DEPARTMENT OF STATISTICS

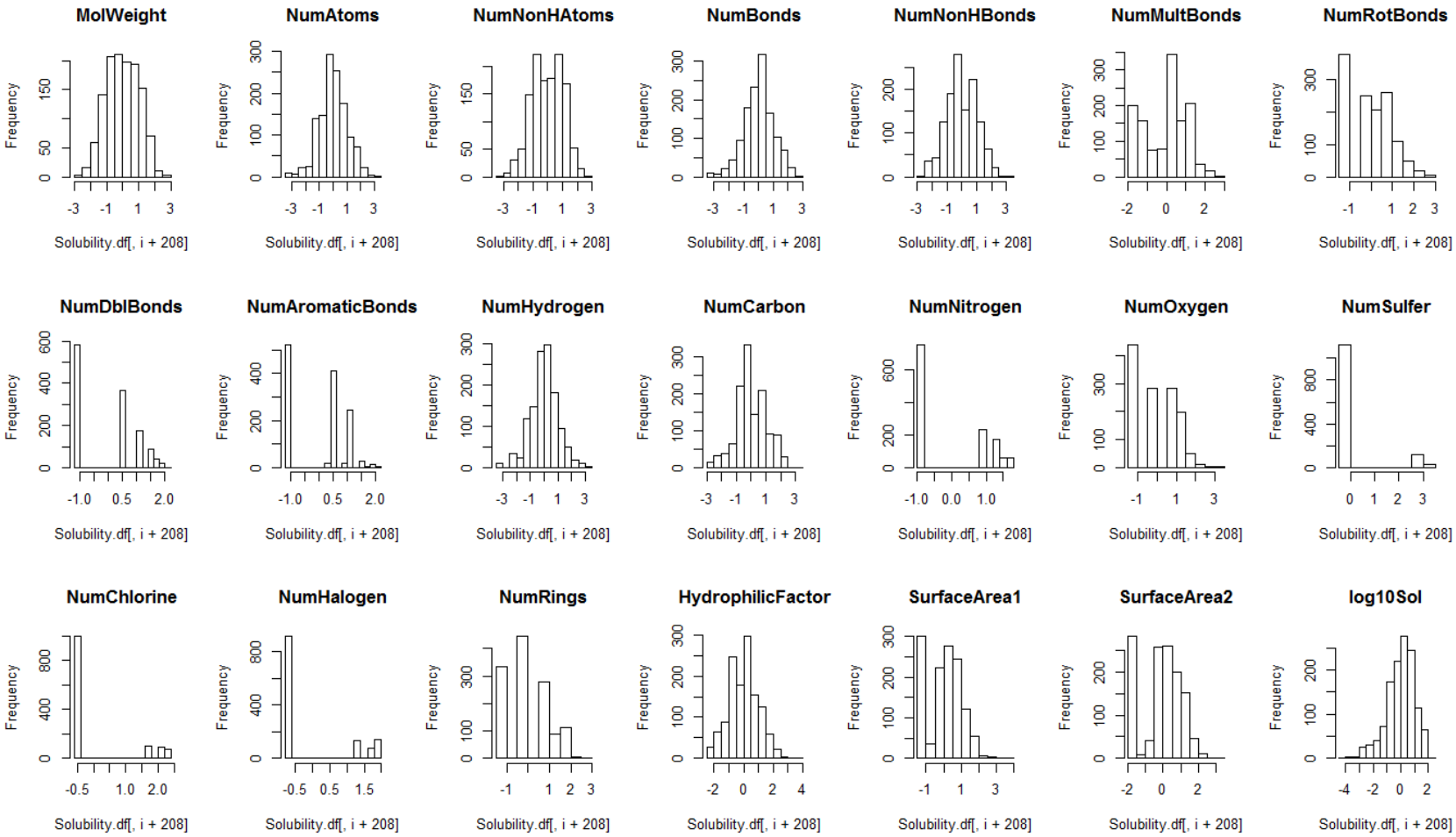
STATS 760 A Survey of Modern Applied Statistics

