6 7 37  
## 15 7 5 1  
## 16 7 6 116  
## 17 7 7 44  
## 18 8 6 22  
## 19 8 7 13  
## 20 9 6 2  
## [1] "trainErr"  
## [1] 6.0000000000 0.5385400715  
## origQ predQ freq  
## 1 3 5 4  
## 2 3 6 10  
## 3 4 3 1  
## 4 4 4 9  
## 5 4 5 65  
## 6 4 6 50  
## 7 4 7 2  
## 8 5 4 24  
## 9 5 5 489  
## 10 5 6 639  
## 11 5 7 12  
## 12 6 4 3  
## 13 6 5 251  
## 14 6 6 1354  
## 15 6 7 143  
## 16 7 5 18  
## 17 7 6 532  
## 18 7 7 169  
## 19 8 5 1  
## 20 8 6 94  
## 21 8 7 45  
## 22 9 6 3  
## [1] "testErr"  
## [1] 7.0000000000 0.4948979592  
## origQ predQ freq  
## 1 3 4 4  
## 2 3 6 2  
## 3 4 4 6  
## 4 4 5 20  
## 5 4 6 10  
## 6 5 4 3  
## 7 5 5 143  
## 8 5 6 146  
## 9 5 7 1  
## 10 6 4 1  
## 11 6 5 60  
## 12 6 6 337  
## 13 6 7 49  
## 14 7 5 3  
## 15 7 6 106  
## 16 7 7 51  
## 17 7 8 1  
## 18 8 6 21  
## 19 8 7 14  
## 20 9 7 2  
## [1] "trainErr"  
## [1] 7.0000000000 0.5306278714  
## origQ predQ freq  
## 1 3 4 1  
## 2 3 5 4  
## 3 3 6 9  
## 4 4 4 16  
## 5 4 5 71  
## 6 4 6 40  
## 7 5 3 1  
## 8 5 4 25  
## 9 5 5 502  
## 10 5 6 618  
## 11 5 7 18  
## 12 6 0 1  
## 13 6 4 3  
## 14 6 5 271  
## 15 6 6 1309  
## 16 6 7 167  
## 17 7 5 20  
## 18 7 6 482  
## 19 7 7 217  
## 20 8 5 3  
## 21 8 6 86  
## 22 8 7 51  
## 23 9 6 1  
## 24 9 7 2  
## [1] "testErr"  
## [1] 8.000000000 0.537755102  
## origQ predQ freq  
## 1 3 4 1  
## 2 3 5 3  
## 3 3 6 1  
## 4 3 7 1  
## 5 4 4 1  
## 6 4 5 18  
## 7 4 6 17  
## 8 5 4 2  
## 9 5 5 130  
## 10 5 6 158  
## 11 5 7 3  
## 12 6 4 1  
## 13 6 5 66  
## 14 6 6 346  
## 15 6 7 34  
## 16 7 5 5  
## 17 7 6 118  
## 18 7 7 38  
## 19 8 5 1  
## 20 8 6 22  
## 21 8 7 12  
## 22 9 6 2  
## [1] "trainErr"  
## [1] 8.0000000000 0.5518121491  
## origQ predQ freq  
## 1 3 5 5  
## 2 3 6 9  
## 3 4 4 3  
## 4 4 5 67  
## 5 4 6 56  
## 6 4 7 1  
## 7 5 4 3  
## 8 5 5 493  
## 9 5 6 654  
## 10 5 7 14  
## 11 6 4 1  
## 12 6 5 272  
## 13 6 6 1324  
## 14 6 7 154  
## 15 7 5 25  
## 16 7 6 530  
## 17 7 7 164  
## 18 8 5 8  
## 19 8 6 86  
## 20 8 7 46  
## 21 9 6 2  
## 22 9 7 1  
## [1] "testErr"  
## [1] 9.0000000000 0.5173469388  
## origQ predQ freq  
## 1 3 4 1  
## 2 3 5 3  
## 3 3 6 2  
## 4 4 4 1  
## 5 4 5 21  
## 6 4 6 14  
## 7 5 4 2  
## 8 5 5 146  
## 9 5 6 142  
## 10 5 7 3  
## 11 6 4 1  
## 12 6 5 71  
## 13 6 6 339  
## 14 6 7 36  
## 15 7 5 4  
## 16 7 6 115  
## 17 7 7 42  
## 18 8 5 1  
## 19 8 6 21  
## 20 8 7 13  
## 21 9 6 1  
## 22 9 7 1  
## [1] "trainErr"  
## [1] 9.0000000000 0.5385400715  
## origQ predQ freq  
## 1 3 5 4  
## 2 3 6 9  
## 3 3 7 1  
## 4 4 4 6  
## 5 4 5 68  
## 6 4 6 52  
## 7 4 7 1  
## 8 5 4 13  
## 9 5 5 523  
## 10 5 6 611  
## 11 5 7 17  
## 12 6 4 2  
## 13 6 5 269  
## 14 6 6 1332  
## 15 6 7 148  
## 16 7 5 27  
## 17 7 6 523  
## 18 7 7 169  
## 19 8 5 1  
## 20 8 6 