MLSheet4

Oliver Cobb

19/04/2018

I have used the following libraries: ‘rpart’,‘rpart.plot’,‘randomForest’,‘nnet’ and ‘neuralnet’.

Question 4

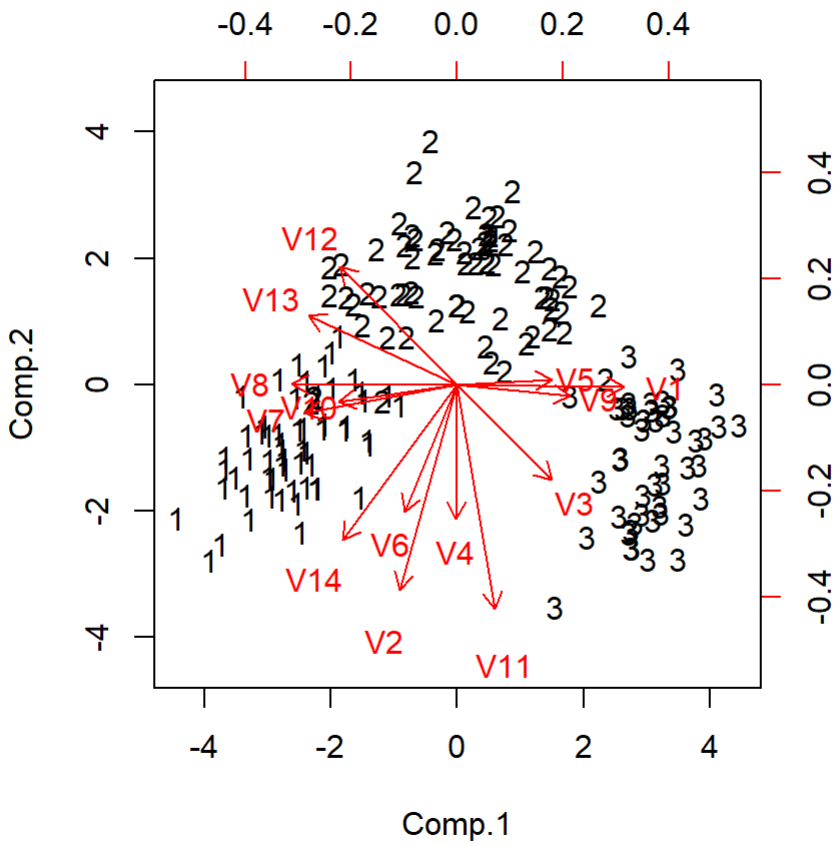
Load in the data and have a look.

```
wine <- read.table("https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wi
ne.data", sep=",")
head(wine)
```

```
##   V1    V2   V3   V4   V5  V6   V7   V8   V9  V10  V11  V12  V13  V14
## 1  1 14.23 1.71 2.43 15.6 127 2.80 3.06 0.28 2.29 5.64 1.04 3.92 1065
## 2  1 13.20 1.78 2.14 11.2 100 2.65 2.76 0.26 1.28 4.38 1.05 3.40 1050
## 3  1 13.16 2.36 2.67 18.6 101 2.80 3.24 0.30 2.81 5.68 1.03 3.17 1185
## 4  1 14.37 1.95 2.50 16.8 113 3.85 3.49 0.24 2.18 7.80 0.86 3.45 1480
## 5  1 13.24 2.59 2.87 21.0 118 2.80 2.69 0.39 1.82 4.32 1.04 2.93  735
## 6  1 14.20 1.76 2.45 15.2 112 3.27 3.39 0.34 1.97 6.75 1.05 2.85 1450
```

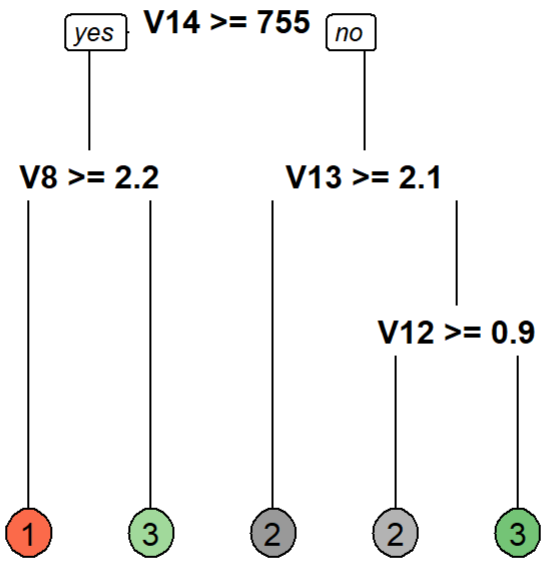
Form the loadings matrix and plot a biplot

```
wine.pca <- princomp(wine, cor=T)
biplot(wine.pca, scale=0, xlabs=as.numeric(wine$V1))
```

