5 Exercises

Application

Refer to the Air Quality data described previously, and the analyses we have done with Ozone as the response variable, and the five explanatory variables (including the two engineered features).
Use Neural Nets (NN) to model the relationship between Ozone and all five explanatories as specified below.:

1. Create a matrix of the five explanatory variables. Rescale each of them to lie between 0 and 1 and same them as another matrix. **Print a summary() of each object to confirm that you have scaled properly.**

```
# Title: STAT 452 Exercise 8 L12Q12
# Author: Injun Son
# Date: October 31, 2020

library(dplyr)
library(MASS)   # For ridge regression
library(glmnet) # For LASSO
source("Helper Functions.R")
data = na.omit(airquality[, 1:4])
data$TWcp = data$Temp*data$Wind
data$TWrat = data$Temp/data$Wind


# 1. Create a matrix of the five explanatory variables. Rescale each of them to lie between
# 0 and 1 and same them as another matrix. Print a summary() of each object to
# confirm that you have scaled properly.

### 75%/25% split into training and validation sets
n = nrow(data)
n.train = floor(n*0.75)
n.valid = n - n.train

inds = c(rep(1, times = n.train), rep(2, times = n.valid))
inds.rand = inds[sample.int(n)]

data.train = data[inds.rand == 1,]
X.train.raw = data.train[,-5]
Y.train = data.train[,5]

data.valid = data[inds.rand == 2,]
X.valid.raw = data.valid[,-5]
Y.valid = data.valid[,5]

```

```
### When fitting neural networks, it's best to scale the training set so
### that all predictors lie between 0 and 1. We then have to apply the same
### scaling to the validation set, so the validation set might have some
### observations below 0 or above 1. This is fine.
### We can use Tom's function, which scales each column in x1 so that
### the min and max in each column of x2 are 0 and 1 respectively.
rescale <- function(x1,x2){
  for(col in 1:ncol(x1)){
    a <- min(x2[,col])
    b <- max(x2[,col])
    x1[,col] <- (x1[,col]-a)/(b-a)
  }
  x1
}

X.train = rescale(X.train.raw, X.train.raw)
X.valid = rescale(X.valid.raw, X.train.raw) # Be careful with the order
```

```
> summary(X.train)
     Ozone           Solar.R           Wind             Temp           TWrat
 Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.00000
 1st Qu.:0.1018   1st Qu.:0.3104   1st Qu.:0.2772   1st Qu.:0.2763   1st Qu.:0.07554
 Median :0.1796   Median :0.6269   Median :0.4022   Median :0.5000   Median :0.11994
 Mean   :0.2402   Mean   :0.5430   Mean   :0.4151   Mean   :0.4807   Mean   :0.16533
 3rd Qu.:0.2904   3rd Qu.:0.7661   3rd Qu.:0.5000   3rd Qu.:0.6579   3rd Qu.:0.20358
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00000
> summary(X.valid)
     Ozone            Solar.R            Wind              Temp              TWrat
 Min.   :0.02994   Min.   :0.05199   Min.   :0.02717   Min.   :-0.05263   Min.   :0.001665
 1st Qu.:0.08683   1st Qu.:0.32798   1st Qu.:0.27038   1st Qu.: 0.40789   1st Qu.:0.084249
 Median :0.17964   Median :0.60245   Median :0.43478   Median : 0.57895   Median :0.117182
 Mean   :0.26369   Mean   :0.54587   Mean   :0.41557   Mean   : 0.53571   Mean   :0.178850
 3rd Qu.:0.42216   3rd Qu.:0.75306   3rd Qu.:0.54348   3rd Qu.: 0.71711   3rd Qu.:0.220365
 Max.   :0.68263   Max.   :0.89297   Max.   :0.87500   Max.   : 0.92105   Max.   :0.797665
```

2. Fit 4 NNs nets using only Temp and Wind, using each combination of 2 and 6 hidden nodes with 0.001 and 1 shrinkage; i,e,; (2,0.001), (2,1), (6,0.001), (6,1)

(a) Refit each one manually 20 times or more and compute the sMSE each time.

