Artificial Neural Net to Classify Simulated Spherical Data

Contents

Introduction

I explore the effectiveness of single layer Neural Nets in classifying data with a spherical class boundary, varying the number of nodes, sample size and noise. The goal is to create a simple example where the effect of each model adjustment is easy to understand. We'll use a logistic activation function, but with one hidden layer, it's still useful to think of each layer as adding a hyperplane decision boundary. A single node would divide the plane in two, 4 nodes could fully enclose the 3-dimensional class boundary and additional nodes would tighten that boundary. For a similar excercise in building intuition about neural net architecture see "ANN - polynomial fit animation".

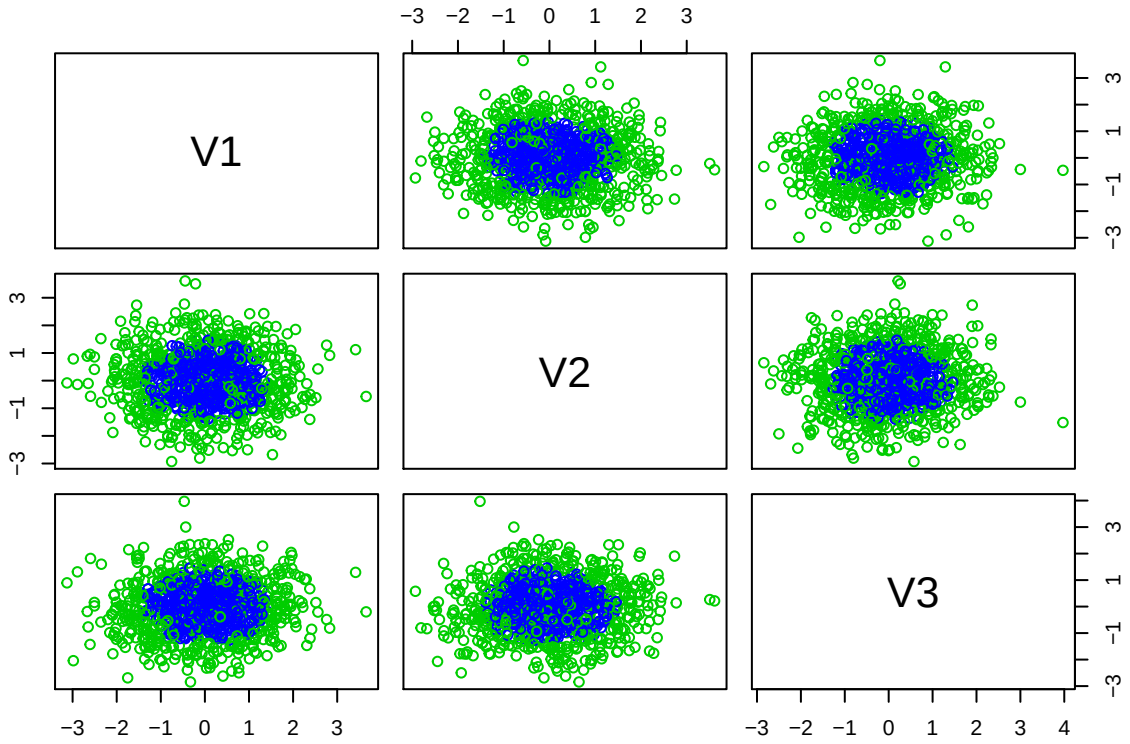
Simulate Data with a Spherical Class Boundary

I start by simulating 1000 observations with 3 covariates then assign all points within a radius of 1.5 units from zero to class "1".

```
SphericalDataset <- function(nObs=1000,nPreds=3,nNulls=0) {
 xyTmp=matrix(rnorm((nPreds+nNulls)*nObs),ncol=nPreds+nNulls)
 clTmp=matrix(0,nObs,ncol=1)
 dataTmp=data.frame()
 #if magnitue of vector of first nPreds variables is less than or equal to 1.5 then class=1 otherwise 0
 for (iTry in (1:nObs)) {
 if (sum(xyTmp[iTry,1:nPreds]^2) <= 1.5^2) {
   clTmp[iTry]=1
 }
 }
 dataTmp=cbind(xyTmp,clTmp)
dataTmp=as.data.frame(dataTmp)
colnames(dataTmp)[nPreds+nNulls+1]="Class"
 return(dataTmp)
 }

Sim.1k.3.0=SphericalDataset()
sum(Sim.1k.3.0$Class) #confirm roughly half in each class
```

```
## [1] 477
```

```
pairs(Sim.1k.3.0[,1:3],col=Sim.1k.3.0$Class+3)
```