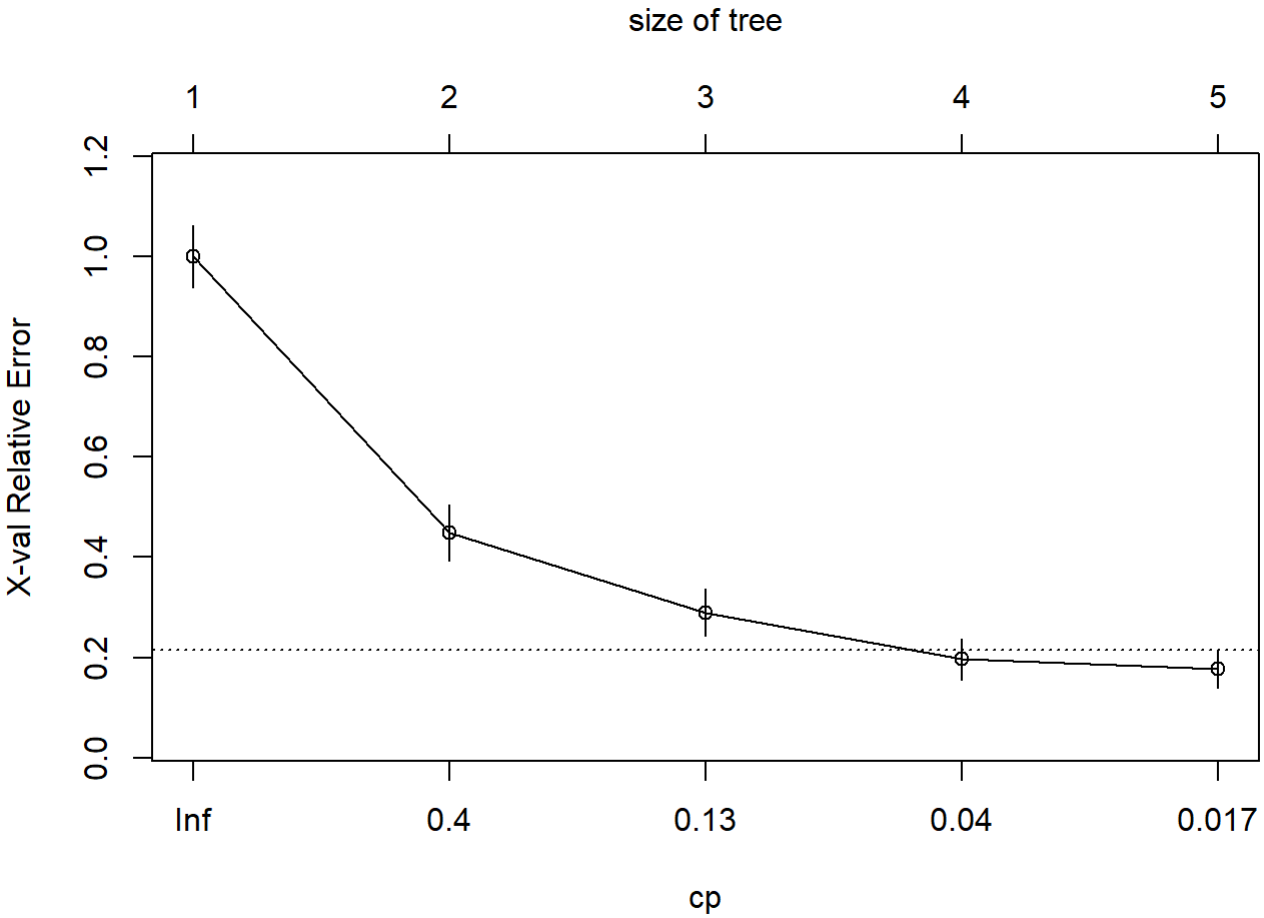


First build and plot a decision tree. Then from plotcp we can see that the cross validation error is minimised at a tree depth of 4. Then from printcp we can see the misclassification error is approx 13%. (‘root nood error’ x ‘final xerror’ x 100%)

```
tree <- rpart(as.factor(V1)~., data=wine)
rpart.plot(tree,extra=0, type=0)
```




```
plotcp(tree)
```




```
printcp(tree)
```

```
##
## Classification tree:
## rpart(formula = as.factor(V1) ~ ., data = wine)
##
## Variables actually used in tree construction:
## [1] V12 V13 V14 V8
##
## Root node error: 107/178 = 0.60112
##
## n= 178
##
##         CP nsplit rel error  xerror     xstd
## 1 0.495327      0   