Case 6: Wine Selection Case

Dr. Avery Haviv

Winter, 2018

This case is targetted towards MS students, though interested MBA students are free to read through it. It focuses on a dataset that documents the attributes and quality of Portuguese white wines. Details of the dataset can be found here: https://archive.ics.uci.edu/ml/datasets/wine

To motivate the data analysis, consider the following context. You work for a major sit down chain restaurant (think Olive Garden or Outback Steak House). To select a wine list, you would like to know how much consumers like different types of wine. However, you cannot survey consumers on all possible wines because of cost.

Instead, you receive the chemical properties of wines that have been served at your restaurant in previous years, along with consumers rating of that wine on a one to ten scale (stored in `quality`). For next year, you plan to find out these chemical attributes before selecting which 40 wines to purchase.

Load and Explore the Data

Following the data analysis guide, we first load and check the data:

```
wineData = read.csv('D:/Dropbox/Teaching Lectures/Wine Case/Wine Case Data.csv')
names(wineData)
```

```
##  [1] "fixed.acidity"        "volatile.acidity"     "citric.acid"
##  [4] "residual.sugar"       "chlorides"            "free.sulfur.dioxide"
##  [7] "total.sulfur.dioxide" "density"              "pH"
## [10] "sulphates"            "alcohol"              "quality"
```

```
summary(wineData)
```

```
##  fixed.acidity    volatile.acidity  citric.acid     residual.sugar
##  Min.   : 3.800   Min.   :0.0800   Min.   :0.0000   Min.   : 0.600
##  1st Qu.: 6.300   1st Qu.:0.2100   1st Qu.:0.2700   1st Qu.: 1.700
##  Median : 6.800   Median :0.2600   Median :0.3200   Median : 5.200
##  Mean   : 6.855   Mean   :0.2782   Mean   :0.3342   Mean   : 6.391
##  3rd Qu.: 7.300   3rd Qu.:0.3200   3rd Qu.:0.3900   3rd Qu.: 9.900
##  Max.   :14.200   Max.   :1.1000   Max.   :1.6600   Max.   :65.800
##    chlorides       free.sulfur.dioxide total.sulfur.dioxide
##  Min.   :0.00900   Min.   :  2.00      Min.   :  9.0
##  1st Qu.:0.03600   1st Qu.: 23.00      1st Qu.:108.0
##  Median :0.04300   Median : 34.00      Median :134.0
##  Mean   :0.04577   Mean   : 35.31      Mean   :138.4
##  3rd Qu.:0.05000   3rd Qu.: 46.00      3rd Qu.:167.0
##  Max.   :0.34600   Max.   :289.00      Max.   :440.0
##     density             pH          sulphates         alcohol
##  Min.   :0.9871   Min.   :2.720   Min.   :0.2200   Min.   : 8.00
##  1st Qu.:0.9917   1st Qu.:3.090   1st Qu.:0.4100   1st Qu.: 9.50
##  Median :0.9937   Median :3.180   Median :0.4700   Median :10.40
##  Mean   :0.9940   Mean   :3.188   Mean   :0.4898   Mean   :10.51
##  3rd Qu.:0.9961   3rd Qu.:3.280   3rd Qu.:0.5500   3rd Qu.:11.40
##  Max.   :1.0390   Max.   :3.820   Max.   :1.0800   Max.   :14.20
##     quality
```

```
##  Min.   :3.000
##  1st Qu.:5.000
##  Median :6.000
##  Mean   :5.878
##  3rd Qu.:6.000
##  Max.   :9.000
```

```
str(wineData)
```

```
## 'data.frame':    4898 obs. of  12 variables:
##  $ fixed.acidity       : num  7 6.3 8.1 7.2 7.2 8.1 6.2 7 6.3 8.1 ...
##  $ volatile.acidity    : num  0.27 0.3 0.28 0.23 0.23 0.28 0.32 0.27 0.3 0.22 ...
##  $ citric.acid         : num  0.36 0.34 0.4 0.32 0.32 0.4 0.16 0.36 0.34 0.43 ...
##  $ residual.sugar      : num  20.7 1.6 6.9 8.5 8.5 6.9 7 20.7 1.6 1.5 ...
##  $ chlorides           : num  0.045 0.049 0.05 0.058 0.058 0.05 0.045 0.045 0.049 0.044 ...
##  $ free.sulfur.dioxide : num  45 14 30 47 47 30 30 45 14 28 ...
##  $ total.sulfur.dioxide: num  170 132 97 186 186 97 136 170 132 129 ...
##  $ density             : num  1.001 0.994 0.995 0.996 0.996 ...
##  $ pH                  : num  3 3.3 3.26 3.19 3.19 3.26 3.18 3 3.3 3.22 ...
##  $ sulphates           : num  0.45 0.49 0.44 0.4 0.4 0.44 0.47 0.45 0.49 0.45 ...
##  $ alcohol             : num  8.8 9.5 10.1 9.9 9.9 10.1 9.6 8.8 9.5 11 ...
##  $ quality             : int  6 6 6 6 6 6 6 6 6 6 ...
```