Assignment 3 2017 Model answers

1. *Read the data into R. Some of the non-binary variables are quite skewed so you may wish to transform them to make them more symmetric. The caret package has a function BoxCoxTransform that you may find useful.*

```
# Read data into R, transforming the non-binary variables

infile  = "type your text file name here"
Solubility.df = read.table(infile, header=TRUE)

# Check that if we add 1 all non-binary variables are positive
# required for BoxCox. The non-binary variables are numbers 209-229

pos = logical(21)
for(i in 209:229) pos[i-208] = all(Solubility.df[,i]+1>0)
> pos
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE

# The response ( variable 220) is  a log, so we could exponentiate it to make it positive
Solubility.df[,229] = exp(Solubility.df[,229])

#Now apply Box-Cox to the non-binary variables
library(caret)
for(i in 1:20){
boxcox = BoxCoxTrans(Solubility.df[,i+208]+1)
Solubility.df[,i+208]= predict(boxcox, Solubility.df[,i+208]+1)

# and the response
boxcox = BoxCoxTrans(Solubility.df[,229])
Solubility.df[,229]= predict(boxcox, Solubility.df[,229])
}


# finally scale the non-binary variables (Useful for ridge regression)
Solubility.df[,209:229] = data.frame(scale(Solubility.df[,209:229]))

# check skewness
par(mfrow=c(3,7))
for(i in 1:21)hist(Solubility.df[,i+208], main = names(Solubility.df)[i+208])
```

Picture is shown on next page.


MolWeight
NumAtoms
NumNonHAtoms
NumBonds
NumNonHBonds
NumMultBonds
NumRotBonds
NumDblBonds
NumAromaticBonds
NumHydrogen
NumCarbon
NumNitrogen
NumOxygen
NumSulfer
NumChlorine
NumHalogen
NumRings
HydrophilicFactor
SurfaceArea1
SurfaceArea2
log10Sol
Frequency
Solubility.df[, i + 208]

Variables look reasonably symmetric. [3 marks]

2. Fit a linear regression as a benchmark. Calculate a bootstrap prediction error.

```
> lm.fit = lm(log10Sol~., data=Solubility.df)
> summary(lm.fit)$sigma
[1] 0.2780027
> summary(lm.fit)$r.squared
[1] 0.9366332
```

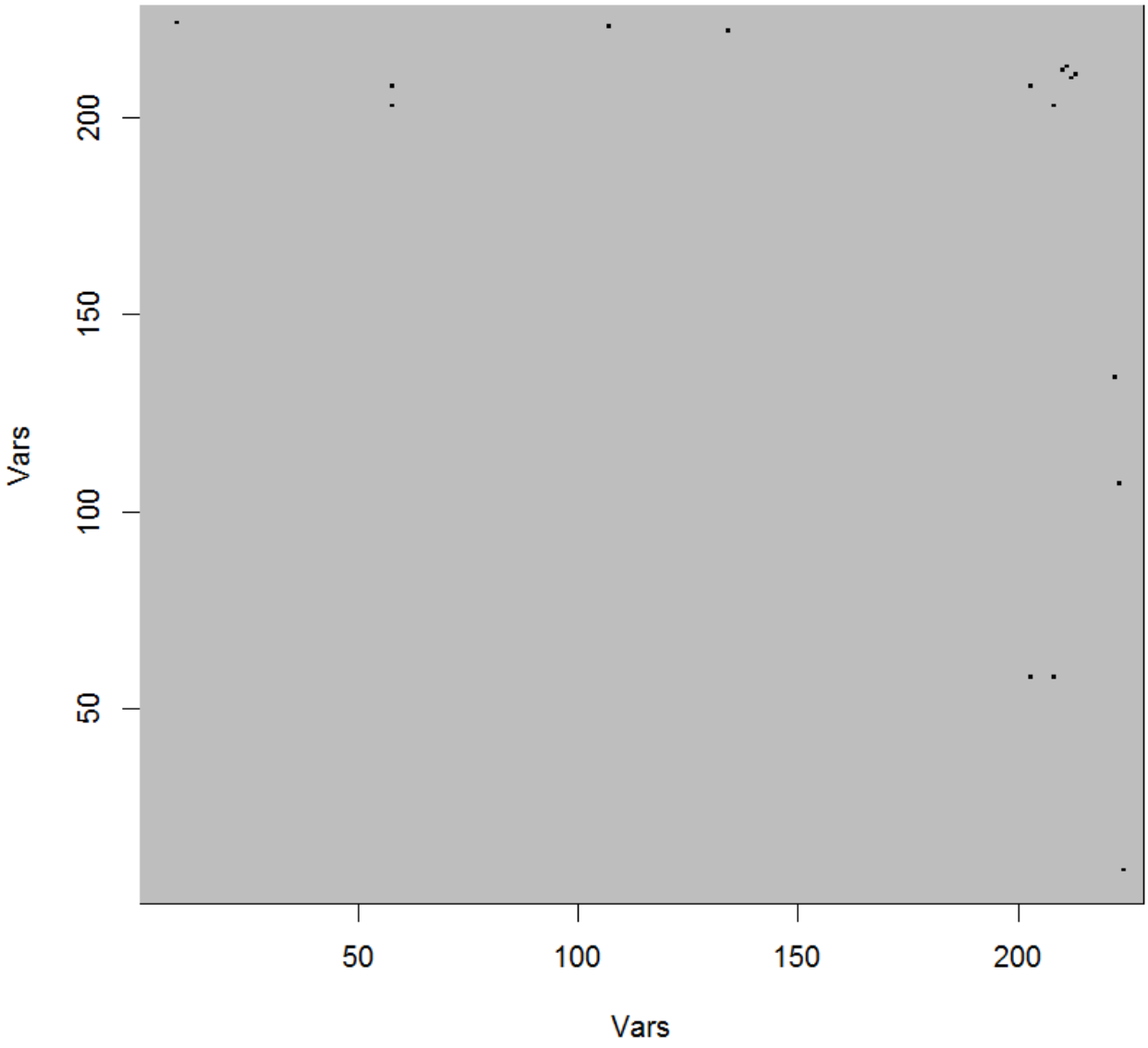
The fit appears good, but with so many variables we may be overfitting. The training error is 0.278^2 = 0.077. Lets work out a bootstrap estimate of the PE. We will use caret to calculate the 632 estimator.

```
lmboot <- train(log10Sol~., data = Solubility.df,
method = "lm",
trControl = trainControl(method="boot632",number=100))

 RMSE       Rsquared   RMSE SD     Rsquared SD
  0.3274103  0.8921733  0.01730904  0.01406932


> 0.3274103^2
[1] 0.1071975
```