1.00000 1.00000 0.061056
## 2 0.317757      1   0.50467 0.44860 0.055335
## 3 0.056075      2   0.18692 0.28972 0.047287
## 4 0.028037      3   0.13084 0.19626 0.040222
## 5 0.010000      4   0.10280 0.17757 0.038502
```

Fit a random forest classifier. Usually gives an OOB cross validation error of ~1.7%.

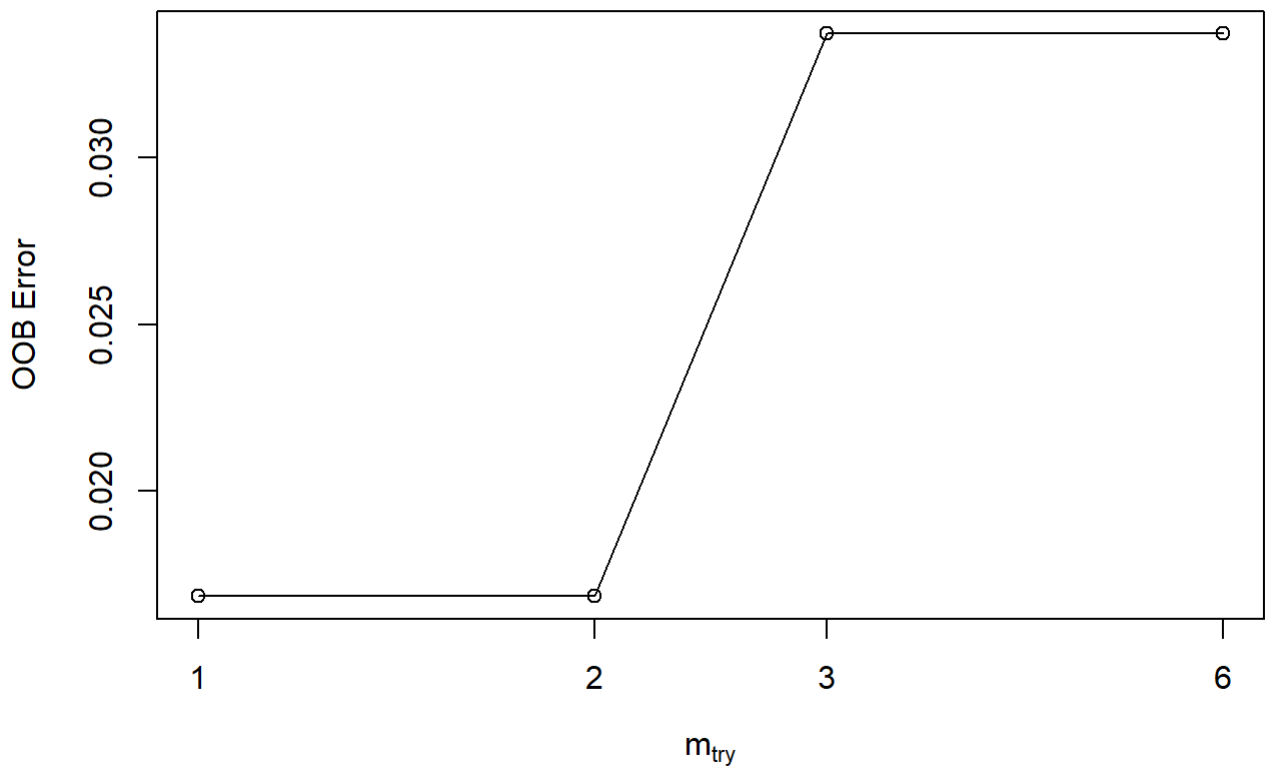
```
rf <- randomForest(wine[,2:14],as.factor(wine[,1]),importance=TRUE)
print(rf)
```

```
##
## Call:
##  randomForest(x = wine[, 2:14], y = as.factor(wine[, 1]), importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 1.69%
## Confusion matrix:
##    1  2  3 class.error
## 1 59  0  0  0.00000000
## 2  1 68  2  0.04225352
## 3  0  0 48  0.00000000
```