91  
## 21 8 7 48  
## 22 9 6 1  
## 23 9 7 2  
## [1] "testErr"  
## [1] 10.0000000000 0.5244897959  
## origQ predQ freq  
## 1 3 4 1  
## 2 3 5 2  
## 3 3 6 3  
## 4 4 4 1  
## 5 4 5 21  
## 6 4 6 14  
## 7 5 4 1  
## 8 5 5 126  
## 9 5 6 163  
## 10 5 7 3  
## 11 6 5 66  
## 12 6 6 345  
## 13 6 7 36  
## 14 7 6 117  
## 15 7 7 43  
## 16 7 8 1  
## 17 8 6 21  
## 18 8 7 14  
## 19 9 6 1  
## 20 9 7 1  
## [1] "trainErr"  
## [1] 10.0000000000 0.5326697295  
## origQ predQ freq  
## 1 3 5 4  
## 2 3 6 9  
## 3 3 7 1  
## 4 4 4 3  
## 5 4 5 64  
## 6 4 6 59  
## 7 4 7 1  
## 8 5 4 5  
## 9 5 5 505  
## 10 5 6 642  
## 11 5 7 12  
## 12 6 4 1  
## 13 6 5 242  
## 14 6 6 1360  
## 15 6 7 148  
## 16 7 5 15  
## 17 7 6 532  
## 18 7 7 172  
## 19 8 5 1  
## 20 8 6 89  
## 21 8 7 50  
## 22 9 6 1  
## 23 9 7 2  
## [1] "testErr"  
## [1] 11.0000000000 0.5204081633  
## origQ predQ freq  
## 1 3 4 2  
## 2 3 5 1  
## 3 3 6 3  
## 4 4 4 3  
## 5 4 5 19  
## 6 4 6 14  
## 7 5 4 3  
## 8 5 5 135  
## 9 5 6 152  
## 10 5 7 3  
## 11 6 4 3  
## 12 6 5 63  
## 13 6 6 348  
## 14 6 7 33  
## 15 7 5 2  
## 16 7 6 119  
## 17 7 7 40  
## 18 8 6 24  
## 19 8 7 11  
## 20 9 6 1  
## 21 9 7 1  
## [1] "trainErr"  
## [1] 11.0000000000 0.5339458908  
## origQ predQ freq  
## 1 3 5 4  
## 2 3 6 9  
## 3 3 7 1  
## 4 4 3 1  
## 5 4 4 5  
## 6 4 5 61  
## 7 4 6 59  
## 8 4 7 1  
## 9 5 4 14  
## 10 5 5 515  
## 11 5 6 624  
## 12 5 7 11  
## 13 6 4 3  
## 14 6 5 261  
## 15 6 6 1361  
## 16 6 7 126  
## 17 7 5 22  
## 18 7 6 530  
## 19 7 7 167  
## 20 8 5 1  
## 21 8 6 93  
## 22 8 7 46  
## 23 9 6 2  
## 24 9 7 1  
## [1] "testErr"  
## [1] 12.0000000000 0.4642857143  
## origQ predQ freq  
## 1 3 4 2  
## 2 3 5 4  
## 3 4 4 9  
## 4 4 5 19  
## 5 4 6 8  
## 6 5 4 2  
## 7 5 5 171  
## 8 5 6 117  
## 9 5 7 3  
## 10 6 5 87  
## 11 6 6 305  
## 12 6 7 55  
## 13 7 5 3  
## 14 7 6 85  
## 15 7 7 71  
## 16 7 8 2  
## 17 8 6 12  
## 18 8 7 21  
## 19 8 8 2  
## 20 9 6 1  
## 21 9 7 1  
## [1] "trainErr"  
## [1] 12.0000000000 0.5229709035  
## origQ predQ freq  
## 1 3 4 1  
## 2 3 5 6  
## 3 3 6 6  
## 4 3 7 1  
## 5 4 3 1  
## 6 4 4 24  
## 7 4 5 71  
## 8 4 6 31  
## 9 5 3 2  
## 10 5 4 39  
## 11 5 5 625  
## 12 5 6 476  
## 13 5 7 22  
## 14 6 4 12  
## 15 6 5 383  
## 16 6 6 1134  
## 17 6 7 221  
## 18 6 8 1  
## 19 7 5 29  
## 20 7 6 413  
## 21 7 7 273  
## 22 7 8 4  
## 23 8 5 4  
## 24 8 6 58  
## 25 8 7 77  
## 26 8 8 1  
## 27 9 6 1  
## 28 9 7 2  
## [1] "testErr"  
## [1] 13.0000000000 0.4663265306  
## origQ predQ freq  
## 1 3 4 2  
## 2 3 5 2  
## 3 3 6 2  
## 4 4 4 8  
## 5 4 5 19  
## 6 4 6 9  
## 7 5 4 2  
## 8 5 5 167  
## 9 5 6 120  
## 10 5 7 4  
## 11 6 5 75  
## 12 6 6 320  
## 13 6 7 52  
## 14 7 5 3  
## 15 7 6 90  
## 16 7 7 68  
## 17 8 6 17  
## 18 8 7 17  
## 19 8 8 