We have some predictor variables for `quality`, which is our target variable. Quality ranges between 3 and 9 out of 10.

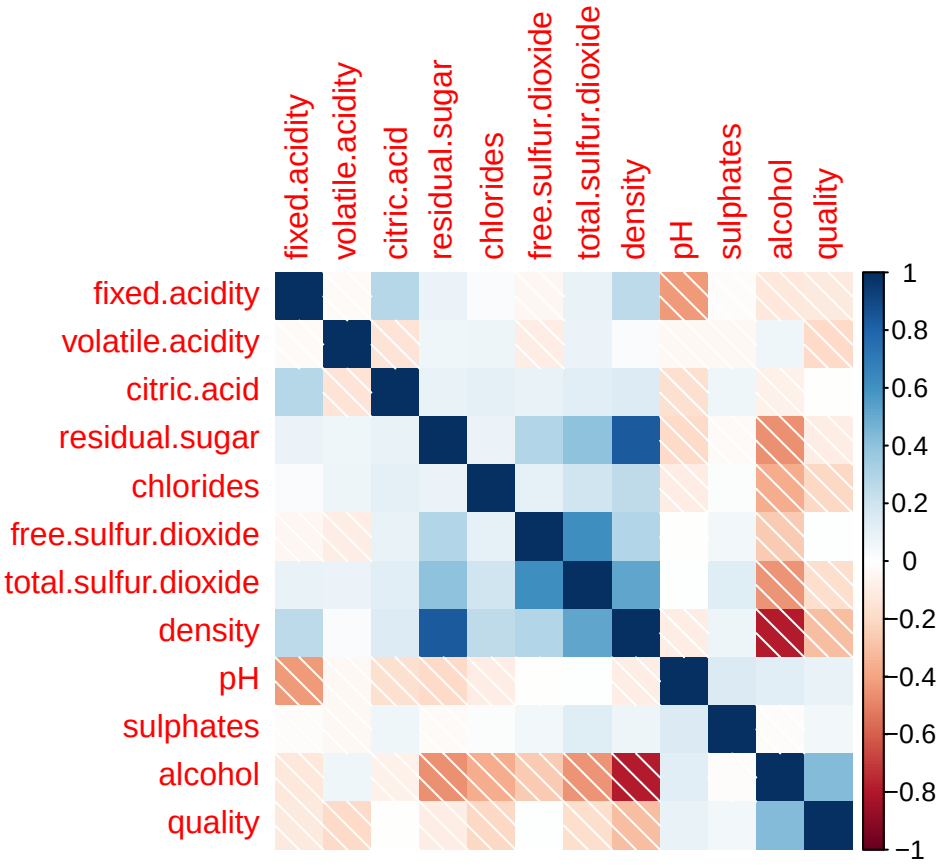
Perform a Descriptive Analysis

Next, we perform a descriptive analysis to learn more about our context. We start with a correlation plot.

```
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```
corrplot(cor(wineData),method='shade')
```