Seems the linear regression might be overfitting, as the training error is quite a bit smaller than the bootstrap error.

In addition, there were several warnings: on investigation these turned out to be caused by some of the bootstrap samples containing identical binary variables, due to pairs of variables in the full data being almost identical. We can display this using a heatmap of the correlations, showing which pairs have correlations greater than say 0.99:

```
Cormat = cor(Solubility.df[,-229])
Vars = 1:228
image(Vars, Vars, (abs(Cormat)>0.99)&(row(Cormat)!=col(Cormat)),
col=c("grey","black"))
```



There are quite a few large correlations, indicated by the (small) black dots. A submodel seems indicated:

```
submodel = step(lm.fit, formula(lm.fit), direction = "back")
```

The submodel has deleted a substantial number of variables. Let's calculate the PE for the submodel

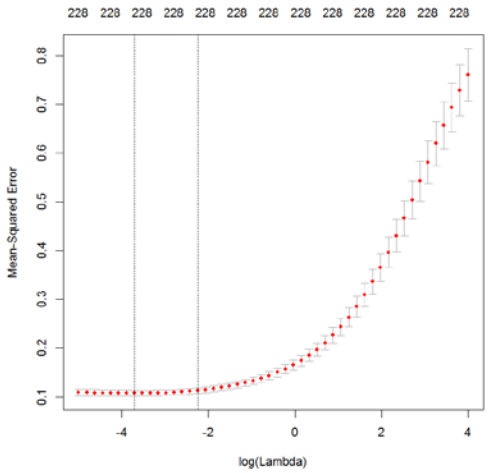
```
lmboot <- train(formula(submodel), data = Solubility.df,
method = "lm",
trControl = trainControl(method="boot",number=100))

RMSE       Rsquared   RMSE SD     Rsquared SD
0.3092598  0.9046216  0.01002624  0.009650583


 > 0.3092598^2
[1] 0.09564162
```

A small improvement, with no warnings. Note that there is a caret function identifying almost identical variables, so this would have been another approach. [4 marks]

3. *Fit a ridge regression and a lasso, finding a suitable value of lambda in each case. Calculate bootstrap predict errors (including the process of finding the lambda.) Do these improve over linear regression?*

Note that Ridge regression will not have a problem with almost identical variables due to the regularization. For Ridge:

```
library(glmnet)
X = as.matrix(Solubility.df[,-229])
y = Solubility.df[,229]
par(mfrow=c(1,1))
lambdaSeq = rev(exp(seq(-5,4, length=50)))
glmnet.fit = glmnet(X,y, alpha=0, lambda=lambdaSeq)
plot.glmnet = cv.glmnet(X,y, alpha=0, lambda=glmnet.fit$lambda)
plot(plot.glmnet)
```




```
> order(plot.glmnet$cvm)[1]
 [1] 43
> plot.glmnet$cvm[43]
[1] 0.1089926
> glmnet.fit$lambda[43]
[1] 0.02437284
```