1  
## 20 9 7 2  
## [1] "trainErr"  
## [1] 13.0000000000 0.5099540582  
## origQ predQ freq  
## 1 3 4 1  
## 2 3 5 6  
## 3 3 6 7  
## 4 4 3 1  
## 5 4 4 13  
## 6 4 5 74  
## 7 4 6 38  
## 8 4 7 1  
## 9 5 4 18  
## 10 5 5 615  
## 11 5 6 507  
## 12 5 7 24  
## 13 6 4 4  
## 14 6 5 321  
## 15 6 6 1216  
## 16 6 7 210  
## 17 7 5 17  
## 18 7 6 437  
## 19 7 7 265  
## 20 8 5 5  
## 21 8 6 70  
## 22 8 7 65  
## 23 9 6 1  
## 24 9 7 2  
## [1] "testErr"  
## [1] 14.000000000 0.462244898  
## origQ predQ freq  
## 1 3 4 2  
## 2 3 5 3  
## 3 3 6 1  
## 4 4 4 9  
## 5 4 5 20  
## 6 4 6 7  
## 7 5 4 4  
## 8 5 5 157  
## 9 5 6 129  
## 10 5 7 3  
## 11 6 5 76  
## 12 6 6 317  
## 13 6 7 54  
## 14 7 5 2  
## 15 7 6 86  
## 16 7 7 73  
## 17 8 6 10  
## 18 8 7 25  
## 19 9 7 2  
## [1] "trainErr"  
## [1] 14.0000000000 0.5002552323  
## origQ predQ freq  
## 1 3 4 2  
## 2 3 5 6  
## 3 3 6 5  
## 4 3 7 1  
## 5 4 3 3  
## 6 4 4 22  
## 7 4 5 70  
## 8 4 6 32  
## 9 5 3 1  
## 10 5 4 31  
## 11 5 5 600  
## 12 5 6 511  
## 13 5 7 21  
## 14 6 3 2  
## 15 6 4 7  
## 16 6 5 336  
## 17 6 6 1213  
## 18 6 7 193  
## 19 7 5 19  
## 20 7 6 407  
## 21 7 7 293  
## 22 8 5 4  
## 23 8 6 55  
## 24 8 7 81  
## 25 9 6 1  
## 26 9 7 2  
## [1] "testErr"  
## [1] 15.000000000 0.506122449  
## origQ predQ freq  
## 1 3 4 1  
## 2 3 5 2  
## 3 3 6 3  
## 4 4 4 3  
## 5 4 5 20  
## 6 4 6 13  
## 7 5 4 2  
## 8 5 5 148  
## 9 5 6 140  
## 10 5 7 3  
## 11 6 5 69  
## 12 6 6 340  
## 13 6 7 38  
## 14 7 5 1  
## 15 7 6 117  
## 16 7 7 43  
## 17 8 6 22  
## 18 8 7 13  
## 19 9 6 1  
## 20 9 7 1  
## [1] "trainErr"  
## [1] 15.0000000000 0.5347115875  
## origQ predQ freq  
## 1 3 5 4  
## 2 3 6 9  
## 3 3 7 1  
## 4 4 4 7  
## 5 4 5 66  
## 6 4 6 53  
## 7 4 7 1  
## 8 5 4 12  
## 9 5 5 526  
## 10 5 6 610  
## 11 5 7 16  
## 12 6 4 2  
## 13 6 5 262  
## 14 6 6 1329  
## 15 6 7 158  
## 16 7 5 17  
## 17 7 6 530  
## 18 7 7 172  
## 19 8 5 2  
## 20 8 6 88  
## 21 8 7 50  
## 22 9 6 1  
## 23 9 7 2  
## [1] "testErr"  
## [1] 16.0000000000 0.5204081633  
## origQ predQ freq  
## 1 3 4 2  
## 2 3 5 2  
## 3 3 6 2  
## 4 4 4 3  
## 5 4 5 15  
## 6 4 6 18  
## 7 5 4 4  
## 8 5 5 135  
## 9 5 6 152  
## 10 5 7 2  
## 11 6 4 2  
## 12 6 5 54  
## 13 6 6 358  
## 14 6 7 33  
## 15 7 5 1  
## 16 7 6 129  
## 17 7 7 31  
## 18 8 6 25  
## 19 8 7 10  
## 20 9 6 1  
## 21 9 7 1  
## [1] "trainErr"  
## [1] 16.0000000000 0.5490045942  
## origQ predQ freq  
## 1 3 4 1  
## 2 3 5 3  
## 3 3 6 10  
## 4 4 3 1  
## 5 4 4 11  
## 6 4 5 55  
## 7 4 6 59  
## 8 4 7 1  
## 9 5 4 28  
## 10 5 5 444  
## 11 5 6 678  
## 12 5 7 14  
## 13 6 4 7  
## 14 6 5 231  
## 15 6 6 1400  
## 16 6 7 113  
## 17 7 5 12  
## 18 7 6 568  
## 19 7 7 139  
## 20 8 5 1  
## 21 8 6 103  
## 22 8 7 36  
## 23 9 6 2  
## 24 9 7 1

