Predicting Wine Quality with Chemical Properties

Samantha Hunter, Shih-Ni Prim, Sara Stegemoller, Janel Warren

2021-04-27

1 Introduction

For the final project, we use a data set for red wine that has 1599 rows and 12 columns, 11 predictors and one response variable, with each row representing one variety of red wine. The predictors are chemical measurements of the wine, such as alcohol or sulphates. Please see Table 1 for an overview of the variables, including the minimums, medians, means, and maximums. For classification methods, we collapse quality into a two-level categorical variable (levels: "low" and "high") and a three-level categorical variable (levels: "L," "M," and "H") (see Table 2 for frequency counts). Before we start the analysis, we separate the data set into a training set (80%) and a test set (20%). Because our data set is fairly large, we hope that, by using 80% for the training set, we will have an accurate model and that 20% of the data contains enough observations to reveal the shortcoming and triumphs of our models. For methods that require tuning parameters, we use cross-validation on the training set and then use the best model to make predictions on the test set. We use test MSE or misclassification/accuracy rates to evaluate the models.

1.1 Research Question

In this project, we use statistical methods to answer our research question: is it possible to predict the quality of a wine using chemical properties? This is an important question for the wine industry, as hiring tasters is expensive, with their answers potentially unreliable, and using chemical properties to make predictions can be cost-saving. Although taste is subjective, we hypothesize that the general quality of a wine should be possible to predict, since chemical properties such as alcohol level, amount of sugar, or citric acid can drive taste. If the answer is indeed positive, we will recommend methods. If the answer is negative, we will explain the reasons.

1.2 Exploratory Data Analysis

Before we begin, we need to confront the limitations of our data set. The response variable quality takes on integer values, but we do not know the scale of the integers. It could be that the distance between 3 and 4 is equal to that between 4 and 5, or it could be that 4 is closer in quality to 3 than it is to 5, if quality is based on a logarithmic or exponential scale. Another limitation to predicting quality is its distribution. While the distribution is normal, our low quality, 3, only contains 10 observations and high quality, 8, only contains 18 observations (see Table 2). Due to lack of data, this may make predictions difficult.

Histograms of all predictors show that the distributions are mostly normal or right-skewed. We also checked correlations between the variables and noticed several pairs of variables with correlations over 0.6: density vs fixed acidity, fixed acidity vs citric acid, fixed sulfur dioxide vs total sulfur dioxide, and fixed acidity vs pH. The variable fixed sulfur dioxide appears in several of the pairs, so we will keep in mind that it correlates with several other predictors.

Table 1: Summary Statistics

variable name	min	median	mean	max
fixed acidity	4.6	7.9	8.32	15.9
volatile acidity	0.12	0.52	0.528	1.58
citric acid	0	0.26	0.271	1
residual sugar	0.9	2.2	2.539	15.5
chlorides	0.012	0.079	0.087	0.611
fixed sulfur dioxide	1	14	15.87	72
total sulfur dioxide	6	38	46.47	289
density	0.99	0.997	0.997	1.004
pH	2.74	3.31	3.311	4.01
sulphates	0.33	0.62	0.658	2
alcohol	8.4	10.2	10.42	14.9
quality	3	6	5.636	8

2 Classification Methods

2.1 Introduction

As Table 2 shows, the response variable quality is numeric and has six unique values: 3, 4, 5, 6, 7, and 8. Its distribution is normal with the majority (1319 of 1599) having values of 5 or 6 and less at the ends. To proceed with classification methods, we collapse the six values into a variable with two levels and one with three levels, as it would be difficult to predict six classes. Given that we do not know how these values correspond to quality, our decision is guided by intuition and the principle that it is beneficial to have enough data points in each category. We use the different wordings, "low-high" versus "L-M-H," to distinguish between the two- and three-level categorical response variables.

Table 2: Frequency Counts for Response Variables

	3	4	5	6	7	8
6-level	10	53	681	638	199	18
	low			high		
2-level	744			855		
	L		M		H	
3-level	63		1319		217	

With the two-level variable, we have more balanced numbers of data points in each level. Nonetheless, the three-level variable may lead to better predictions, as it has a higher no-information rate, which is the proportion of the biggest group in the training set. Specifically, if we simply predict the two-level response variable as "high," the accuracy rate would be 0.525. If we predict the three-level response variable as "M," the accuracy rate would be 0.821. The no-information rates provide the baseline rates, and we aim to find models that achieve higher test accuracy rates.

We proceed with our analysis on both variables to determine the more suitable categorical variable. For each model, we train on the training set, make predictions on the test set, and calculate test accuracy rates to evaluate the model's performance. We start with classification trees (unpruned and pruned) for their interpretability and graduate to ensemble learning methods: bagged tree, random forest, and boosted trees. These latter methods tend to have better performance than classification trees, but at the cost of interpretability.