There is a strong set of relationships between alcohol, density, and residual sugar. Most factors (except alcohol) have a negative relationship with quality. We run a simple regression:

```
summary(lm(quality~.,data=wineData))
```

```
##
## Call:
## lm(formula = quality ~ ., data = wineData)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.8348 -0.4934 -0.0379  0.4637  3.1143
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)           1.502e+02  1.880e+01   7.987 1.71e-15 ***
## fixed.acidity         6.552e-02  2.087e-02   3.139  0.00171 **
## volatile.acidity     -1.863e+00  1.138e-01 -16.373  < 2e-16 ***
## citric.acid           2.209e-02  9.577e-02   0.231  0.81759
## residual.sugar        8.148e-02  7.527e-03  10.825  < 2e-16 ***
## chlorides            -2.473e-01  5.465e-01  -0.452  0.65097
## free.sulfur.dioxide   3.733e-03  8.441e-04   4.422 9.99e-06 ***
## total.sulfur.dioxide -2.857e-04  3.781e-04  -0.756  0.44979
## density              -1.503e+02  1.907e+01  -7.879 4.04e-15 ***
## pH                    6.863e-01  1.054e-01   6.513 8.10e-11 ***
## sulphates             6.315e-01  1.004e-01   6.291 3.44e-10 ***
## alcohol               1.935e-01  2.422e-02   7.988 1.70e-15 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7514 on 4886 degrees of freedom
## Multiple R-squared:  0.2819, Adjusted R-squared:  0.2803
## F-statistic: 174.3 on 11 and 4886 DF,  p-value: < 2.2e-16
```

Note that fixed acidity has a positive regression coefficient, but a negative initial correlation. The sign changed because we are controlling for more other variables in the regression.

We can create all possible scatterplots using the `pairs` function, though I have omitted it to reduce the file size. Note that there might be a potential outlier, as one wine has a far greater `density` than all others.

Continuing the descriptive analysis, we can now fit a one and two dimensional `earth` model.

```
library('earth')
```

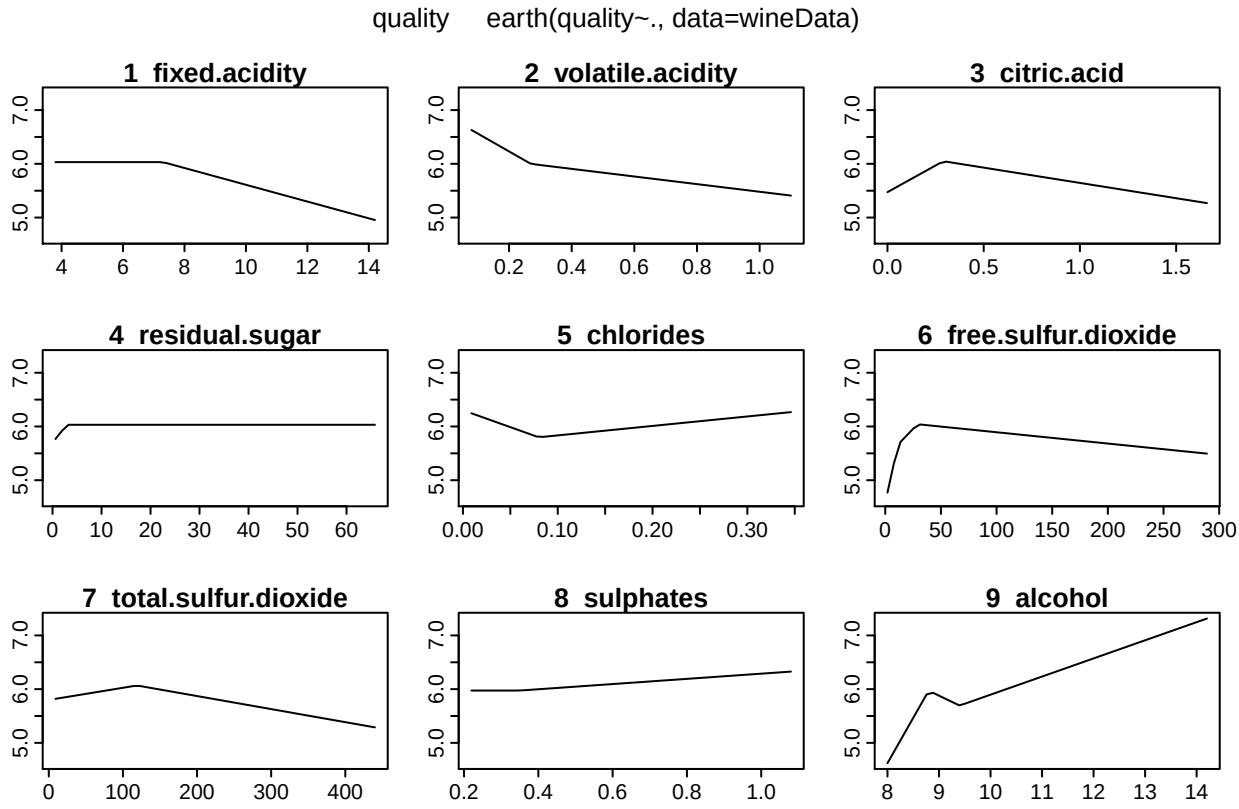
```
## Loading required package: plotmo
```

```
## Loading required package: plotrix
```

```
## Loading required package: TeachingDemos
```