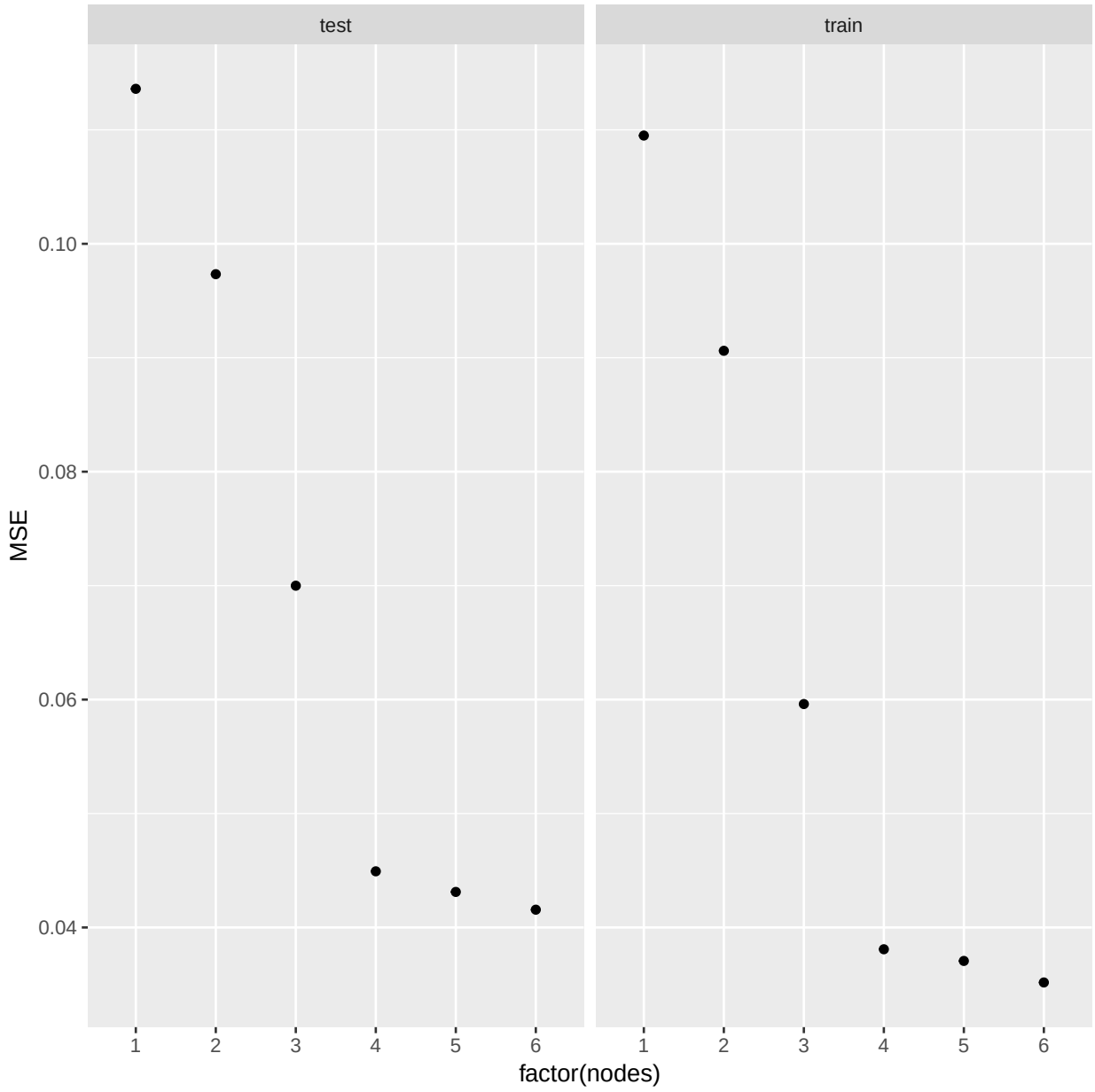


As mentioned, In 3d space, at least 4 planes are needed to completely enclose an object. With a single hidden layer, I would need at least 4 nodes to completely enclose the inner class and many more to exclusively enclose it.