Find an optimal value for mtry. The graph of mtry against OOB error varies every time so won't read too much into it.

```
tuneRF(wine[,2:14],as.factor(wine[,1]),importance=TRUE)
```

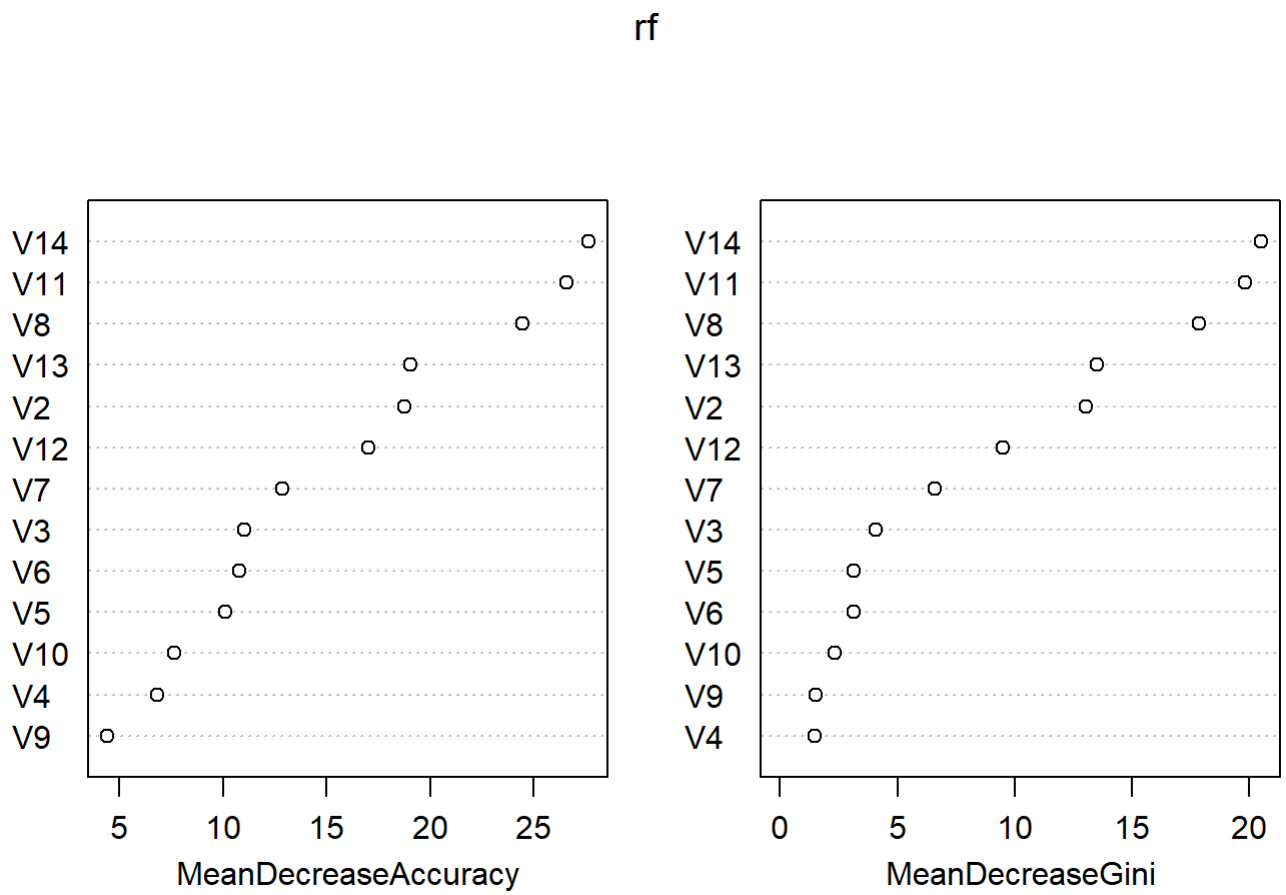
```
## mtry = 3  OOB error = 3.37%
## Searching left ...
## mtry = 2     OOB error = 1.69%
## 0.5 0.05
## mtry = 1     OOB error = 1.69%
## 0 0.05
## Searching right ...
## mtry = 6     OOB error = 3.37%
## -1 0.05
```



```
##       mtry   OOBError
## 1.OOB    1 0.01685393
## 2.OOB    2 0.01685393
## 3.OOB    3 0.03370787
## 6.OOB    6 0.03370787
```

