**DEPARTMENT OF STATISTICS**

**STATS 784** Data Mining

Assignment 2 Model answer

*Q1. Write an R function to evaluate the neural network function, given a vector of inputs and the weights. You can input the weights in any way that seems convenient. Include a worked example in your answer and supply me with a machine–readable version of your code.*

Here is a sample function, which uses the data structure employed in the nnet function. It will accept a data frame as an input for prediction.

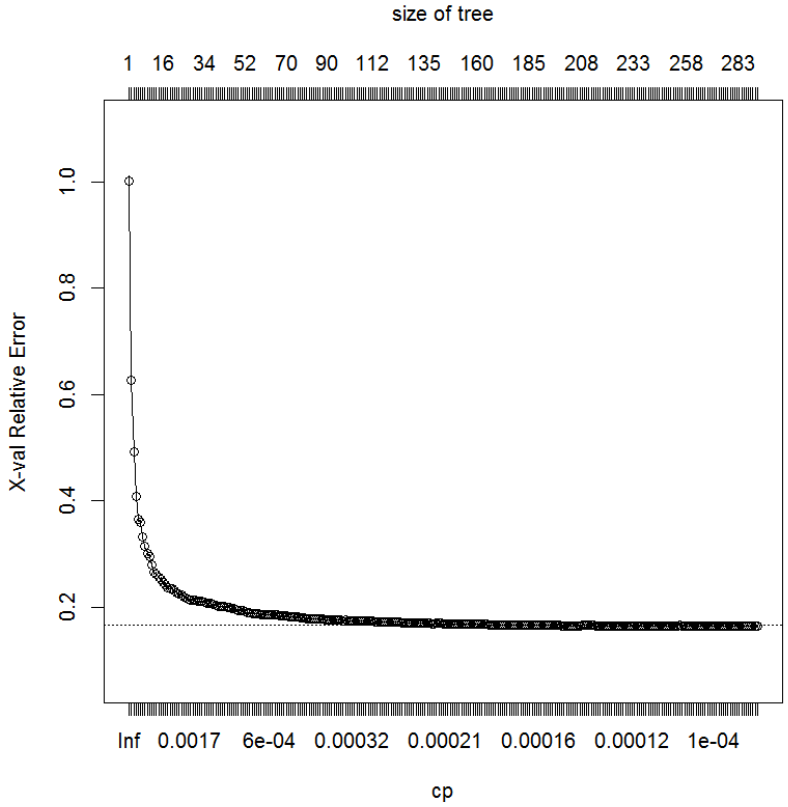
```
library(nnet)
n=100
h = 4
p=5
nNew=2
y=rnorm(n)
X = matrix(rnorm(n*p),n,p)
mydata=data.frame(X,y)
x = matrix(rnorm(nNew*p),nNew,p)

# fit the neural network
nnet.fit = nnet(y~., data=mydata,size=h, linout=TRUE)
# do the prediction
predict(nnet.fit,newdata=data.frame(X[1:2,]))
         [,1]
1 -0.08167895
2 -1.30393404


# now write our own function to do predictions

myNnet = function(nnet.fit, newdata){
# arguments as for predict
p=nnet.fit$n[1]
h=nnet.fit$n[2]
x=as.matrix(newdata)
A = matrix(nnet.fit$wts[1:((p+1)*h)],h, p+1, byrow=TRUE)
a = nnet.fit$wts[((p+1)*h+1):length(nnet.fit$wts)]
sigma = function(x) 1/(1+exp(-x))
as.vector(t(rbind(1,sigma(A%*%t(cbind(1,x)))))%*%a)
}


# use as before
myNnetV2(nnet.fit,newdata=data.frame(X[1:2,]))
[1] -0.08167895 -1.30393404
```

[10 marks]

Q2. *Repeat Question 2 of the last assignment using trees and neural networks in place of the linear model you used in Assignment 1. You will have to figure out how to deal with the factors. We will discuss some strategies for doing this in our labs.*

First fit a tree, using a small value of cp:

```
> # do a tree
> library(rpart)
> tree.fit = rpart(log(price) ~ ., data = new.housing.df, cp=0.0001)
> plotcp(tree.fit)
```




Now find the cp giving the smallest error:

```
> cvTable =printcp(trcvTable =printcp(tree.fit)ee.fit)
> nullError = mean((log(new.housing.df$price)-
                    mean(log(new.housing.df$price)))^2)
> xvalError = nullError*min(cvTable[,4])
> xvalError

[1] 0.04513239
```

Now try a neural network, using caret to select the size and decay parameters:

```
library(caret)
library(nnet)
my.grid <- expand.grid(.decay = c(0,01, 0.15, 0.001), .size = c(4,6,8))

nn.CV <- train(log(price)~., data = new.housing.df,
method = "nnet",
maxit = 500,
tuneGrid = my.grid,
trace = FALSE,
linout = TRUE,
trControl = trainControl(method="cv", number=10,repeats=10)
)

No pre-processing
Resampling: Cross-Validated (10 fold)

Summary of sample sizes: 19451, 19453, 19452, 19452, 19452, 19452, ...

Resampling results across tuning parameters:

  decay  size  RMSE       Rsquared     RMSE SD      Rsquared SD
  0.000  4     0.4990486  0.769473109  0.085837425           NA
  0.000  6     0.5267093  0.001361214  0.009024680           NA
  0.000  8     0.4727478  0.380888640  0.112781066  0.436952162
  0.001  4     0.3711409  0.595297791  0.167662492  0.407754444
  0.001  6     0.2781318  0.664960973  0.131089853  0.351115557
  0.001  8     0.2640607  0.686464385  0.141755245  0.362669036
  0.150  4     0.1815606  0.881197877  0.003367343  0.007628566
  0.150  6     0.1825147  0.879975836  0.003924109  0.007870321
  0.150  8     0.2178002  0.878872967  0.108845387  0.006336701
  1.000  4     0.1867819  0.874362385  0.002252275  0.006115945
  1.000  6     0.1839517  0.878205128  0.003023317  0.006134047
  1.000  8     0.1843851  0.877597664  0.003930224  0.007547774

RMSE was used to select the optimal model using  the smallest value.
The final values used for the model were size = 4 and decay = 0.15.
```

The best error is 0.1815606 ^2 = 0.0330

Let's try running the neural network several times to see if the result above represents an approximate minimum:

```
fit.list = vector(length=10, mode="list")
MSE = numeric(10)
for(i in 1:10){
fit.list[[i]] = nnet(log(price)~., data=new.housing.df, size=4,
    linout=TRUE, decay=0.15, maxit=1000)
MSE[i]=fit.list[[i]]$value
}
MSE/dim(new.housing.df)[1]
 [1] 0.03222965 0.03246331 0.03276212 0.03249191 0.03274420 0.03176052
 [7] 0.03199673 0.03188747 0.03207831 0.03325348
```

Since with over 20,000 observations and a relatively small number of variables we expect the training and test errors to be similar, it would appear that the PE won't change much over repeated runs of the nnet function, and that an estimated PE 0f around 0.0330 seems quite plausible.

[10 marks]