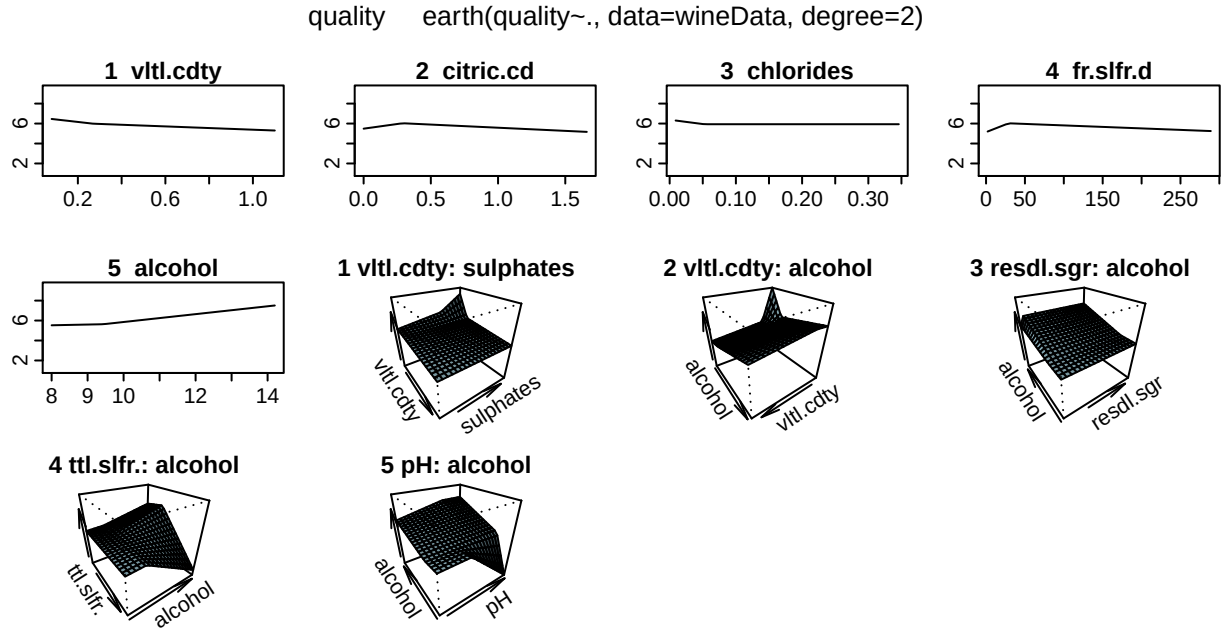
```
plotmo(earth(quality~.,dat=wineData))
```

```
##  plotmo grid:    fixed.acidity volatile.acidity citric.acid residual.sugar
##                            6.8             0.26        0.32            5.2
##  chlorides free.sulfur.dioxide total.sulfur.dioxide density   pH sulphates
##      0.043                  34                  134 0.99374 3.18      0.47
##  alcohol
##     10.4
```




```
plotmo(earth(quality~.,dat=wineData,degree=2))
```

```
##  plotmo grid:    fixed.acidity volatile.acidity citric.acid residual.sugar
##                            6.8              0.26        0.32            5.2
##  chlorides free.sulfur.dioxide total.sulfur.dioxide density   pH sulphates
##      0.043                  34                  134 0.99374 3.18      0.47
##  alcohol
##     10.4
```



I would note two things at this stage:

1. A few variables have non-linear relationships with quality. This is particularly important as both `citric.acid` and `free.sulfur.dioxide` might appear to have no relationship with `quality` if we do not allow for non-linearities
2. Alcohol content is quite important in predicting wine quality (I suppose this is not surprising...). Most of the important interactions are with alcohol. We can specifically consider these interactions in future analysis.