Single Layer Neural Network

Now I create neural networks with different number of nodes in a single hidden layer and plot the Mean Squared Error.

```
Sim.10k.3.0=SphericalDataset(nObs=10000)
nnerr.1k.3.0=data.frame()

for (nodes in 1:6) {
nn.Tmp <- neuralnet(Class~V1+V2+V3,data=Sim.1k.3.0,err.fct="sse",hidden=nodes)
MSE.Train=sum(nn.Tmp$err.fct(Sim.1k.3.0$Class,nn.Tmp$net.result[[1]]))/1000
nnPred <- compute(nn.Tmp,Sim.10k.3.0[,1:3])$net.result[,1]
MSE.Test <-sum(nn.Tmp$err.fct(Sim.10k.3.0$Class,nnPred))/10000

row.Tmp=data.frame(nodes=nodes,MSE=c(MSE.Train,MSE.Test),type=c("train","test"))
nnerr.1k.3.0=rbind(nnerr.1k.3.0,row.Tmp)
}
ggplot(nnerr.1k.3.0,aes(x=factor(nodes),y=MSE))+geom_point()+facet_wrap(~type)
```



The model error decreases as the complexity increases up to 4 nodes. Once the class boundary is enclosed, increasing the number of nodes either either does not improve performance or improves it very slightly. At some point the model will merely overfit the available training data.

Adding Noise and Changing Sample Size

Now I'll explore the overfitting phenomenon more by varying the number of nodes, number of observations and number of non predictive variables.

