DEPARTMENT OF STATISTICS

STATS 760 A Survey of Modern Applied Statistics

Assignment 2 2017 Model answers

Question 1. The first step is to read in the data. In fact, there are several duplicated rows in the data so we remove then, and then scale the variables. The following code does this:

```
> URL = "C:\\Users\\alee044\\Documents\\Teaching\\760\\2017\\Assignments\\Assignment 2\\red.txt"
> red.df = read.table(URL, header=TRUE, sep=";")
> redDups = duplicated(red.df)
> redNoDupsStd.df = data.frame(scale(red.df[!redDups,]))
```

Linear regression

We first fit a linear regression, and run the **allpossregs** function in the R330 package to find the subset with the smallest cross-validated prediction error:

```
> allpossregs(red.lm, cv.rep=20)
       rssp sigma2 adjRsq      Cp      AIC      BIC      CV
1  1044.670  0.770  0.230 273.744 1632.744 1643.173 103.871
2   917.306  0.676  0.324  77.172 1436.172 1451.816  91.450
3   892.608  0.659  0.341  40.665 1399.665 1420.523  89.114
4   881.653  0.651  0.349  25.586 1384.586 1410.658  88.181
5   873.318  0.645  0.355  14.591 1373.591 1404.878  87.483
6   866.516  0.641  0.359   5.986 1364.986 1401.487  87.011
7   864.810  0.640  0.360   5.326 1364.326 1406.042  86.973
8   864.128  0.640  0.360   6.262 1365.262 1412.192  87.086
9   864.083  0.641  0.359   8.191 1367.191 1419.336  87.322
10  864.050  0.641  0.359  10.140 1369.140 1426.500  87.546
11  863.960  0.641  0.359  12.000 1371.000 1433.574  87.776
   fixed.acidity volatile.acidity citric.acid residual.sugar
1              0                0           0              0
2              0                1           0              0
3              0                1           0              0
4              0                1           0              0
5              0                1           0              0
6              0                1           0              0
7              0                1           0              0
8              0                1           1              0
9              0                1           1              1
10             1                1           1              1
11             1                1           1              1
   chlorides free.sulfur.dioxide total.sulfur.dioxide density
1          0                   0                    0       0
2          0                   0                    0       0
3          0                   0                    0       0
4          1                   0                    0       0
5          1                   0                    1       0
```

```
6          1                   0                   1       0
7          1                   1                   1       0
8          1                   1                   1       0
9          1                   1                   1       0
10         1                   1                   1       0
11         1                   1                   1       1
   pH sulphates alcohol
1   0         0       1
2   0         0       1
3   0         1       1
4   0         1       1
5   0         1       1
6   1         1       1
7   1         1       1
8   1         1       1
9   1         1       1
10  1         1       1
11  1         1       1
>
```

Model 7 (ie dropping fixed.acidity, citric.acid, residual.sugar and density) was consistenly the best over several similar runs.

To compute the PE using 5-fold cross-validation, we can't use the above CV values as they are relative not absolute measures of PE. Instead, let's use the function crossval in the bootstrap package. To do this we need to write functions theta.fit and theta.predict to do the fitting and predicting

```
####################################################
# using bootstrap package
library(bootstrap)
theta.fit = function(x,y){
lm(y~x)
}

theta.predict = function(fit,x){
cbind(1, x)%*% coef(fit)
}

y = redNoDupsStd.df[,12]
# full model

y = redNoDupsStd.df[,12]
x = as.matrix(redNoDupsStd.df[,-12])

# repeat 20 times
cv5 = numeric(20)
for(i in 1:20){
CV5 = crossval(x,y,theta.fit,theta.predict,ngroup=5)
cv5[i]=mean((CV5$cv.fit-y)^2)
}
> mean(cv5)
[1] 0.6523595
> sd(cv5)
```

```
# best submodel
x = as.matrix(redNoDupsStd.df[,-c(1,3,4,8,12)])

# repeat 20 times
cv5 = numeric(20)
for(i in 1:20){
CV5 = crossval(x,y,theta.fit,theta.predict,ngroup=5)
cv5[i]=mean((CV5$cv.fit-y)^2)
}
mean(cv5)
[1] 0.6467387
```