2.2 Classification Tree

Classification trees do not necessarily provide good predictions, but the nature of a tree mirrors the decision making process of a human and thus easy to understand and interpret. The tree structure also takes care of interactions between predictors. The accuracy rate for the two-level variable (0.684) is better than the no-information rate, but that for the three-level variable (0.828) is about the same as the no-information rate. The confusion matrices in Table 3 also show the accuracy rates by category, which tell a different story than the overall accuracy rates. The accuracy rate is over 90% for "M" in the three-level variable but much lower for "H" and zero for "L." For the two-level variable, the overall accuracy rate is lower, but the performance across categories is more even. We will see this trend across all methods.

Table 3: Confusion Matrix for Classificaiton Tree with 2-level Variable (left) and 3-level Variable (right)

	Actual		
Predicted	low	high	Pred Rate
low	100	64	0.73
high	37	119	0.65

	Actual			
Predicted	L	M	H	Pred Rate
L	0	0	0	0
M	8	250	28	0.929
H	0	19	15	0.349

To avoid overfitting, we now use pruning, which can potentially improve results because of the bias-variance trade-off. Smaller trees have higher bias and lower variance, and larger trees have lower bias and higher variance; we want to seek out a balance between bias and variance to minimize the test error rate. We use cross-validation guided by the misclassification rate to select the optimal tree size. Cross-validation shows that the optimal tree size for the two-level variable is 10, which is exactly the number of nodes in the unpruned tree. Thus, we do not prune it. The optimal tree size for the three-level variable is 4 terminal nodes, while our unpruned tree has 11 nodes. We next prune the tree for the three-level variable.

Table 4: Confusion Matrix for Pruned Classificaiton Tree with 3-level Variable

	Actual			
Predicted	L	M	H	Pred Rate
L	0	0	0	0
M	8	258	28	0.959
H	0	11	15	0.349

The pruned tree's overall test accuracy rate is 0.853. This is better than the accuracy rate for the unpruned tree (0.828) and the no-information rate for three-level variables (0.821), likely due to the pruning process removing unwanted noise from unimportant variables and reducing the effects of overfitting. Thus, we treat the unpruned tree for the two-level variable and the pruned tree for the three-level variable as our final classification tree models. In Table 4, we again see the high accuracy rate for the "M" category but zero for the "L" category. In Figure 1, the tree plot for the two-level variable shows some interaction between predictors. Though it is clear alcohol is the most important predictor, it is hard to conclude exactly how alcohol level is associated with ratings. In two of the terminal leaves, we can see that lower chlorides is associated with higher ratings. The pruned tree for the three-level variable is much simpler, as only alcohol and sulphates are used to determine quality. The tree plot shows that higher alcohol level is associated with higher ratings.

2.3 Bagged Tree

Bootstrap Aggregating (bagging) uses bootstrap samples from the training data sets to average together many different trees. With bagging, it is possible to reduce variance without increasing bias, because the

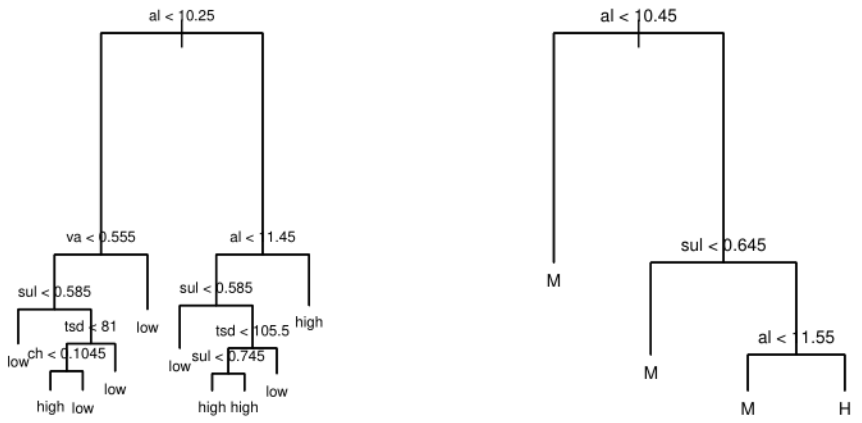

Figure 1: Plots for Unpruned Tree with 2-level Variable (left) and Pruned Tree with 3-level Variable (right)

final predictions are based on the averaged predictions or majority vote across different trees. The overall test accuracy rate is 0.766 for the two-level variable and 0.881 for the three-level variable, the best so far. The confusion matrices in Table 5 show that the accuracy rates by category are also better than the classification trees, and the "H" category in the three-level variable improves the most, from a 0.349 prediction rate to 0.512.