Type of Question

Next, we must figure out if this is a predictive or causal question. Are we fundamentally trying to the quality of a particular wine? Or do we only want to know what the quality is, without impacting any of the independent variables?

In terms of R code, will we be making predictions on data that is given to us? Or will we construct a new dataset (like we did when pricing)?

This particular case is a predictive question. The restaurant does not make wine, it purchases it. It cannot affect the quality of wines directly, and will not be manipulating any of the attributes of the wine. The model will be applied to new wines, not any data we construct.

We would then move to the predictive portion of the data analysis guide.

Improve the Data

An important part of any predictive problem is the dataset that is being used. In this case, I have provided the full set of variables that were in the dataset. However, in a business context, you should always consider if you could acquire more data that might make the predictive problem easier. Recall Homework 3 - introducing new columns improves predictions a lot more than model training.

In this context, what variables do you think would be useful predictors of wine quality? How would you obtain those?

Cross Validation

Given that this is a predictive problem, the goal is to find the very best predictive model that we can. We must use cross-validation to determine which model is the best. I use standard cross-validation in this case, implementing K-Fold cross-validation would be an improvement.

```
#Generate Training and Validation Datasets
set.seed(1) #This is an arbitrary number
isTraining = runif(nrow(wineData)) < .8
isValidation = !isTraining

trainingData = subset(wineData,isTraining)
validationData = subset(wineData,isValidation)
```

To get the best model I will need to try many. Therefore, I want code that allows me to test models quickly and correctly. The function below returns the root mean squared error (RMSE) for a model in the validation data.

```
getWineRMSE = function(model){
  prediction = predict(model,validationData)
  #Take the errors, square them, take the mean, then take the square root
  #to calculate the RMSE
  return(mean((validationData$quality-prediction)^2)^.5)
}
```

With this function in hand, we can start training the predictive model. Has a relatively large number of rows (just under 5000) and a relatively small number of columns (11 independent variables total). Therefore, I would expect in the bottom left quadrant of the Predictive Model Guidelines to perform well here.

As a baseline, we can run regressions with different number of interactions:

```
getWineRMSE(lm(quality~.,data=trainingData))
```

```
## [1] 0.7683738
```

```
getWineRMSE(lm(quality~.^2,data=trainingData))
```

```
## [1] 0.7643278
```

```
getWineRMSE(lm(quality~.^3,data=trainingData))
```

```
## [1] 0.8055322
```

```
getWineRMSE(lm(quality~.^4,data=trainingData))
```

```
## [1] 1.00857
```

We can see that second degree interactions seem worthwhile. An ANOVA tells us if any variables can be removed from the analysis.

```
anova(lm(quality~.,data=trainingData))
```

```
## Analysis of Variance Table
##
## Response: quality
##                        Df  Sum Sq Mean Sq  F value    Pr(>F)
## fixed.acidity           1   35.43   35.43  63.4333 2.164e-15 ***
## volatile.acidity        1  115.88  115.88 207.4653 < 2.2e-16 ***
## citric.acid             1    0.00    0.00   0.0054  0.941696
## residual.sugar          1   15.41   15.41  27.5820 1.587e-07 ***
## chlorides               1  108.68  108.68 194.5786 < 2.2e-16 ***
## free.sulfur.dioxide     1    3.74    3.74   6.6887  0.009739 **
## total.sulfur.dioxide    1   45.33   45.33  81.1520 < 2.2e-16 ***
## density                 1  343.44  343.44 614.8936 < 2.2e-16 ***
## pH                      1   90.07   90.07 161.2554 < 2.2e-16 ***
## sulphates               1   25.25   25.25  45.2024 2.038e-11 ***
## alcohol                 1   31.13   31.13  55.7282 1.023e-13 ***
## Residuals            3860 2155.92    0.56
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Citric acid was insignificant here, so we can try removing this variable:

```
getWineRMSE(lm(quality~.-citric.acid,data=trainingData))
```

```
## [1] 0.7683661
```

```
getWineRMSE(lm(quality~(.-citric.acid)^2,data=trainingData))
```

```
## [1] 0.7641658
```

```
getWineRMSE(lm(quality~(.-citric.acid)^3,data=trainingData))
```

```
## [1] 0.7622935
```

```
getWineRMSE(lm(quality~(.-citric.acid)^4,data=trainingData))
```

```
## [1] 0.8393603
```

That did improve things, as the first model is now the strongest. However, keep in mind that in the descriptive analysis there was a non-linear relationship between `citric.acid` and `quality`, therefore we might keep it in for now, but we can always consider removing it.

Now, lets try the MARS model:

```
getWineRMSE(earth(quality~.,data=trainingData))
```

```
## [1] 0.7686575
```

```
getWineRMSE(earth(quality~.,degree=2,data=trainingData))
```

```
## [1] 0.711183
```

```
getWineRMSE(earth(quality~.,degree=3,data=trainingData))
```

```
## [1] 0.7062887
```

```
getWineRMSE(earth(quality~.,degree=4,data=trainingData))
```

```
## [1] 0.7062887
```

MARS with high order interactions outperforms linear regression. This makes some sense: We have a lot of data, so we can estimate a large number of coefficients precisely. If we had less data, the regression/lower interactions would likely perform better.

Next, we can adjust `thresh` which is a tuning parameter that has a default value of 0.001. I adjust it to higher and lower values.

```
getWineRMSE(earth(quality~.,degree=3,data=trainingData,thresh=.01))
```

```
## [1] 0.726966
```

```
getWineRMSE(earth(quality~.,degree=3,data=trainingData,thresh=.000001))
```

```
## [1] 0.7062887
```

Neither improved the fit. We could spend longer training the model formula, however, instead, let's try a Random Forest.

I will be using the `ranger` package instead of the `randomForest` package. They are very similar. However, `ranger` is much faster, particularly on a multi-core machine.

```
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(ranger)
```

```
##
## Attaching package: 'ranger'
```

```
## The following object is masked from 'package:randomForest':
##
##     importance
```

```
#It helps to sort columns alphabetically first
trainingData = trainingData[,order(colnames(trainingData))]
validationData = validationData[,order(colnames(validationData))]

#Using the ranger package requires some adjustment to our predictive function
getWineRMSERanger = function(model){
  #Including interactions in ranger is a bit tricky, so I need to generate the predictors
  #for the validation set myself.  Without interactions, you can use the same predict
  #function as before
  predictionMat = model.matrix(as.formula(paste0('quality~',
paste(model$forest$independent.variable.names,collapse='+'))),data=validationData)

  prediction = predict(model,predictionMat)$prediction

  #Take the errors, square them, take the mean, then take the square
  #root to calculate the RMSE
  return(mean((validationData$quality-prediction)^2)^.5)
}

system.time(randomForest(quality~.,data=trainingData,ntree=1000))
```

```
##    user  system elapsed
##   16.24    0.00   16.24
```

```
system.time(ranger(quality~.,data=trainingData,num.trees=1000))
```

```
##    user  system elapsed
##    6.22    0.39    0.23
```

On my computer, `ranger` is almost 100 times faster, but the difference will be smaller on your computers. We go through the tunning process for the random forest in a similar manner. Some packages do provide some methods to tune parameters automatically (`tuneRF` and `rfcv` in the `randomForest` package), but these only train specific tuning parameters, and will not work on the others.

In an application, we should set `num.trees`, which controls the number of trees in the forest, to be quite high. Without enough trees, our out of sample estimation results have some statistical noise, so it makes comparing models difficult. By default, random forest estimates with 500 trees, but more would be better! Estimating more trees will generally improve model fit, but it will increase computational time.

How many do we need? Well, we can keep increasing the number of trees until the resulting RMSE is consistent.