This is the CV5 estimate. Note that crossval calculates the CV predictions only, we have to combine them to get the prediction error.

```
sd(cv5)
[1] 0.003075318

# using caret

library(caret)

# full model
lmCV <- train(quality~., data = redNoDupsStd.df,
method = "lm",
trControl = trainControl(method="cv", number=5,
repeats=100))

  RMSE       Rsquared   RMSE SD     Rsquared SD
  0.8067013  0.3523172  0.02674587  0.02964645

# submodel

lmCVsub <- train(quality~., data = redNoDupsStd.df[,-c(1,3,4,8)],
method = "lm",
trControl = trainControl(method="cv", number=5,
repeats=100))

  RMSE       Rsquared   RMSE SD     Rsquared SD
  0.8034705  0.3540637  0.03825519  0.03468824
```

0.8034705^2 = 0.6455648 so this agrees pretty well with the crossval calculation above.

Gams

Let's fit a series of models using "do-it-yourself" backward elimination

```
use= rep(TRUE,11)
smooth.names = paste("s(",varnames[use],")", sep="")
formula = as.formula(paste("y ~", paste(smooth.names, collapse="+")))
```

```
myCV(formula, data=redNoDupsStd.df, fittingFunctiongam)


# begin backward elimination
gam.fit = gam(formula, data=redNoDupsStd.df)



summary(gam.fit)

s(fixed.acidity)        2.045  2.627  0.979  0.38535
s(volatile.acidity)     1.000  1.000 49.353 3.35e-12 ***
s(citric.acid)          3.095  3.899  1.117  0.34443
s(residual.sugar)       1.000  1.000  0.371  0.54245
s(chlorides)            7.722  8.546  2.546  0.00800 **
s(free.sulfur.dioxide)  2.888  3.677  1.112  0.34330
s(total.sulfur.dioxide) 6.725  7.589  3.217  0.00165 **
s(density)              1.000  1.000  2.155  0.14235
s(pH)                   1.000  1.000  8.608  0.00340 **
s(sulphates)            4.282  5.292 23.797  < 2e-16 ***
s(alcohol)              7.744  8.577 11.189 3.93e-16 ***
AIC(gam.fit)
[1]  3182.695

The largest P-value is 0.54245  suggesting residual sugar should be removed. We can use
caret to get the PE:

my.grid <- expand.grid(.select=c(FALSE),.method ="GCV.Cp")
train(quality~., data = redNoDupsStd.df,
method = "gam",
tuneGrid = my.grid,
trControl = trainControl(method="cv", number=5,
repeats=100))$results[1,3]

[1] 0.8156662
```

Proceeding in this way we eliminate successively residual sugar, fixed acidity, citric acid and sulphur dioxide (not quite the same as linear regression)

We get the following results:

Model	AIC	CV5
Full	3182.695	0.634
-residual sugar	3182.005	0.631
- fixed acidity	3177.554	0.633
- citric acid	3182.386	0.644
-sulphur dioxide	3176.335	0.628

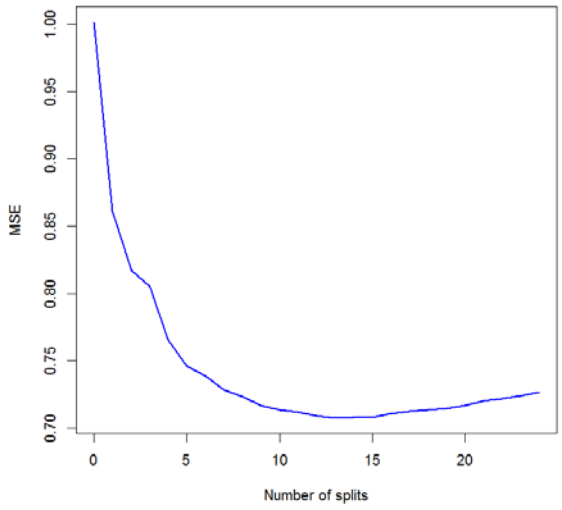
suggesting there is not much difference, but the 7-variable model looks OK.