Table 5: Confusion Matrix for Bagged Tree with 2-level Variable (left) and 3-level Variable (right)

	Actual		
Predicted	low	high	Pred Rate
low	109	47	0.796
high	28	136	0.743

	Actual			
Predicted	L	M	H	Pred Rate
L	0	1	0	0
M	8	260	21	0.967
H	0	8	22	0.512

Bagging is not without limitation. We can no longer visualize these trees and, with this method, different trees can be driven by strong predictors (in our model, alcohol, sulphates, and volatile acidity) and thus can be similar to one another. Given this similarity in predictions, however, bagging may not reduce variance as much as other methods that do not use correlated predictions. In other words, the bootstrap trees are correlated, which prevent further decrease of misclassification error. To mitigate this, we can add randomness to create dissimilar trees and reduce correlation, which can be achieved by random forest.

2.4 Random Forest

Bagging is a special case of random forest that uses all predictors in each tree. Random forest, on the other hand, randomly selects a set of m predictors, typically $\sqrt{p}$, to avoid the problem of strong predictors driving the results. Random forest trees are less correlated, which can reduce variance. The plots in Figure 2 show how the accuracy rates change with the increase of number of predictors (the `mtry` hyperparameter). The highest accuracy rates with the smallest number of predictors are achieved with $mtry = 3$ for the two-level variable and $mtry = 2$ for the three-level variable. Indeed, these values are close to $\sqrt{11} = 3.32$, commonly considered the best `mtry`.

As we have seen throughout the models, the model using the three-level response variable has a higher overall test accuracy rate, 0.9, than that using the two-level response variable, 0.781. These are slightly better than the accuracy rates for bagged trees, 0.881 for three-level and 0.766 for two-level variables. This is

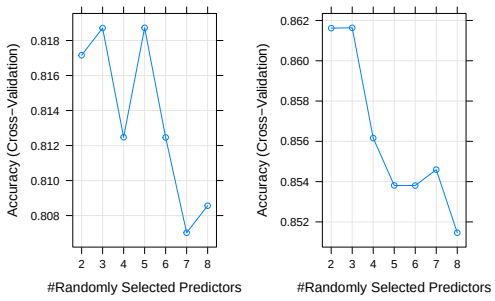

Figure 2: Accuracy Rates vs Number of Predictors Used for 2-level Variable (left) and 3-level Variable (right)

expected, since we continue to see several predictors as stronger than others, which we will discuss later. The confusion matrices in Table 6 show that the accuracy rate by category reaches 98.5% for the "M" category but remains low for "L" and "H." The accuracy rates by category for the two-level variables have also improved and are much better than the no-information rate (0.525). All in all, random forest seems a good candidate.

Table 6: Confusion Matrix for Random Forest Tree with 2-level Variable (left) and 3-level Variable (right)

	Actual		
Predicted	low	high	Pred Rate
low	111	44	0.81
high	26	139	0.76

	Actual			
Predicted	L	M	H	Pred Rate
L	0	0	0	0
M	8	265	20	0.985
H	0	4	23	0.535

2.5 Boosted Tree

Finally, we use boosting to see if we can improve our prediction rates. With boosted trees, the models are made of small trees that are fit sequentially, and so each tree tries to compensate for the previous trees, looking to improve upon areas in which previous trees did poorly. Boosting can thus reduce bias. Unlike bagging and random forest, however, boosting can overfit, but it often does so slowly, so it is possible to catch overfitting before it becomes a problem. This approach has several tuning parameters: `n.trees` (total number of trees to fit), `shrinkage` (λ, a small positive number that tells which multiple of the tree to add at different steps), `interaction.depth` (the number of levels in each tree). and `n.minobsinnode` (the minimal number of observations in each node). We use cross-validation to choose the optimal hyperparameters. The plots in Figure 3 show how the accuracy rates change with the number of boosting iterations. The best accuracy rate is achieved with 650 iterations and 4 interaction depths for the two-level variable and 250 iterations and 4 interaction depths for the three-level variable. After these points, the models likely start overfitting.

The test accuracy rate is 0.781 for the two-level variable, similar to that for the random forest model. The test accuracy rate is 0.872 for the three-level variable, worse than the corresponding models for random forest and bagged trees but similar to the pruned tree. Both overall accuracy rates are better than the no-information rates. The confusion matrices in Table 7 show that, compared to random forest, the accuracy rates by category are the same for the two-level variable and slightly lower for the three-level variable.