```
#See if 1000 is enough
getWineRMSERanger(ranger(quality~.,data=trainingData,num.trees=1000))
```

```
## [1] 0.6122517
```

```
getWineRMSERanger(ranger(quality~.,data=trainingData,num.trees=1000))
```

```
## [1] 0.6131978
```

```
#Numbers are different, increasing to 5000
getWineRMSERanger(ranger(quality~.,data=trainingData,num.trees=5000))
```

```
## [1] 0.6128182
```

```
getWineRMSERanger(ranger(quality~.,data=trainingData,num.trees=5000))
```

```
## [1] 0.6121938
```

```
#Still some noise
getWineRMSERanger(ranger(quality~.,data=trainingData,num.trees=20000))
```

```
## [1] 0.612076
```

```
getWineRMSERanger(ranger(quality~.,data=trainingData,num.trees=20000))
```

```
## [1] 0.6121567
```

```
#20000 seems good
```

We will use 20000 trees going forward. Trees perform extremely well here! This is likely because of the large amount of data, and a relatively small number of predictors. Given this fit, I will train `mtry`, which controls how many variables are in each tree.

```
getWineRMSERanger(ranger(quality~.^2,data=trainingData,num.trees=20000))
```

```
## [1] 0.6071845
```

```
getWineRMSERanger(ranger(quality~.^2,data=trainingData,mtry=3,num.trees=20000))
```

```
## [1] 0.6107382
```

```
getWineRMSERanger(ranger(quality~.^2,data=trainingData,mtry=5,num.trees=20000))
```

```
## [1] 0.6082432
```

```
getWineRMSERanger(ranger(quality~.^2,data=trainingData,mtry=9,num.trees=20000))
```

```
## [1] 0.6071361
```

```
getWineRMSERanger(ranger(quality~.^2,data=trainingData,mtry=11,num.trees=20000))
```

```
## [1] 0.6071444
```

```
getWineRMSERanger(ranger(quality~.^2,data=trainingData,mtry=15,num.trees=20000))
```

```
## [1] 0.6071911
```

The interacted variables helped. Furthermore, since 5 worked better than 3, I tried larger values. I also try to train `min.node.size`. The default is 5, so I will try values both above and below this.

```
getWineRMSERanger(ranger(quality~.^2,data=trainingData,mtry=11,min.node.size=2,num.trees=20000))
```

```
## [1] 0.6044492
```

```
getWineRMSERanger(ranger(quality~.^2,data=trainingData,mtry=11,min.node.size=8,num.trees=20000))
```

```
## [1] 0.610361
```

Min node size of 2 did best, so I am going to try 1 and 3.

```
getWineRMSERanger(ranger(quality~.^2,data=trainingData,mtry=11,min.node.size=1,num.trees=20000))
```

```
## [1] 0.6041492
```

```
getWineRMSERanger(ranger(quality~.^2,data=trainingData,mtry=11,min.node.size=3,num.trees=20000))
```

```
## [1] 0.6054619
```

One works well.

Variable Selection

One approach to generating a good model formula is to select the terms analytically. A simple, but popular way (even in high in predictive contests) is to look at the univariate correlations between predictor variables and the target variable, and only include those which cross some threshold. To do this efficiently, I would write it into a function. As before, I will show you what I would consider good code for this to be:

```
getSelectedVars = function(threshold){
  #Generate all model terms with a 2-degree interaction.
  #The -1 removes the constant since we always want that
  fullModelMatrix = model.matrix(quality~-1+.^2,data=trainingData)

  #Check the univariate correlation with quality, our Y variable
  corWithQuality = cor(trainingData$quality,fullModelMatrix)

  #Get the variable names of those with a correlation above the threshold
  goodVars = colnames(fullModelMatrix)[abs(corWithQuality)>threshold]

  #Generate the model formula, using the combination of goodVars and
  #varSelectedFormula = as.formula(paste0('quality~alcohol*(',paste(goodVars,collapse='+'),')'))
  varSelectedFormula = as.formula(paste0('quality~',paste(goodVars,collapse='+')))
  return(varSelectedFormula)
}
#Call the function to get a formula
```