Trees

We can fit the tree with say cp=0.001 and look at the cp plot. Since these are based on cross-validation, successive plots will differ slightly. To get around this, I repeated the cp plot 100 times, and averaged the plots, using the following code:

```
library(rpart)
M=100
MSE = matrix(0, M, 25)
for(i in 1:M){
tree.fit = rpart(quality~., data=redNoDupsStd.df, cp=0.001)
MSE[i,] =tree.fit$cptable[1:25,4]
}
MSEvec = apply(MSE, 2,mean)
plot(1:25, MSEvec, xlab="Number of splits", ylab = "MSE", type="l",
  lwd=2, col="blue")
> min(MSEvec)
[1] 0.7077107
```

This produced the plot below. Looks like a tree with 13 splits is the best one, with MSE 0.7077. Note that since we have standardized the data, the relative MSE's are essentially the absolute ones.

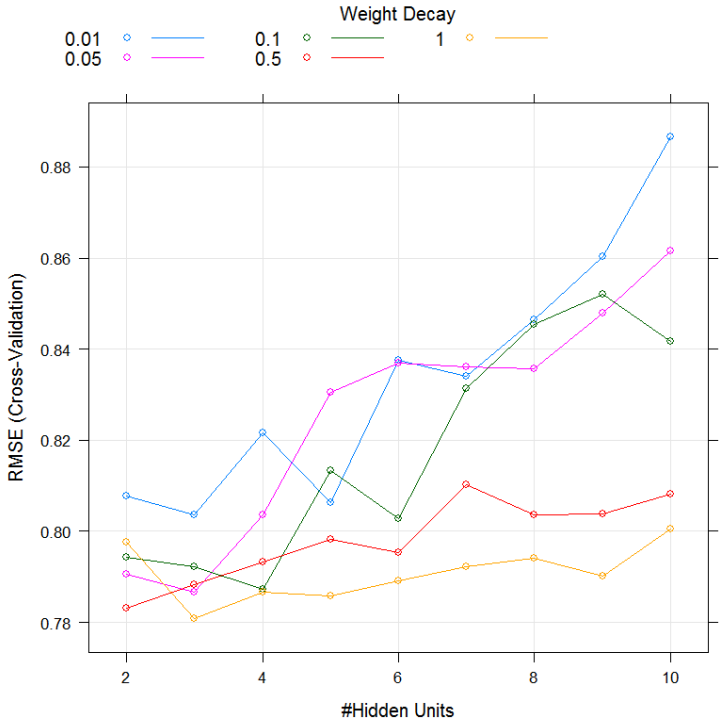


Neural networks

For this part we used caret, searching over values 2:10 for the hidden layer, and Decay parameters of 0.01, 0.05, 0.1, 0.5, 1. We used the code below:

```
library(caret)
library(nnet)
```

```
my.grid <- expand.grid(.decay = c(0.01, 0.05, 0.1, 0.5, 1), .size = 2:10)
nnCV <- train(quality~., data = redNoDupsStd.df,
method = "nnet",
maxit = 1000,
tuneGrid = my.grid,
trace = FALSE,
linout = TRUE,
trControl = trainControl(method="cv", number=5,
repeats=1000))
plot(nnCV)
```



Looks like 3 units with a decay of 1 was the best of these: MSE was 0.7807724^2= 0.6096055.

Summary table

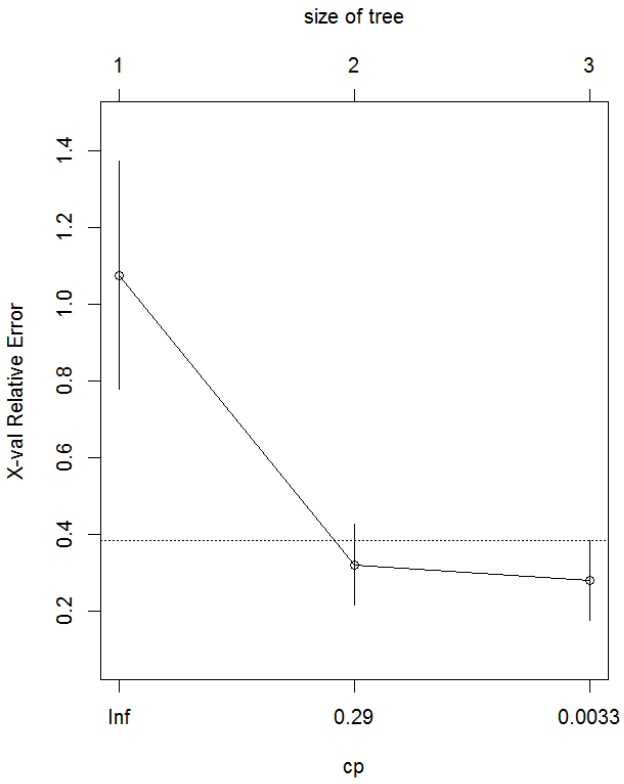
Method	MSE (CV5)
Linear regression, best submodel	0.647
Gam, all variables	0.634
Gam, backward elimination	0.628
Tree (13 terminal nodes, cp=0.0065)	0.708
Neural net (h=3, decay=1)	0.609