i. **Report the sMSE for the optimal fit for each model.**

(2, 0.001)

```
70  ##################################### (2, 0.001)
71
72  n.hidden = 2
73  shrink = 0.001
74
75  fit.nnet = nnet(y = Y.train, x = X.train[,c(3,4)], linout = TRUE, size = n.hidden,
76                  decay = shrink, maxit = 500)
77
78  n.nnets = 20 # Number of times to re-fit
79  ### Container for SSEs
80  all.SSEs = rep(0, times = 20)
81  all.MSEs = rep(0, times = 20)
82  all.nnets = list(1:20)
83  for(i in 1:n.nnets){
84    ### Fit model. We can set the input "trace" to FALSE to suppress the
85    ### printed output from the nnet() function.
86    this.nnet = nnet(y = Y.train, x = X.train[,c(3,4)], linout = TRUE, size = n.hidden,
87                     decay = shrink, maxit = 500, trace = FALSE)
88
89    ### Get the model's SSE
90    this.SSE = this.nnet$value
91    this.MSE = this.nnet$value / nrow(X.train)
92
93    ### Store results. We have to use double square brackets when storing or
94    ### retrieving from a list.
95    all.SSEs[i] = this.SSE
96    all.MSEs[i] = this.MSE
97    all.nnets[[i]] = this.nnet
98  }
99  all.MSEs
100
101 ind.best = which.min(all.SSEs)
102 fit.nnet.best = all.nnets[[ind.best]]
103
104 x1 <- seq(from=2.3, to=20.7, by=.5)
105 x2 = seq(from=57, to=97, by=.5)
106 xy1 <- data.frame(expand.grid(wind=x1, temp=x2))
107
108 pred2 <- predict(fit.nnet ,newdata=rescale(xy1, X.valid.raw[,c(3,4)]))
109 surface2 = matrix(pred2, nrow=length(x1))
```

```
open3d()
persp3d(x = x1, y = x2,
        z = surface2, col = "orange", xlab="Wind", ylab="Temp",
        zlab="Ozone")
points3d(data$Ozone ~ data$Wind + data$Temp, col="blue")
```

```
> all.MSEs
 [1]    56.16532    56.33156    93.24751   109.80589 58121.62992    43.01570   109.80538    40.53980 58120.69365
[10]   105.25589 25434.99372   109.80613 58125.31099 58120.17503    31.22951    43.01570    99.33437    56.16532
[19]   109.80778    79.74207
```

```
##################################### (2, 1)
n.hidden = 2
shrink = 1

fit.nnet = nnet(y = Y.train, x = X.train[,c(3,4)], linout = TRUE, size = n.hidden,
                decay = shrink, maxit = 500)

n.nnets = 20 # Number of times to re-fit
### Container for SSEs
all.SSEs = rep(0, times = 20)
all.MSEs = rep(0, times = 20)
all.nnets = list(1:20)
for(i in 1:n.nnets){
  ### Fit model. We can set the input "trace" to FALSE to suppress the
  ### printed output from the nnet() function.
  this.nnet = nnet(y = Y.train, x = X.train[,c(3,4)], linout = TRUE, size = n.hidden,
                   decay = shrink, maxit = 500, trace = FALSE)

  ### Get the model's SSE
  this.SSE = this.nnet$value
  this.MSE = this.nnet$value / nrow(X.train)

  ### Store results. We have to use double square brackets when storing or
  ### retrieving from a list.
  all.SSEs[i] = this.SSE
  all.MSEs[i] = this.MSE
  all.nnets[[i]] = this.nnet
}
all.MSEs
ind.best = which.min(all.SSEs)
fit.nnet.best = all.nnets[[ind.best]]

x1 <- seq(from=2.3, to=20.7, by=.5)
x2 = seq(from=57, to=97, by=.5)
xy1 <- data.frame(expand.grid(wind=x1, temp=x2))

pred2 <- predict(fit.nnet ,newdata=rescale(xy1, X.valid.raw[,c(3,4)]))
surface2 = matrix(pred2, nrow=length(x1))


open3d()
persp3d(x = x1, y = x2,
```

```
persp3d(x = x1, y = x2,
        z = surface2, col = "orange", xlab="Wind", ylab="Temp",
        zlab="Ozone")
points3d(data$Ozone ~ data$Wind + data$Temp, col="blue")
```

```
> all.MSEs
 [1] 1721.232 1621.757 1137.290 1721.232 1954.603 1137.290 1137.290 1721.232 1137.290 1721.232 1621.757 1621.757 1137.290
[14] 1137.290 1721.232 1137.290 1721.232 1137.290 1137.290 1137.290
```

```r
n.hidden = 6
shrink = 0.001

fit.nnet = nnet(y = Y.train, x = X.train, linout = TRUE, size = n.hidden,