testCopy <- computeResultColumns(nnet,testingData)  
print("testErr")

## [1] "testErr"

print(c(sum(testCopy$error)/nrow(testCopy)))

## [1] 0.5204081633

trainCopy <- computeResultColumns(nnet,trainingData)  
print("trainErr")

## [1] "trainErr"

print(c(sum(trainCopy$error)/nrow(trainCopy)))

## [1] 0.5490045942

count(testCopy,c("origQ","predQ"))

## origQ predQ freq  
## 1 3 4 2  
## 2 3 5 2  
## 3 3 6 2  
## 4 4 4 3  
## 5 4 5 15  
## 6 4 6 18  
## 7 5 4 4  
## 8 5 5 135  
## 9 5 6 152  
## 10 5 7 2  
## 11 6 4 2  
## 12 6 5 54  
## 13 6 6 358  
## 14 6 7 33  
## 15 7 5 1  
## 16 7 6 129  
## 17 7 7 31  
## 18 8 6 25  
## 19 8 7 10  
## 20 9 6 1  
## 21 9 7 1

## It looks like no matter how many hidden nodes/layers we can never predict   
## a 3, 8 or 9. That is a problem!  
##  
## Let’s see if an SVM does better

|  |
| --- |
|  |
| |  | | --- | | ## Unscale Q first scaledWhite$Q <- 3+6\*scaledWhite$Q count(scaledWhite$Q)  ## x freq ## 1 3 20 ## 2 4 163 ## 3 5 1457 ## 4 6 2198 ## 5 7 880 ## 6 8 175 ## 7 9 5  set.seed(2) train <- sample(1:nrow(scaledWhite),nrow(scaledWhite)\*(8/10)) test<- -train  trainingData<-scaledWhite[train,] testingData<-scaledWhite[test,]   ## We’ll use as.factor so that we can try and predict 7 classes ## don’t know if it will work  formula <- as.factor(Q)~a1+a2+a3+a4+a5+a6+a7+a8+a9+a10+a11 trainingData<-scaledWhite[train,] testingData<-scaledWhite[test,]  names(testingData)  ## [1] "a1" "a2" "a3" "a4" "a5" "a6" "a7" "a8" "a9" "a10" "a11" ## [12] "Q"  svm\_model <- svm(formula, data=trainingData) summary(svm\_model)  ##  ## Call: ## svm(formula = formula, data = trainingData) ##  ##  ## Parameters: ## SVM-Type: C-classification  ## SVM-Kernel: radial  ## cost: 1  ## gamma: 0.09090909  ##  ## Number of Support Vectors: 3521 ##  ## ( 127 1021 1525 140 691 14 3 ) ##  ##  ## Number of Classes: 7  ##  ## Levels:  ## 3 4 5 6 7 8 9  testingData$result <- predict(svm\_model,testingData) trainingData$result <- predict(svm\_model,trainingData)  count(testingData,c("Q","result"))  ## Q result freq ## 1 3 5 3 ## 2 3 6 3 ## 3 4 4 1 ## 4 4 5 27 ## 5 4 6 8 ## 6 5 5 169 ## 7 5 6 123 ## 8 5 7 1 ## 9 6 5 76 ## 10 6 6 347 ## 11 6 7 24 ## 12 7 5 4 ## 13 7 6 111 ## 14 7 7 46 ## 15 8 6 25 ## 16 8 7 10 ## 17 9 6 1 ## 18 9 7 1  ## Certainly no better than the neural net. Arguably worse ## No 3’s. Only one 4, and no 8 or 9.  ## Let's look at a histgram hist(scaledWhite$Q)  count(scaledWhite$Q)  ## x freq ## 1 3 20 ## 2 4 163 ## 3 5 1457 ## 4 6 2198 ## 5 7 880 ## 6 8 175 ## 7 9 5  ## The histogram is telling us what to do ## 3 and 4 are poor wines, but because they are few we should combine them ## same for 8 and 9 ## we will hope that the model will do better with 5 classes instead of 7  scaledWhite$Q2 <- apply(scaledWhite,MARGIN=1,function(x){  if (x["Q"] <= 4) return("poor")  if (x["Q"] <= 5) return("belowAvg")  if (x["Q"] <= 6) return("Avg")  if (x["Q"] <= 7) return("aboveAvg")  return("veryGood") })  count(scaledWhite,c("Q","Q2"))  ## Q Q2 freq ## 1 3 poor 20 ## 2 4 poor 163 ## 3 5 belowAvg 1457 ## 4 6 Avg 2198 ## 5 7 aboveAvg 880 ## 6 8 veryGood 175 ## 7 9 veryGood 5  ##Let's Try again formula <- as.factor(Q2)~a1+a2+a3+a4+a5+a6+a7+a8+a9+a10+a11 trainingData<-scaledWhite[train,] testingData<-scaledWhite[test,]  svm\_model <- svm(formula, data=trainingData) summary(svm\_model)  ##  ## Call: ## svm(formula = formula, data = trainingData) ##  ##  ## Parameters: ## SVM-Type: C-classification  ## SVM-Kernel: radial  ## cost: 1  ## gamma: 0.09090909  ##  ## Number of Support Vectors: 3519 ##  ## ( 141 1017 1526 143 692 ) ##  ##  ## Number of Classes: 5  ##  ## Levels:  ## aboveAvg Avg belowAvg poor veryGood  testingData$result <- predict(svm\_model,testingData) trainingData$result <- predict(svm\_model,trainingData)  count(testingData,c("Q2","result"))  ## Q2 result freq ## 1 aboveAvg aboveAvg 46 ## 2 aboveAvg Avg 111 ## 3 aboveAvg belowAvg 4 ## 4 Avg aboveAvg 24 ## 5 Avg Avg 347 ## 6 Avg belowAvg 76 ## 7 belowAvg aboveAvg 1 ## 8 belowAvg Avg 123 ## 9 belowAvg belowAvg 169 ## 10 poor Avg 11 ## 11 poor belowAvg 30 ## 12 poor poor 1 ## 13 veryGood aboveAvg 11 ## 14 veryGood Avg 26  write.csv(testingData,file="svm.csv") ## The SVM is still incapable of predicting the VeryGood wines ## Let's see if a neural net does better scaledWhite$Q3<-0 scaledWhite$Q3 <- apply(scaledWhite,MARGIN=1,function(x){  if (x["Q"] <= 4) return(0)  if (x["Q"] <= 5) return(.25)  if (x["Q"] <= 6) return(.5)  if (x["Q"] <= 7) return(.75)  return(1.0) })   trainingData<-scaledWhite[train,] testingData<-scaledWhite[test,] formula<-Q3 ~ a1 + a2 + a3 + a4 + a5 + a6 + a7 + a8 + a9 + a10 + a11 nnet<- neuralnet(formula, trainingData, hidden = c(12,6), threshold = 0.1) result <- compute(nnet, testingData[,1:11]) testingData$result <- sapply(result$net.result, function(b) {  return(b) }) write.csv(testingData,file="nn2.csv")  ## Look at some graphs | |