We see that the neural net wins, but only by a bit. The tree is the worst. None of the methods is doing very well, since the prediction error of the null model (i.e. fitting a mean) is about 1. We will see in Assignment 4 that this may be due to treating the response as a continuous variable.

[8 marks for each of the four methods – 32 in all]

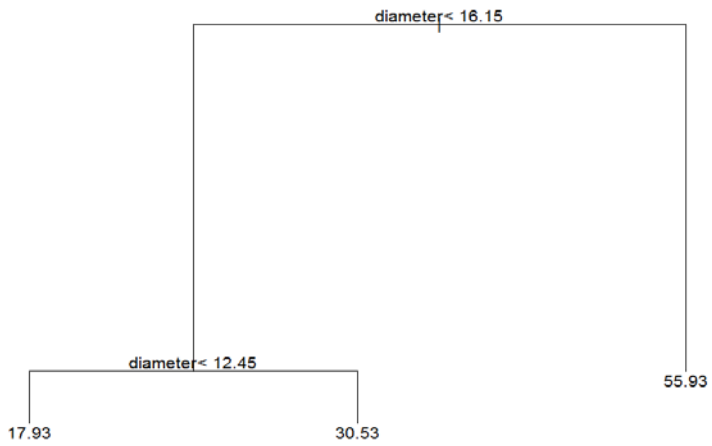
Question 2. We fit the tree as follows:

```
tree.fit = rpart(volume~., data=cherry.df, cp=0.0001)
plotcp(tree.fit)
```

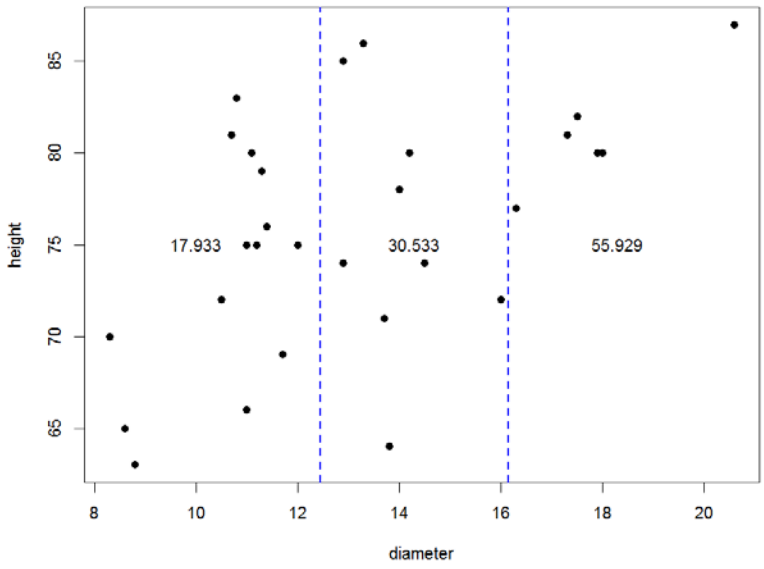


Thus, a very simple tree with three terminal nodes is indicated:

```
plot(tree.fit)
text(treefit)
```



This corresponds to the regions shown below:



Code for diagram:

```
plot(height~diameter, data=cherry.df, pch=19)
abline(v=12.45, lty=2, lwd=2, col="blue")
abline(v=16.15, lty=2, lwd=2, col="blue")
text(10,75,round(mean(cherry.df$volume[cherry.df$diameter <12.45]),3))
mid = (cherry.df$diameter >12.45)&(cherry.df$diameter <16.15)
text(14.3,75,round(mean(cherry.df$volume[mid]),3))
text(18.3,75,round(mean(cherry.df$volume[cherry.df$diameter >16.15]),3))
```

[8 marks for a correct diagram]