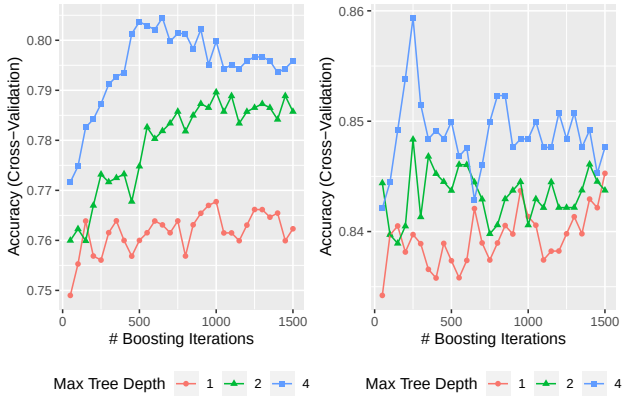

Figure 3: Iterations vs Accuracy for Boosted Trees for 2-level Variable (left) and 3-level Variable (right)

Table 7: Confusion Matrix for Boosted Tree with 2-level Variable (left) and 3-level Variable (right)

	Actual		
Predicted	low	high	Pred Rate
low	111	44	0.81
high	26	139	0.76

	Actual			
Predicted	L	M	H	Pred Rate
L	0	1	0	0
M	8	258	22	0.959
H	0	10	21	0.488

3 Regression Methods

3.1 Multiple Linear Regression (MLR)

With regression methods, we start by creating a multiple linear regression model using the same training set used in the classification methods. This additive model utilizes all 11 predictors and has a test MSE of 0.39. We then use forward subset selection, backward subset selection, and best subset selection to reduce the number of predictors used. Reducing the number of predictors can remove noise introduced by unimportant predictors, which in turn can reduce variance. All three methods find that the best number of predictors is between 6 and 8, depending on the criterion used. The test MSE is 0.39 for the six-predictor model (`Wine6`), 0.389 for the seven-predictor model (`Wine7`), and 0.389 for the eight-predictor model (`Wine8`). Reducing the number of variables, in our case, does not reduce test MSE, possibly because the number of observations, 1599, is so much larger than the number of predictors, 11. Because of this, we expect a relatively small variance to begin with, making the test MSE difficult to improve.

In the residual plots for the 11-predictor model (`WineLR`), shown in Figure 4, the Q-Q plot looks fine despite some deviation in the low and high theoretical quantiles. The leverage plot shows a few high leverage points but no outliers. The Residual vs Fitted plot, on the other hand, has a clear problem. It should have no pattern, but this plot shows six distinct lines. By plotting each integer value for quality separately on the residual plots, we find that each line on the full residual plot represents a separate quantity integer value. For example, the top most line contains observations of quality rating 8, the next line contains observations of quality rating 7, and the pattern continues until we get to the bottom line with observations of quality rating 3. Since no obvious transformations could fix this problem, we move on to a more flexible learning model.

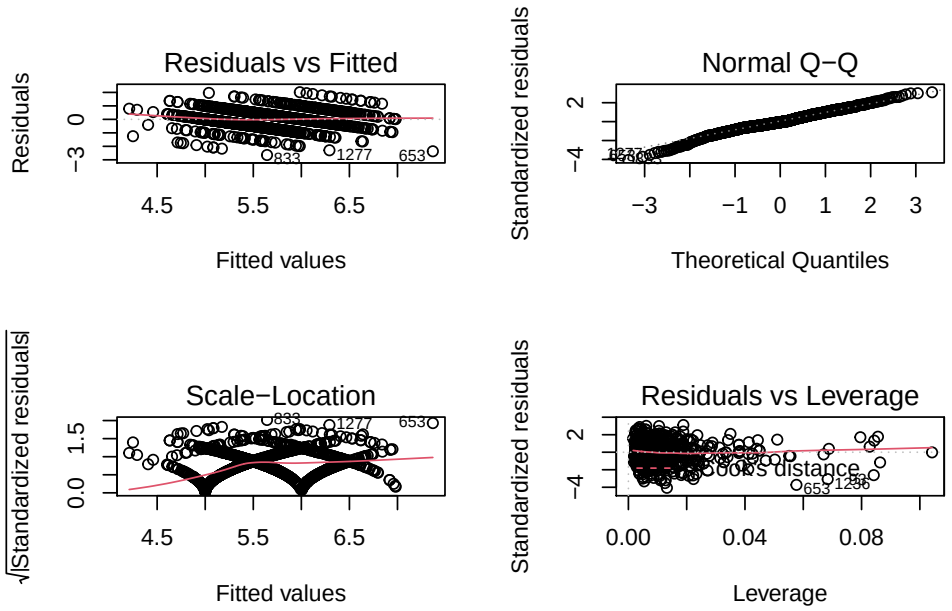

Figure 4: Plots for Linear Regresion Model with 11 Predictors

3.2 Trees: Regression Tree, Bagging, and Random Forest

For our flexible learning model, we use regression trees so that we can compare our regression results with classification results. We start with an unpruned tree that considers all 11 predictors, because the test MSE was not greatly improved by subsetting with either best subset selection or LASSO regression (see appendix for LASSO regression). The resulting tree has 10 terminal nodes and uses six predictors: alcohol, sulphates, citric acid, volatile acidity, total sulfur dioxide, and density. The test MSE is 0.443, better than the test MSE for multiple linear regression. In Figure 5, the tree plot of the unpruned regression tree has 10 terminal nodes and six variables that drive decisions. Most leaves take the values between 5 and 6.