```
varSelectedFormula = getSelectedVars(.1)
varSelectedFormula
```

```
## quality ~ alcohol + chlorides + density + fixed.acidity
+ pH +
## total.sulfur.dioxide + volatile.acidity +
alcohol:chlorides +
## alcohol:citric.acid + alcohol:density +
alcohol:fixed.acidity +
## alcohol:free.sulfur.dioxide + alcohol:pH +
alcohol:sulphates +
## chlorides:citric.acid + chlorides:density +
chlorides:fixed.acidity +
## chlorides:free.sulfur.dioxide + chlorides:pH +
chlorides:residual.sugar +
## chlorides:sulphates + chlorides:total.sulfur.dioxide +
chlorides:volatile.acidity +
## citric.acid:total.sulfur.dioxide +
citric.acid:volatile.acidity +
## density:fixed.acidity + density:total.sulfur.dioxide +
density:volatile.acidity +
## fixed.acidity:total.sulfur.dioxide +
fixed.acidity:volatile.acidity +
## pH:total.sulfur.dioxide + pH:volatile.acidity +
residual.sugar:total.sulfur.dioxide +
## residual.sugar:volatile.acidity +
sulphates:total.sulfur.dioxide +
## sulphates:volatile.acidity +
total.sulfur.dioxide:volatile.acidity
## <environment: 0x000000001ea94368>
```

Note that this function brings model terms together using the `paste` function. This can be a very useful way to write out long formulas, which I use a lot when writing my own code.

Given that we have changed the model formula so much, we need to retune the model, but let's start at our previous values.

```
varSelectedFormula = getSelectedVars(.1)
getWineRMSERanger(ranger(varSelectedFormula,data=trainingData,mtry=11,
                         min.node.size=1,num.trees=20000))
```

```
## [1] 0.6104752
```

```
varSelectedFormula = getSelectedVars(.05)
getWineRMSERanger(ranger(varSelectedFormula,data=trainingData,mtry=11,
                         min.node.size=1,num.trees=20000))
```

```
## [1] 0.6056722
```

```
varSelectedFormula = getSelectedVars(.01)
getWineRMSERanger(ranger(varSelectedFormula,data=trainingData,mtry=11,
                         min.node.size=1,num.trees=20000))
```

```
## [1] 0.6043647
```

```
varSelectedFormula = getSelectedVars(.005)
getWineRMSERanger(ranger(varSelectedFormula,data=trainingData,mtry=11,
                         min.node.size=1,num.trees=20000))
```

```
## [1] 0.6037695
```

```
bestTrainModel = ranger(varSelectedFormula,data=trainingData,mtry=11,
                        min.node.size=1,num.trees=20000)
```

We could continue the tuning process from here. Offline, I also trained a neural network, but it performed similarly to MARS. Assuming the 0.01 model was best (after training), we can often improve the fit further by retuning the model parameters.

Lets call it `chosenModel`:

```
chosenModel = ranger(varSelectedFormula,data=wineData,mtry=11,min.node.size=1,num.trees=20000)
```

We can see the gains of our predictive exercise in the validation set. Lets compare the quality of the top 40 wines according to a regression and our trained random forest model:

```
lmOrder = order(predict(lm(quality~.,data=trainingData),validationData),decreasing=TRUE)[1:20]
#We use bestTrainModel since we are evaluating in the validation set



predictionMat = model.matrix(as.formula(paste0('quality~',
paste(bestTrainModel$forest$independent.variable.names,collapse='+'))),data=validationData)
bestOrder = order(predict(bestTrainModel,predictionMat)$prediction,decreasing=TRUE)[1:20]

#Average quality for the best wines from each model:
mean(validationData$quality[lmOrder])
```

```
## [1] 6.7
```

```
mean(validationData$quality[bestOrder])
```

```
## [1] 7.8
```

Improving the model fit improves our selected wine by more than a point, which is substantial given the relatively limited variation in quality. Only 42 of the 997 wines in the validation data had a score of 8 or higher. With 20 guesses, our trained model picked 17 of them, whereas the regression only got 3.

```
sum(validationData$quality>=8)
```

```
## [1] 42
```

```
sum(validationData$quality[lmOrder]>=8)
```

```
## [1] 3
```

```
sum(validationData$quality[bestOrder]>=8)
```

```
## [1] 17
```

That seems like a result worth drinking too!