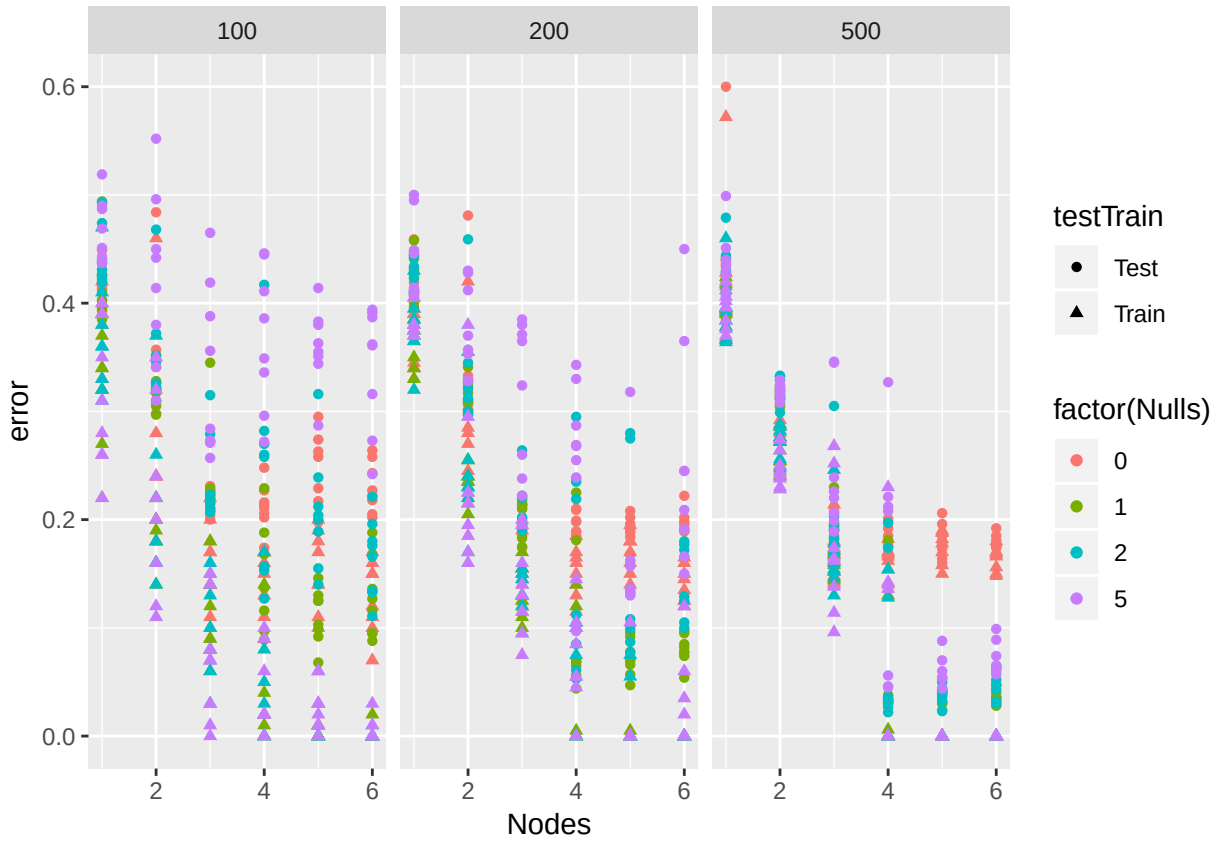
```
dfTmp<-data.frame()
form=c("","Class~V1+V2","Class~V1+V2+V3","Class~V1+V2+V3+V4","","","Class~V1+V2+V3+V4+V5+V6+V7")
for (obs in c(100,200,500)) {
  for (nulls in c(0,1,2,5)) {
    for (nodes in 1:6) {
```

```
    attempts=0
        for (iTry in 1:8) {

       SimTmp = SphericalDataset(nObs=obs,nNulls=nulls,nPreds=3)
       TestTmp = SphericalDataset(nObs=1000,nNulls=nulls,nPreds=3)

       vars= nulls+2
       nnTmp=neuralnet(form[vars],data=SimTmp,err.fct="ce",hidden=nodes,linear.output = FALSE, thresho

      #if neuralnet function fails to converge then reiterate up to 4 times per model
       if (length(attributes(nnTmp)$names) <13) {
           attempts=attempts+1
           if (attempts<5) {
            iTry=iTry-1
           }

           next
       }

     nnTestPred <- compute(nnTmp,TestTmp[,1:vars])$net.result[,1]
     tblTestTmp <- table(nnTestPred>0.5,TestTmp[,"Class"])
     errTestTmp <- 1 - sum(diag(tblTestTmp))/sum(tblTestTmp)

     tblTrainTmp <- table(nnTmp$net.result[[1]][,1]>0.5,SimTmp[,"Class"])
     errTrainTmp <- 1 - sum(diag(tblTrainTmp))/sum(tblTrainTmp)

     dfTmp <- rbind(dfTmp,data.frame(Obs=obs,Nulls=nulls,Nodes=nodes,error=c(errTestTmp,errTrainTmp),t

     }
    }
  }
}
```

```
## Warning: Algorithm did not converge in 1 of 1 repetition(s) within the stepmax.
```

```
## Warning: Algorithm did not converge in 1 of 1 repetition(s) within the stepmax.
```

```
ggplot(dfTmp,aes(x=Nodes,y=error,shape=testTrain,colour=factor(Nulls)))+geom_point()+facet_wrap(~Obs)
```