                decay = shrink, maxit = 500)

n.nnets = 20 # Number of times to re-fit
### Container for SSEs
all.SSEs = rep(0, times = 20)
all.MSEs = rep(0, times = 20)
all.nnets = list(1:20)
for(i in 1:n.nnets){
  ### Fit model. We can set the input "trace" to FALSE to suppress the
  ### printed output from the nnet() function.
  this.nnet = nnet(y = Y.train, x = X.train, linout = TRUE, size = n.hidden,
                   decay = shrink, maxit = 500, trace = FALSE)

  ### Get the model's SSE
  this.SSE = this.nnet$value
  this.MSE = this.nnet$value / nrow(X.train)

  ### Store results. We have to use double square brackets when storing or
  ### retrieving from a list.
  all.SSEs[i] = this.SSE
  all.MSEs[i] = this.MSE
  all.nnets[[i]] = this.nnet
}
all.MSEs
### Get the best model. We have to use double square brackets when storing or
### retrieving from a list.
# ind.best = which.min(all.SSEs)
# fit.nnet.best = all.nnets[[ind.best]]
ind.best = which.min(all.MSEs)
fit.nnet.best = all.nnets[[ind.best]]

### Get predictions and MSPE
pred.nnet = predict(fit.nnet.best, X.valid)
MSPE.nnet = get.MSPE(Y.valid, pred.nnet)
```

```
> all.MSEs
 [1] 58119.579968    34.550258    34.433917     7.315829    15.705896     8.412257 58120.278681     8.862501    12.428587
[10]     8.813635    10.565405 58122.348502 58117.578898 58117.820407    13.542882     7.553607     6.763851    28.733807
[19]    12.141556    27.231437
```

```
222 ################################### (6, 1)
223 n.hidden = 6
224 shrink = 1
225
226 fit.nnet = nnet(y = Y.train, x = X.train[,c(3,4)], linout = TRUE, size = n.hidden,
227                 decay = shrink, maxit = 500)
228
229 n.nnets = 20 # Number of times to re-fit
230 ### Container for SSEs
231 all.SSEs = rep(0, times = 20)
232 all.MSEs = rep(0, times = 20)
233 all.nnets = list(1:20)
234 for(i in 1:n.nnets){
235   ### Fit model. We can set the input "trace" to FALSE to suppress the
236   ### printed output from the nnet() function.
237   this.nnet = nnet(y = Y.train, x = X.train[,c(3,4)], linout = TRUE, size = n.hidden,
238                    decay = shrink, maxit = 500, trace = FALSE)
239
240   ### Get the model's SSE
241   this.SSE = this.nnet$value
242   this.MSE = this.nnet$value / nrow(X.train)
243
244   ### Store results. We have to use double square brackets when storing or
245   ### retrieving from a list.
246   all.SSEs[i] = this.SSE
247   all.MSEs[i] = this.MSE
248   all.nnets[[i]] = this.nnet
249 }
250 all.MSEs
251 ### Get the best model. We have to use double square brackets when storing or
252 ### retrieving from a list.
253 # ind.best = which.min(all.SSEs)
254 # fit.nnet.best = all.nnets[[ind.best]]
255 ind.best = which.min(all.SSEs)
256 fit.nnet.best = all.nnets[[ind.best]]
257
258 x1 <- seq(from=2.3, to=20.7, by=.5)
259 x2 = seq(from=57, to=97, by=.5)
260 xy1 <- data.frame(expand.grid(wind=x1, temp=x2))
261
262 #pred.nnet = predict(fit.nnet.best, X.valid)
263 # pred.nnet = predict(fit.nnet.best, X.valid)
264 # MSPE.nnet = get.MSPE(Y.valid, pred.nnet)
265 # pred2 <- predict(pred.nnet ,newdata=rescale(xy1, X.valid.raw[,c(3,4)]))
266 pred2 <- predict(fit.nnet ,newdata=rescale(xy1, X.valid.raw[,c(3,4)]))
267 surface2 = matrix(pred2, nrow=length(x1))
268
269
270 open3d()
271 persp3d(x = x1, y = x2,
272         z = surface2, col = "orange", xlab="Wind", ylab="Temp",
273         zlab="Ozone")
274 points3d(data$Ozone ~ data$Wind + data$Temp, col="blue")
```

```
> all.MSEs
 [1] 3394.466 3889.075 3399.422 3394.466 3410.263 3402.865 3394.466 3401.277 3401.277 3398.238 3403.193 3403.194 3520.883
[14] 3520.882 3394.466 3394.471 3395.892 3400.495 3401.277 3395.892
```