Thus, the best lambda value is `0.02437284. Similar code (with alpha=1) gives the lasso solution.`

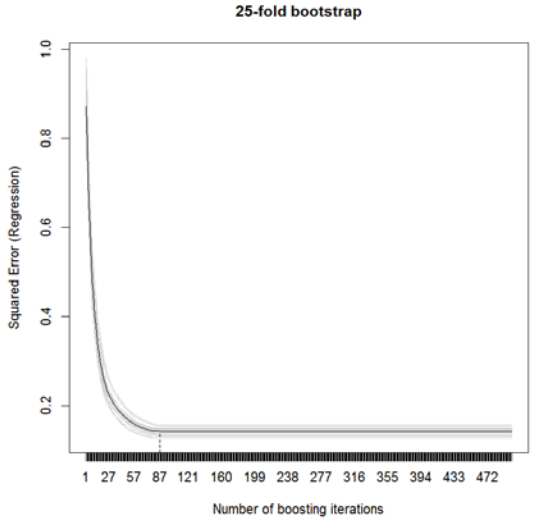
```
#   use caret to find bootstrap errors

my.grid = expand.grid(.alpha = c(0,1), .lambda = c(0.02, 0.04,0.06,0.08,0.10))
glmnet.boot <- train(log10Sol~., data = Solubility.df,
method = "glmnet",
tuneGrid = my.grid,
trControl = trainControl(method="boot632",number=100))

> glmnet.boot

RMSE was used to select the optimal model using  the smallest value.
The final values used for the model were alpha = 0 and lambda = 0.06.
> 0.3247736^2
[1] 0.1054779
```

[10 marks]

4. Fit boosted trees and random forests. Again, calculate prediction errors.

First, boosted trees , using blackboost. Trees should be small

```
# Boosting with trees as base learner
library(party)
library(mboost)
my.grid = expand.grid(.mstop = c(80,100,120), .maxdepth = c(2,3,4))
sol.tree <- blackboost(log10Sol~., data = Solubility.df,
     control= boost_control (mstop = 500, nu = 0.1,center = TRUE),
     tree_controls = ctree_control(maxdepth = 2))
errors = cvrisk(sol.tree)
plot(errors)
```