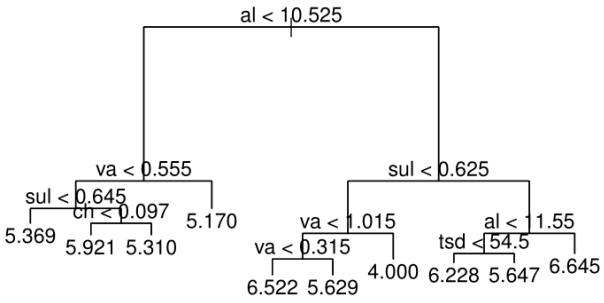

Figure 5: Tree Plot for Unpruned Regression Tree

Unfortunately, this tree only predicts values above 4 and below 7, meaning that this model will never predict the worst and best wine qualities. This is likely due to the sparse number of observations in these categories; recall, only 10 of 1599 observations have a quality rating of 3. If these observations are not extremely different from the rest, our model does not have enough information to predict a quality rating of 3 instead of 4, of which there are more training data. For easier interpretation, we could round the predictions

of leaves to integer values. To capture most of the values of wine quality, we should consider rounding down the lowest leaf value from 4.6 to 4 and the highest leaf value from 6.6 to 7. We should round the rest of leaves to the nearest integer for interpretability. Although, these recommendations may be unnecessary, because our test MSE from this unpruned tree is 0.443, worse than what we found using multiple linear regression.

Because this tree is unable to predict the upper and lower bounds of wine quality, we will not prune this tree but instead move on to bagging to improve the tree. As discussed before, this method can reduce variance by taking repeated samples with replacement from the data and averaging over the results with the downside of losing interpretability. The test MSE is 0.294, smaller than the previous regression methods.

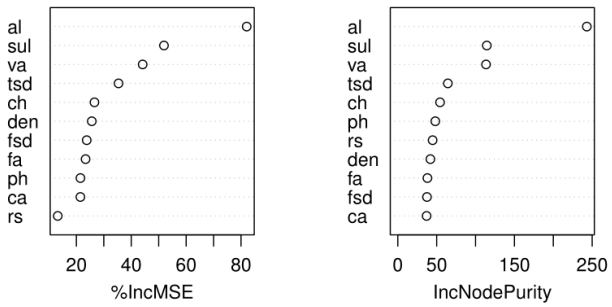

Figure 6: Variable Importance of Bagged Tree

The bagged tree could, once again, be driven by highly correlated trees; Figure 6 shows that alcohol and sulphates drive the models, which could prevent bagging from further reducing variance. To explore aggregating while controlling for correlation between trees, we next use random forest to predict quality. We set the hyperparameter `mtry` to be 3 and 4 because $mtry$ is typically $\sqrt{p}$, and we also choose $mtry = 7$ because this value is chosen by best subset selection. The test MSE is 0.289 from $mtry = 3$, 0.29 from $mtry = 4$, and 0.295 from $mtry = 7$, the best among regression models. This is consistent with the findings of our classification methods that random forest works well with the data set.

4 Discussion

4.1 Classification Methods

4.1.1 Model Comparison and Recommendations

Table 8 shows the no-information rates and overall test accuracy rates from all the classification models. The number of levels of response (2 or 3) results in larger differences in accuracy rates than does the type of methods. The three-level response models perform better than the two-level response models for all methods, possibly because observations with ratings 5 and 6 are more frequent than those with ratings 3, 4, 5 or those with ratings 6, 7, 8. In other words, collapsing observations with ratings of 5 in with lower ratings and ratings of 6 in with higher ratings does not allow the variance explained by these two ratings to be fully realized.

Table 8: Training Accuracy Rates for All Classification Models

	no-information rate	classification tree	bagged tree	random forest	boosted tree
2-level	0.525	0.684	0.766	0.781	0.781
3-level	0.821	0.853	0.881	0.900	0.872

Clearly, the random forest model using the three-level variable achieves the highest overall accuracy rate, 90%. Thus we conclude that we can indeed use chemical properties to predict wine quality. However, the best accuracy rate is only 7.9% higher than the no-information rate. The random forest model also has the highest accuracy rate for the two-level variable (78.1%), which is 25.6% higher than the no-information rate.

The most ideal model, however, depends on the winemakers' goal. If they are interested in the highest accuracy rate, we recommend using random forest on the three-level response variable. If they are interested in understanding the relationship between predictors and wine quality, interpreting the model, and saving costs for measuring chemical properties, we recommend the pruned classification tree for the three-level variable. This model only uses two predictors (alcohol and sulphates), and alcohol has to be measured if the wine is to be sold in the United States, so the only extra cost is measuring sulphates. The tree plot is extremely easy to read, and the test accuracy rate is 85.3%, 4.7% lower than our best model and only 3.2% higher than the no-information rate. It would be up to them to decide whether they can accept a lower accuracy rate in exchange for a much simpler model, interpretability, and lower cost for taking measurements.