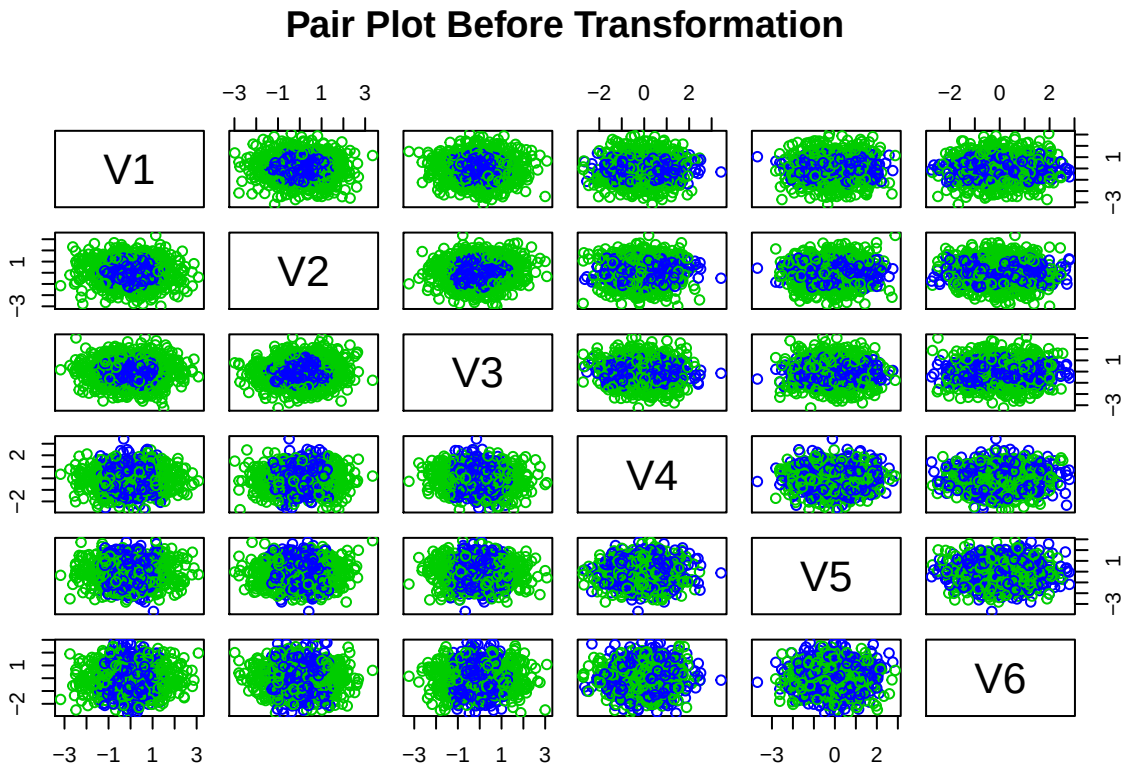
As expected, more noise and less data leads to more overfitting, particularly in the models with higher complexity. Higher sample sizes produce much more stable outcomes (test and train results are more similar). The models get more accurate as the number of nodes increases (although we don't see significant change after 3 nodes), and increasing the number of non predictive variables creates some of the best performing training errors but the worst performing test errors.

Adding Rotations

Finally I apply a linear transformation so each of the six variables has some signal and some noise. While this makes the class boundary harder for us to visualize in pair plots, the model performance should be similar.