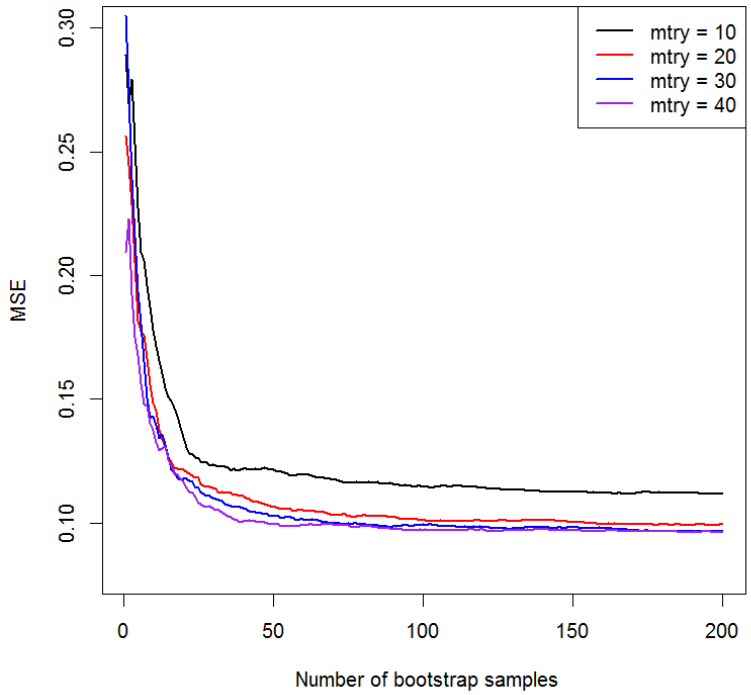


We will use caret to find the bootstrap error estimates.

```
> my.grid = expand.grid(.mstop = c(80,100,120), .maxdepth = c(2,3,4))
> glmnet.boot <- train(log10Sol~., data = Solubility.df,
+ method = "blackboost",
+ tuneGrid = my.grid,
+ trControl = trainControl(method="boot632",number=100))

Resampling results across tuning parameters:

  maxdepth  mstop  RMSE       Rsquared   RMSE SD     Rsquared SD
  2          80    0.3652974  0.8692471  0.01511913  0.01221823
  2         100    0.3648743  0.8695110  0.01536040  0.01238189
  2         120    0.3648743  0.8695110  0.01536040  0.01238189
  3          80    0.3409366  0.8865031  0.01567803  0.01168346
  3         100    0.3409366  0.8865031  0.01567803  0.01168346
  3         120    0.3409366  0.8865031  0.01567803  0.01168346
  4          80    0.3205003  0.8995133  0.01402244  0.01055573
  4         100    0.3205003  0.8995133  0.01402244  0.01055573
  4         120    0.3205003  0.8995133  0.01402244  0.01055573

RMSE was used to select the optimal model using  the smallest value. The final values used
for the model were mstop = 80 and maxdepth = 4.

> 0.3205003^2
[1] 0.1027204
```

For random forests:

```
library(randomForest)
sol10.rf=randomForest(log10Sol~., data = Solubility.df,
ntree=200, mtry=10, importance=TRUE)
plot(1:200, sol10.rf$mse, lwd=2, type="l", ylim = c(0.08, 0.3),
xlab = " Number of bootstrap samples", ylab = "MSE")

sol20.rf=randomForest(log10Sol~., data = Solubility.df,
ntree=200, mtry=20, importance=TRUE)
lines(1:200, sol20.rf$mse, lwd=2, col="red")

sol30.rf=randomForest(log10Sol~., data = Solubility.df,
ntree=200, mtry=30, importance=TRUE)
lines(1:200, sol30.rf$mse,lwd=2, col="blue" )

sol40.rf=randomForest(log10Sol~., data = Solubility.df,
ntree=200, mtry=40, importance=TRUE)
lines(1:200, sol40.rf$mse,lwd=2, col="purple" )

legend("topright", paste("mtry =", c(10,20,30, 40)), lwd=2,
col=c("black","red","blue","purple"))
```



Looks like mtry=40 is about right. The MSE(based on the out-of-bag samples) is

```
> min(sol40.rf$mse)

[1] 0.09453398
```

Finally, Let's try caret to calculate the 632 bootstrap errors. This takes quite a while as we are bootstrapping the bootstrap!

```
rf.boot632 <- train(log10Sol~., data = Solubility.df,
method = "rf",
tuneGrid = my.grid,
trControl = trainControl(method="boot632",number=100))
> rf.boot

  mtry  RMSE       Rsquared   RMSE SD     Rsquared SD
  10    0.2847141  0.9181337  0.01933616  0.010552734
  20    0.2650820  0.9250110  0.01807561  0.010060970
  30    0.2595427  0.9268816  0.01741740  0.009886580
  40    0.2569637  0.9276767  0.01718255  0.009893061

RMSE was used to select the optimal model using  the smallest value.
The final value used for the model was mtry = 40.
> 0.2569637^2
[1] 0.06603034
```

[10 marks]

5. *Compare all your results, taking care to explain how you got the various estimates of PE (particularly for part 4.)*

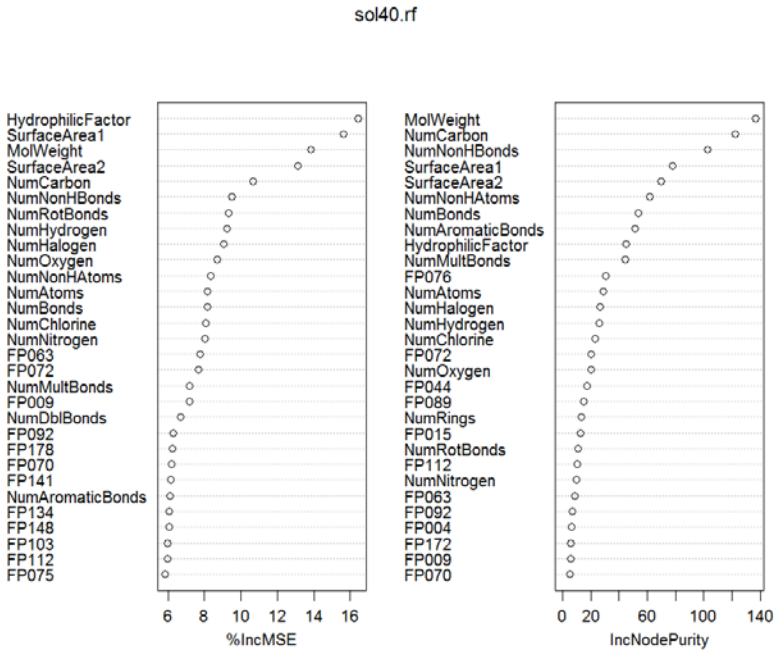
The estimates of PE in the table below are obtained by using caret and the 632 estimator.