Nonetheless, the accuracy rates by category, as seen in Table 9, offer alternative recommendations. If the winemakers care mostly about identifying medium quality wines, they should go with the three-level response variable, since the highest accuracy rate for category "M" is an impressive 98.5% for random forest. If they are interested in identifying all levels of ratings, the three-level variable has disappointing accuracy rates of 0% for "L" and about 50% for "H." The two-level variable might be the better alternative, as random forest and boosted trees are able to achieve accuracy rates of 81% for "low" and 76% for "high."

Table 9: Accuracy Rates by Category for 2-level Variable (top) and 3-level Variable (bottom)

	Classification Tree	Bagged Tree	Random Forest	Boosted Tree
low	0.73	0.796	0.81	0.81
high	0.65	0.743	0.76	0.76
L	0	0	0	0
M	0.959	0.967	0.985	0.959
H	0.349	0.512	0.535	0.488

4.1.2 Variable Importance

Table 10 shows whether each of the predictors are important in the tree models. The numbers in parentheses after the model names indicate whether the model uses the two-level or the three-level variable. Several predictors have high importance in multiple models, such as alcohol, volatile acidity, and sulphates. As discussed before, if strong predictors exist, the trees in bagging tend to be similar, which can limit the amount of variance bagging can decrease. This explains why random forest models have the best performance, since it randomly picks a subset of predictors for each tree.

4.2 Regression Methods

4.2.1 Model Comparison and Recommendations

As shown in Table 11, the unpruned regression tree has the highest test MSE, but the tree could still be beneficial because only five predictors are needed: alcohol, sulfates, volatile acidity, fixed acidity, and total sulfur dioxide. Winemakers could save cost and time by only taking measurements of these five predictors. Multiple linear models have lower test MSE than the regression tree, and they are also easy to interpret, especially with a subset of predictors. The multiple linear model with six predictors could be a good alternative to the tree. Clearly, bagging or random forests are the best at predicting the quality of wine among

Table 10: Important Variables from Classification Tree, Bagged Tree, and Random Forest

Models	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	fixed sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
Classification (2)		X			X		X			X	X
Bagged (2)							X	X		X	X
Random Forest (2)		X								X	X
Classification (3)										X	X
Bagged (3)	X	X	X							X	X
Random Forest (3)		X								X	X

the regression methods used in this project. Any winemaker who cares more about the resulting prediction than interpretability should choose these ensemble learning methods.

Table 11: Comparison of Regression Models

	Multiple Linear Regression				Trees				
Type of Tree	NA				Unpruned	Random Forest			Bagging
number of predictors	11	6	7	8	11	3	4	7	11
Test MSE	0.39	0.39	0.389	0.389	0.443	0.289	0.29	0.295	0.294

4.3 Further Analysis

Much more could be done if we were to repeat this project. Certainly, we would want to address the limitations in our data. As mentioned before, we need more observations for quality 3 and quality 8 wines to better predict those categories. Especially for quality 3, we may want to ask the winemakers for more information about the winemaking process such as factors besides the chemical properties for such a low quality wine. If we are speaking with the authors of the data set, we certainly want to know about how quality was measured. Are these integer values equidistant in terms of quality, or is it on some other scale?

In terms of modeling, we may want to consider using dimension reduction methods such as Principal Component Analysis since some predictors are correlated. This might produce better performance because the new predictors would be uncorrelated with each other. We could also explore more ways to collapse the response variable quality. As shown in the comparison of overall accuracy rates and accuracy rates by category, the difference in prediction capability comes more from how the levels are collapsed than from the type of learning methods. It would be interesting to see if other ways of combining the values could allow the same models to achieve better performance.

5 Conclusion

This project identifies random forest as the best statistical learning method to answer our research question: can we use chemical properties to predict wine quality? We have found some evidence to give a positive answer to this question, but some caveats remain. For classification methods, high overall accuracy rates come with low accuracy for certain categories. For regression methods, the models never predict a wine quality rating of 3. These problems are inevitable when a category contains few observations. Further, the balance between interpretability and accuracy are hard to achieve; in the real world, clear communication with the winemakers will be essential to help them select the best models.