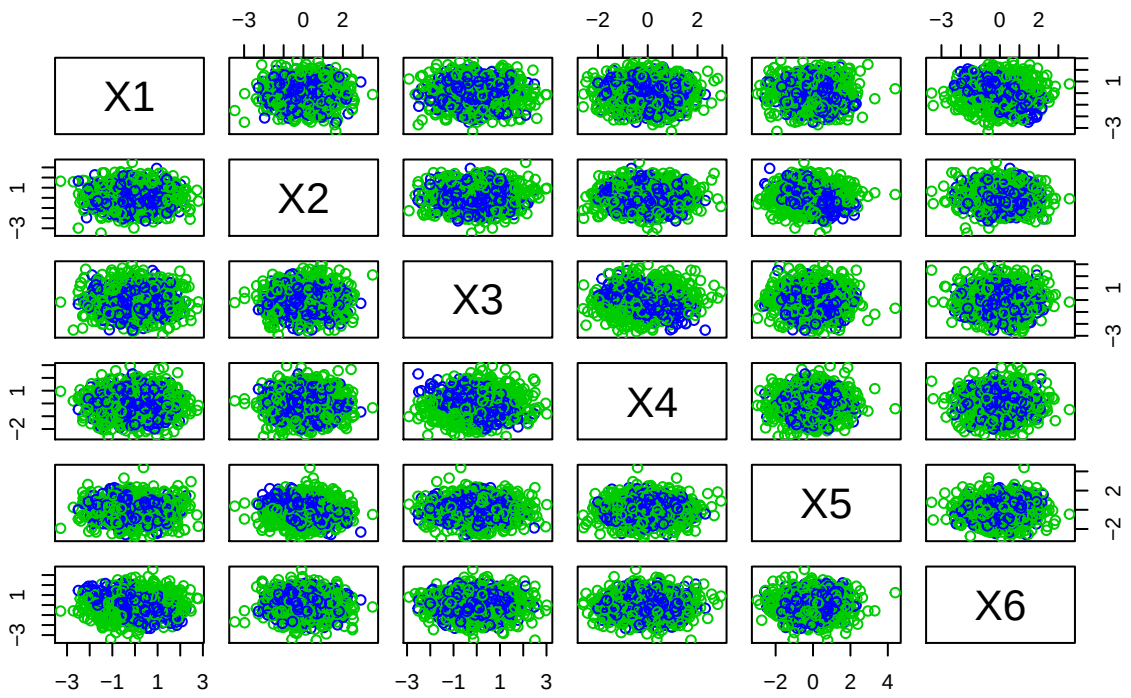
```
Rotate <- function(df) {
  #make rotation matrices
  M.I<-diag(6)
  M.16<-M.I;M.25<-M.I;M.34<-M.I
  M.16[1,1]<-cos(pi/4);M.25[2,2]<-cos(pi/4);M.34[3,3]<-cos(pi/4)
  M.16[1,6]<- -sin(pi/4);M.25[2,5]<- -sin(pi/4);M.34[3,4]<- -sin(pi/4)
  M.16[6,1]<-sin(pi/4);M.25[5,2]<-sin(pi/4);M.34[4,3]<-sin(pi/4)
  M.16[6,6]<-cos(pi/4);M.25[5,5]<-cos(pi/4);M.34[4,4]<-cos(pi/4)

  #Create rotation matrix
  Rot<-M.16%*%M.25%*%M.34

  #Apply to dataframe
  M<-data.matrix(df[1:6]) %*% t(Rot)
  df.new<-cbind(data.frame(M),Class=df$Class)
```

```
  return(df.new)
}

#visualize rotated data in pair plots
Sim.1k.3.3<-SphericalDataset(nObs=1000,nNulls=3,nPreds=3)
Sim.1k.3.3.r<-Rotate(Sim.1k.3.3)
pairs(Sim.1k.3.3[,1:6],col=Sim.1k.3.3$Class+3,main = "Pair Plot Before Transformation")
```