Plotting the importance of each variable we see that V14, V8 and V11 are the most important.

```
varImpPlot(rf)
```



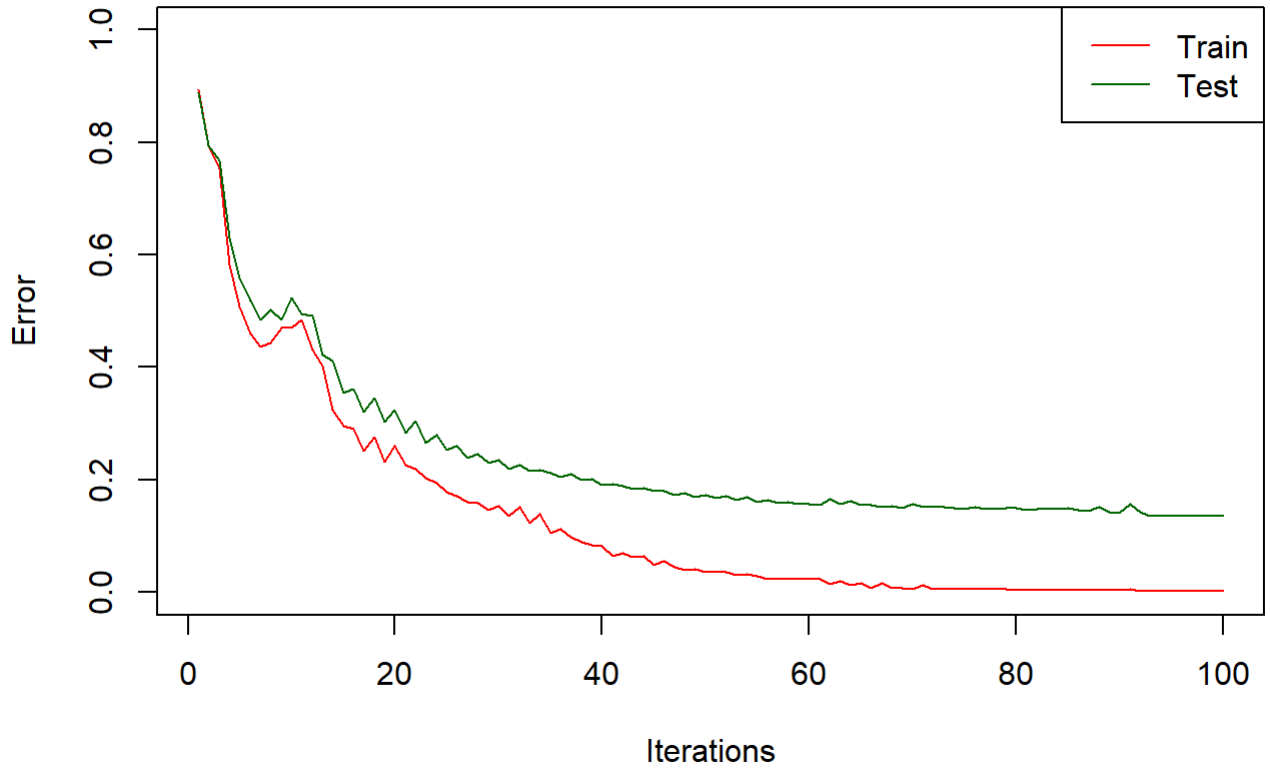
Question 5

Load in the data and form the training and testing data frames. I used 1/4 of the data rather than 1/10th.

```
testx <- read.table("http://www.stats.ox.ac.uk/~palamara/teaching/SML18/usps_testx.da
ta")
testy <- read.table("http://www.stats.ox.ac.uk/~palamara/teaching/SML18/usps_testy.da
ta")
trainx <- read.table("http://www.stats.ox.ac.uk/~palamara/teaching/SML18/usps_trainx.
data")
trainy <- read.table("http://www.stats.ox.ac.uk/~palamara/teaching/SML18/usps_trainy.
data")

################

ntr<- dim(trainx)[1]
trainx <- trainx / 256
testx  <- testx  / 256
trainy <- as.factor(trainy[,1])
testy  <- as.factor(testy[,1])

subsample_idx <- seq(1,ntr,by=4)
trainx <- trainx[subsample_idx,]
trainy <- trainy[subsample_idx]
```