6 Appendix

```
knitr::opts_chunk$set(echo = FALSE, eval = TRUE, cache = TRUE, warning = FALSE, message = FALSE
knitr::opts_chunk$set(tidy.opts = list(width.cutoff = 80), tidy = TRUE)
# knitr::knit_hooks$set(plot = function(x, options) {
# paste0(knitr::hook_plot_tex(x, options), '\n\\FloatBarrier\n') })
library(tree)
library(tidyverse)
library(caret)
library(rattle)
library(randomForest)
wine <- read_delim("winequality-red.csv", delim = ";")
# shorten variable names
fa <- wine$`fixed acidity`
va <- as.numeric(wine$`volatile acidity`)
ca <- as.numeric(wine$`citric acid`)
rs <- wine$`residual sugar`
ch <- as.numeric(wine$chlorides)
fsd <- wine$`free sulfur dioxide`
tsd <- wine$`total sulfur dioxide`
den <- as.numeric(wine$density)
ph <- wine$pH
sul <- as.numeric(wine$sulphates)
al <- wine$alcohol
qual <- wine$quality
winez <- data.frame(fa, va, ca, rs, ch, fsd, tsd, den, ph, sul, al, qual)
# collapse qual into 2 labels
winez$qual2 <- as.factor(ifelse(winez$qual < 6, "low", "high"))
# collapse qual into 3 labels
winez$qual3 <- as.factor(ifelse(winez$qual < 5, "L", ifelse(winez$qual < 7, "M",
    "H")))
table(qual)
table(winez$qual2)
table(winez$qual3)
# separate data into a training set and a test set
set.seed(2)
train <- sample(nrow(winez), nrow(winez) * 0.8)
winez_train <- winez[train, ]
winez_test <- winez[-train, ]
# a table for data set overview and summary statistics
row1 <- c("fixed acidity", 4.6, 7.9, 8.32, 15.9)
row2 <- c("volatile acidity", 0.12, 0.52, 0.528, 1.58)
row3 <- c("citric acid", 0, 0.26, 0.271, 1)
row4 <- c("residual sugar", 0.9, 2.2, 2.539, 15.5)
row5 <- c("chlorides", 0.012, 0.079, 0.087, 0.611)
row6 <- c("fixed sulfur dioxide", 1, 14, 15.87, 72)
row7 <- c("total sulfur dioxide", 6, 38, 46.47, 289)
```

```
row8 <- c("density", 0.99, 0.997, 0.997, 1.004)
row9 <- c("pH", 2.74, 3.31, 3.311, 4.01)
row10 <- c("sulphates", 0.33, 0.62, 0.658, 2)
row11 <- c("alcohol", 8.4, 10.2, 10.42, 14.9)
row12 <- c("quality", 3, 6, 5.636, 8)
# table for summary statistics
table1 <- data.frame(rbind(row1, row2, row3, row4, row5, row6, row7, row8, row9,
    row10, row11, row12))
colnames(table1) <- c("variable name", "min", "median", "mean", "max")
knitr::kable(table1, row.names = FALSE, caption = "Summary Statistics")
# correlation
cor(winez[1:12])
# numbers for frequncy counts for categorical variables
freq3 <- sum(winez$qual == 3)
freq4 <- sum(winez$qual == 4)
freq5 <- sum(winez$qual == 5)
freq6 <- sum(winez$qual == 6)
freq7 <- sum(winez$qual == 7)
freq8 <- sum(winez$qual == 8)
# no-information rates
noInfo2 <- max(prop.table(table(winez_train$qual2)))
noInfo3 <- max(prop.table(table(winez_train$qual3)))
# classification tree with 2-level response variable
decisionTree2 <- tree(qual2 ~ fa + va + ca + rs + ch + fsd + tsd + den + ph + sul +
    al, data = winez_train)
# summary(decisionTree2)
decisionTree2_pred <- predict(decisionTree2, newdata = winez_test, type = "class")
decisionTree2_test <- mean(decisionTree2_pred == winez_test$qual2)
# classification tree with 3-level response variable
decisionTree3 <- tree(qual3 ~ fa + va + ca + rs + ch + fsd + tsd + den + ph + sul +
    al, data = winez_train)
# summary(decisionTree3)
decisionTree3_pred <- predict(decisionTree3, newdata = winez_test, type = "class")
decisionTree3_test <- mean(decisionTree3_pred == winez_test$qual3)
# values for creating confusion matrices for classification trees
tree2_1 <- sum(decisionTree2_pred == "low" & winez_test$qual2 == "low")
tree2_2 <- sum(decisionTree2_pred == "low" & winez_test$qual2 == "high")
tree2_3 <- sum(decisionTree2_pred == "high" & winez_test$qual2 == "low")
tree2_4 <- sum(decisionTree2_pred == "high" & winez_test$qual2 == "high")
tree2_low <- round(tree2_1/sum(winez_test$qual2 == "low"), 3)
tree2_high <- round(tree2_4/sum(winez_test$qual2 == "high"), 3)