ii. Comment on the stability of fits for different models. In other words, **which models were most/least consistent with the sMSE values produced by different fits?**

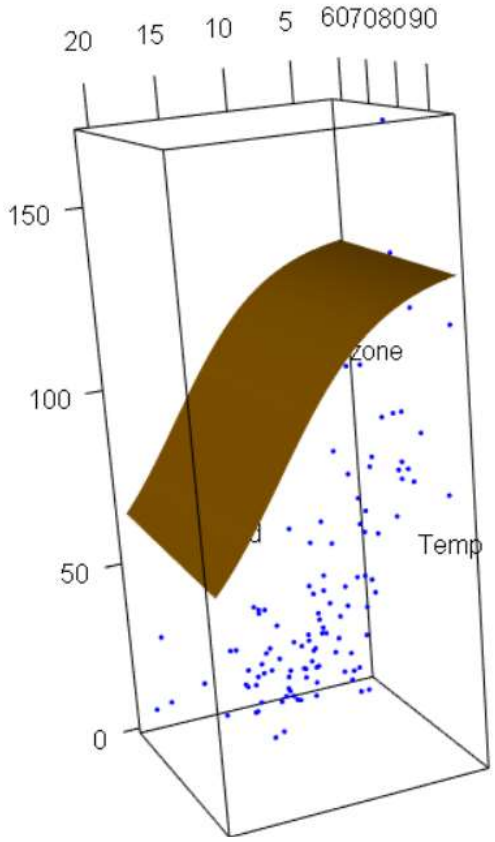
-> Model with 6 hidden nodes and 1 shrinkage has the most consistent sMSE.

->Model with 2 hidden nods and 0.001 shrinkage has the inconsistent sMSE.

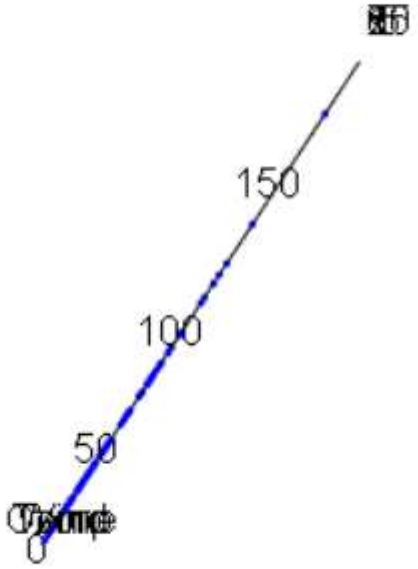
(b) Make a 3-D plot of each model's fit.

i. **Report a screenshot of each fit, rotated to roughly the same angle each time to show a good comparison of the fits.** (I find it best to look down through the corner with low temp and high wind, so that the high ozone values are in the back.)

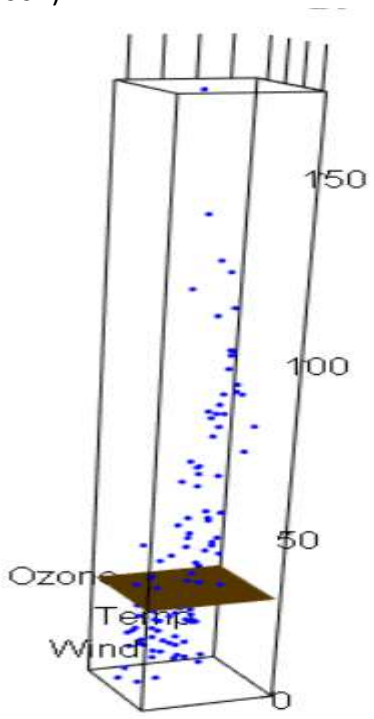
(2, 0.001)



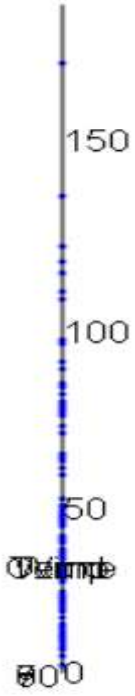
(2, 1)



(6, 0.001)



(6, 1)



ii. **Comment separately on how increasing number of nodes or increasing shrinkage appears to affect the fits.**

➔ It looks like increasing shrinkage make more exact fits, but this looks like an over fit.