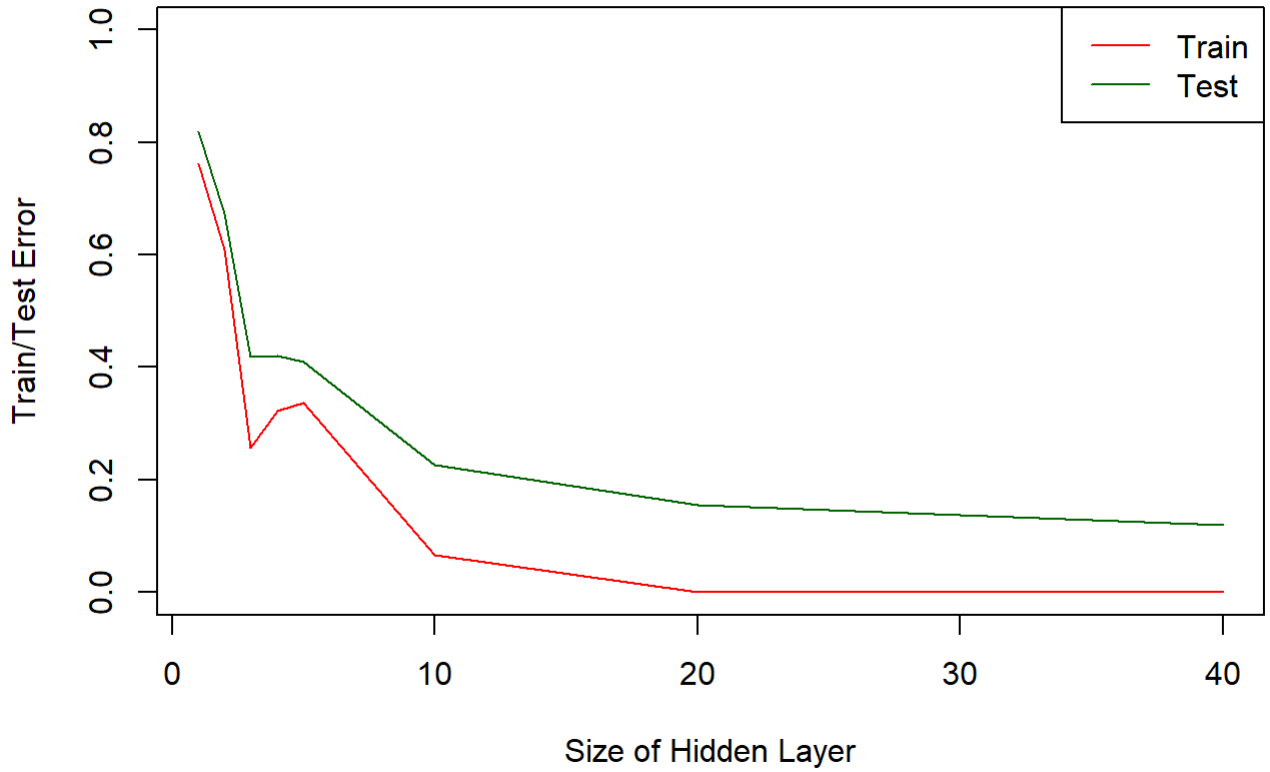
Plot a the test error of a neural network against the number of iterations. I have surpressed the printed output after each iteration to save space.

```
#regularization parameter
decay=0.01

#train a net for 10 iterations -- softmax=TRUE for log-linear model
net <- nnet(trainx,class.ind(trainy),size=10,softmax=TRUE,
            MaxNWts=10000,maxit=1,decay=decay, trace=FALSE)
trainp <- predict(net,trainx,type="class")
testp  <- predict(net,testx ,type="class")
trainerrs <- sum(trainy!=trainp)/length(trainp)
testerrs  <- sum(testy!=testp)/length(testp)
#print(paste("..train error after 1 iteration:",trainerr))
#print(paste("..test error after 1 iteration:",testerr))

for (t in (2:100)) {
  net <- nnet(trainx,class.ind(trainy),size=10,softmax=TRUE,
              MaxNWts=10000,maxit=1,decay=decay,Wts=net$wts, trace=FALSE)
  trainp <- predict(net,trainx,type="class")
  testp  <- predict(net,testx ,type="class")
  trainerr <- sum(trainy!=trainp)/length(trainp)
  testerr  <- sum(testy!=testp)/length(testp)
  #print(paste("..train error after ",t," iterations:",trainerr))
  #print(paste("..test error after ",t," iterations:",testerr))
  trainerrs <- append(trainerrs, trainerr)
  testerrs <- append(testerrs, testerr)
}
plot(1:100,trainerrs, ylim=c(0,1), type="l", col="red", xlab="Iterations", ylab="Erro
r")
lines(1:100, testerrs, col="darkgreen")
legend("topright", legend=c("Train","Test"), lty=1, col=c("red","darkgreen"))
```