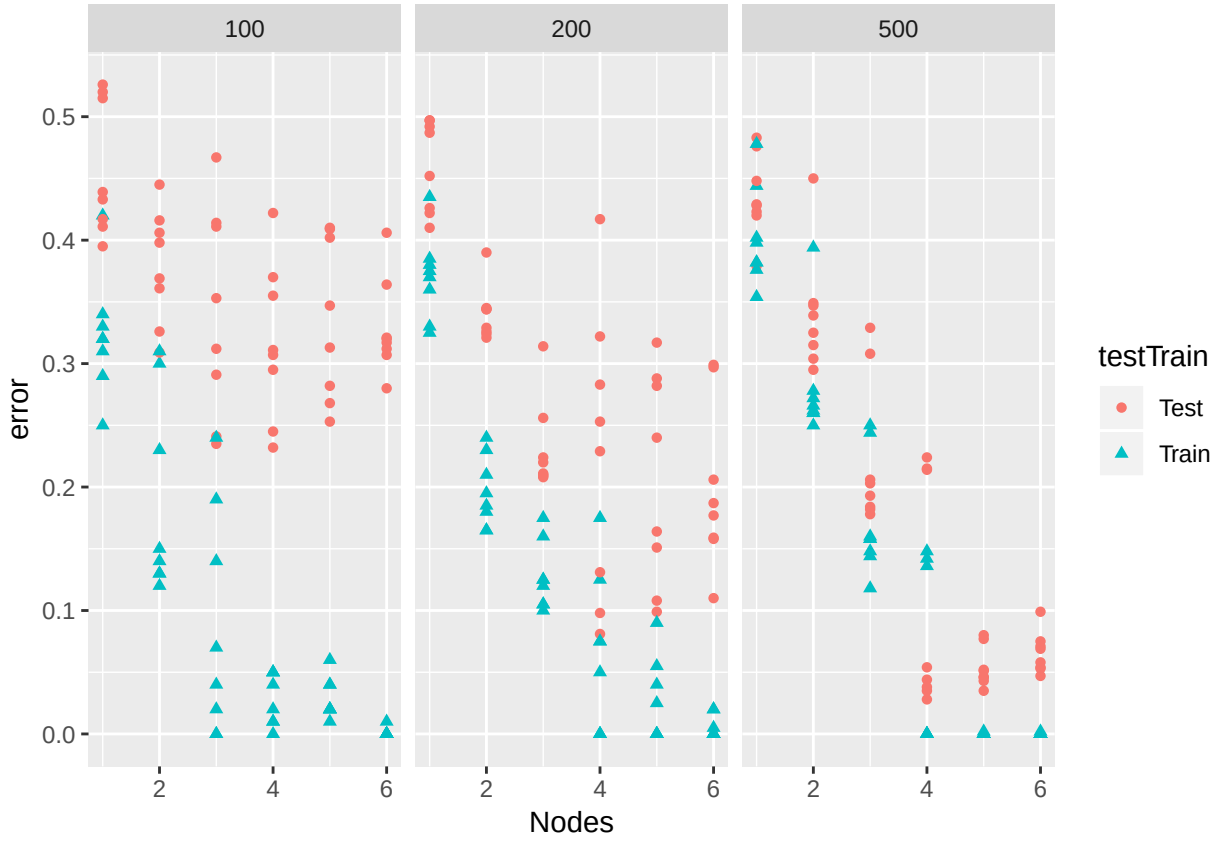

```
pairs(Sim.1k.3.3.r[,1:6],col=Sim.1k.3.3.r$Class+3, main = "Pair Plot After Transformation")
```

Pair Plot After Transformation




```
dfTmp2<-data.frame()
form="Class~X1+X2+X3+X4+X5+X6"
for (obs in c(100,200,500)) {
  for (nodes in 1:6) {
    attempts=0
    for (iTry in 1:8) {

      SimTmp = Rotate(SphericalDataset(nObs=obs,nNulls=3,nPreds=3))
      TestTmp = Rotate(SphericalDataset(nObs=1000,nNulls=3,nPreds=3))



      nnTmp=neuralnet(form,data=SimTmp,err.fct="ce",hidden=nodes,linear.output = FALSE, threshold=.15)

      #if neuralnet function fails to converge then reiterate up to 4 times per model
      if (length(attributes(nnTmp)$names) <13) {
        attempts=attempts+1
        if (attempts<5) {
          iTry=iTry-1
          }
        next
        }

      nnTestPred <- compute(nnTmp,TestTmp[,1:6])$net.result[,1]
      tblTestTmp <- table(nnTestPred>0.5,TestTmp[,"Class"])
      errTestTmp <- 1 - sum(diag(tblTestTmp))/sum(tblTestTmp)
```

```
      tblTrainTmp <- table(nnTmp$net.result[[1]][,1]>0.5,SimTmp[,"Class"])
      errTrainTmp <- 1 - sum(diag(tblTrainTmp))/sum(tblTrainTmp)

      dfTmp2 <- rbind(dfTmp2,data.frame(Obs=obs,Nodes=nodes,error=c(errTestTmp,errTrainTmp),testTrain=c

    }
  }
}

ggplot(dfTmp2,aes(x=Nodes,y=error,shape=testTrain,colour=testTrain))+geom_point()+facet_wrap(~Obs)
```



Although the class sphere is now partially in all 6 dimensions, the Neural Networks are still able to enclose it with 4 nodes.