Method	PE (0.632 estimator)
LS	0.017
LS (submodel)	0.096
Ridge	0.105
Lasso	0.128
Boosted tree	0.102
Random Forest	0.066

[3 marks]

6. *Which variables are the most important in predicting the solubility?*

We use the function varImpPlot in the randomForest package to calculate the variable importance (using both definitions) and plot the results.



(Produced by the code `varImpPlot(sol40.rf)`). Thus, the most important variables are HydrophilicFactor, MolWeight,SurfaceArea1, Numcarbon etc

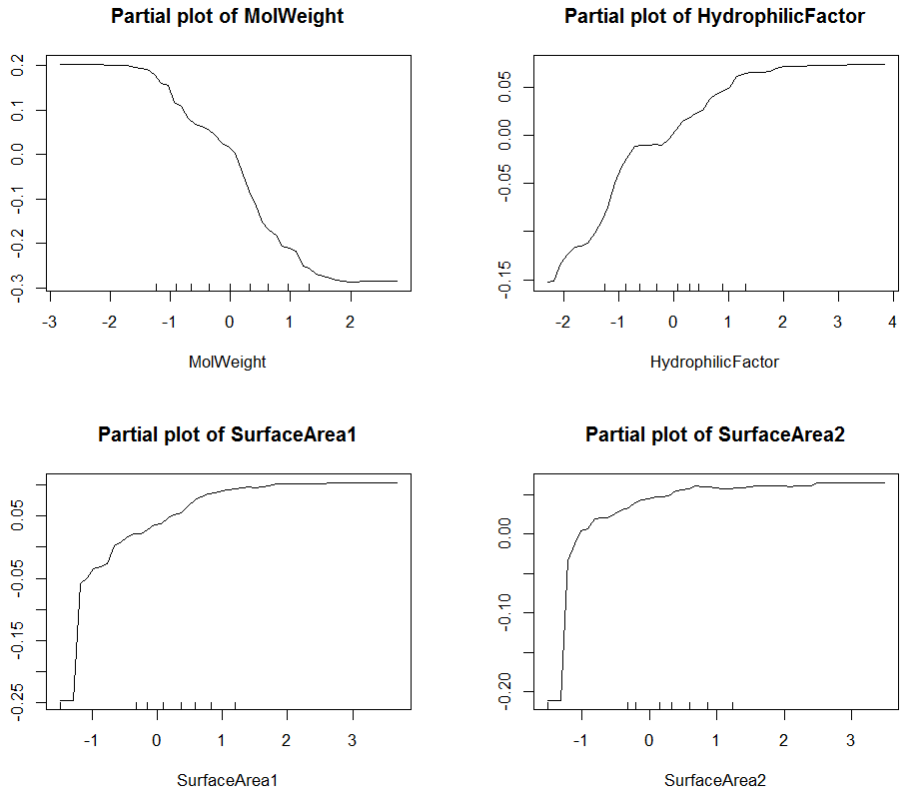
[5 marks]

7. *Draw a partial dependence plot for each of the four continuous descriptors. How do these variables relate to solubility?*

The following code produces the plots:

```
par(mfrow=c(2,2))
partialPlot(sol40.rf, x.var = MolWeight,
pred.data=Solubility.df, main = "Partial plot of MolWeight" )
partialPlot(sol40.rf, x.var = HydrophilicFactor,
pred.data=Solubility.df, main = "Partial plot of HydrophilicFactor" )
partialPlot(sol40.rf, x.var = SurfaceArea1,
pred.data=Solubility.df, main = "Partial plot of SurfaceArea1" )
partialPlot(sol40.rf, x.var = SurfaceArea2,
pred.data=Solubility.df, main = "Partial plot of SurfaceArea2" )
```

The plots are shown below. We see that as Molweight goes up, solubility goes down, and as HydrophilicFactor and SurfaceArea go up, solubility also goes up.



[5 marks]