Plot the training and testing error as a function of the number of units in the hidden layer. We see that the misclassification rate for a single layer neural net plateuaus at around 15%.

```
sizes <- c(1,2,3,4,5,10,20,40)
trainerr2 <- c()
testerr2 <- c()
for (s in sizes) {

net <- nnet(trainx,class.ind(trainy),size=s,softmax=TRUE,
              MaxNWts=15000,maxit=25,decay=decay, trace=FALSE)
  trainp <- predict(net,trainx,type="class")
  testp  <- predict(net,testx ,type="class")
  trainerr2 <- append(trainerr2,sum(trainy!=trainp)/length(trainp))
  testerr2  <- append(testerr2,sum(testy!=testp)/length(testp))

}

plot(sizes,trainerr2, type="l", col="red", ylab ="Train/Test Error", xlab="Size of Hi
dden Layer",ylim=c(0,1))
lines(sizes,testerr2, col="darkgreen")
legend("topright", legend=c("Train","Test"), lty=1, col=c("red","darkgreen"))
```



Investigate whether increasing the number of layers imporves classification performance. I found that it didn't improve upon the 15% misclassification rate of the single layer net.

```
data = cbind(class.ind(trainy),trainx)
names(data)[1:10] = paste0("label",names(data[1:10]))
fml <- as.formula(paste(paste(paste0('label',0:9),collapse=" + "),
                        ' ~ ' ,paste(paste0("V",1:256),collapse='+')))


for (m in c(5,10,20,40,60)) {
  net <- neuralnet(fml, data, hidden=c(40,m), err.fct="ce", linear.output=FALSE)
  trainp<- as.factor(apply(neuralnet::compute(net,trainx)$net.result,1,which.max)-1)
  testp  <- as.factor(apply(neuralnet::compute(net,testx)$net.result,1,which.max)-1)
  trainerr <- sum(trainy!=trainp)/length(trainp)
  testerr <- sum(testy!=testp)/length(testp)
  print(paste("Test error with ", m," units in the second layer is: ", testerr))}
```

```
## [1] "Test error with  5  units in the second layer is:  0.345"
## [1] "Test error with  10  units in the second layer is:  0.178"
## [1] "Test error with  20  units in the second layer is:  0.1835"
## [1] "Test error with  40  units in the second layer is:  0.152"
## [1] "Test error with  60  units in the second layer is:  0.159"
```

```
for (m in c(5,10,20,40,60)) {
  net <- neuralnet(fml, data, hidden=c(40,20,m), err.fct="ce", linear.output=FALSE)
  trainp<- as.factor(apply(neuralnet::compute(net,trainx)$net.result,1,which.max)-1)
  testp  <- as.factor(apply(neuralnet::compute(net,testx)$net.result,1,which.max)-1)
  trainerr <- sum(trainy!=trainp)/length(trainp)
  testerr <- sum(testy!=testp)/length(testp)
  print(paste("Test error with ", m," units in the third layer is: ", testerr))}
```

```
## [1] "Test error with  5  units in the third layer is:  0.281"
## [1] "Test error with  10  units in the third layer is:  0.198"
## [1] "Test error with  20  units in the third layer is:  0.168"
## [1] "Test error with  40  units in the third layer is:  0.1675"
## [1] "Test error with  60  units in the third layer is:  0.1685"
```