tree3_1 <- sum(decisionTree3_pred == "L" & winez_test$qual3 == "L")
tree3_2 <- sum(decisionTree3_pred == "L" & winez_test$qual3 == "M")
tree3_3 <- sum(decisionTree3_pred == "L" & winez_test$qual3 == "H")
tree3_4 <- sum(decisionTree3_pred == "M" & winez_test$qual3 == "L")
tree3_5 <- sum(decisionTree3_pred == "M" & winez_test$qual3 == "M")
tree3_6 <- sum(decisionTree3_pred == "M" & winez_test$qual3 == "H")
```

```
tree3_7 <- sum(decisionTree3_pred == "H" & winez_test$qual3 == "L")
tree3_8 <- sum(decisionTree3_pred == "H" & winez_test$qual3 == "M")
tree3_9 <- sum(decisionTree3_pred == "H" & winez_test$qual3 == "H")
tree3_L <- round(tree3_1/sum(winez_test$qual3 == "L"), 3)
tree3_M <- round(tree3_5/sum(winez_test$qual3 == "M"), 3)
tree3_H <- round(tree3_9/sum(winez_test$qual3 == "H"), 3)
pruneTree2 <- cv.tree(decisionTree2, FUN = prune.misclass)
pruneTree3 <- cv.tree(decisionTree3, FUN = prune.misclass)
# prune tree
prunedTree3 <- prune.misclass(decisionTree3, best = 4)
# make predictions
prunedTree3_pred <- predict(prunedTree3, newdata = winez_test, type = "class")
# calculate accuracy rate
prunedTree3_test <- mean(prunedTree3_pred == winez_test$qual3)
# values for creating confusion matrices for pruned tree
Ptree3_1 <- sum(prunedTree3_pred == "L" & winez_test$qual3 == "L")
Ptree3_2 <- sum(prunedTree3_pred == "L" & winez_test$qual3 == "M")
Ptree3_3 <- sum(prunedTree3_pred == "L" & winez_test$qual3 == "H")
Ptree3_4 <- sum(prunedTree3_pred == "M" & winez_test$qual3 == "L")
Ptree3_5 <- sum(prunedTree3_pred == "M" & winez_test$qual3 == "M")
Ptree3_6 <- sum(prunedTree3_pred == "M" & winez_test$qual3 == "H")
Ptree3_7 <- sum(prunedTree3_pred == "H" & winez_test$qual3 == "L")
Ptree3_8 <- sum(prunedTree3_pred == "H" & winez_test$qual3 == "M")
Ptree3_9 <- sum(prunedTree3_pred == "H" & winez_test$qual3 == "H")
Ptree3_L <- round(Ptree3_1/sum(winez_test$qual3 == "L"), 3)
Ptree3_M <- round(Ptree3_5/sum(winez_test$qual3 == "M"), 3)
Ptree3_H <- round(Ptree3_9/sum(winez_test$qual3 == "H"), 3)
# tree plots
knitr::include_graphics("classT1.png")
# par(mfrow = c(1, 2)) plot(decisionTree2) text(decisionTree2, pretty = 50, cex =
# .5) plot(prunedTree3) text(prunedTree3, pretty = 50, cex = .6) bagging with
# 2-level variable
bag_tree2 <- train(qual2 ~ fa + va + ca + rs + ch + fsd + tsd + den + ph + sul +
    al, data = winez_train, method = "treebag", preProcess = c("center", "scale"),
    trControl = trainControl(method = "cv", number = 10))
bag_tree2_pred <- predict(bag_tree2, newdata = winez_test)
bagMatrix2 <- table(bag_tree2_pred, winez_test$qual2)
bag2_test <- mean(bag_tree2_pred == winez_test$qual2)
# varImp(bag_tree2)

# bagging with 3-level variable
bag_tree3 <- train(qual3 ~ fa + va + ca + rs + ch + fsd + tsd + den + ph + sul +
    al, data = winez_train, method = "treebag", preProcess = c("center", "scale"),
    trControl = trainControl(method = "cv", number = 10))
bag_tree3_pred <- predict(bag_tree3, newdata = winez_test)
bagMatrix3 <- table(bag_tree3_pred, winez_test$qual3)
bag3_test <- mean(bag_tree3_pred == winez_test$qual3)
```

```
# varImp(bag_tree3) values for creating confusion matrices for bagged trees
Btree2_1 <- sum(bag_tree2_pred == "low" & winez_test$qual2 == "low")
Btree2_2 <- sum(bag_tree2_pred == "low" & winez_test$qual2 == "high")
Btree2_3 <- sum(bag_tree2_pred == "high" & winez_test$qual2 == "low")
Btree2_4 <- sum(bag_tree2_pred == "high" & winez_test$qual2 == "high")
Btree2_low <- round(Btree2_1/sum(winez_test$qual2 == "low"), 3)
Btree2_high <- round(Btree2_4/sum(winez_test$qual2 == "high"), 3)
Btree3_1 <- sum(bag_tree3_pred == "L" & winez_test$qual3 == "L")
Btree3_2 <- sum(bag_tree3_pred == "L" & winez_test$qual3 == "M")
Btree3_3 <- sum(bag_tree3_pred == "L" & winez_test$qual3 == "H")
Btree3_4 <- sum(bag_tree3_pred == "M" & winez_test$qual3 == "L")
Btree3_5 <- sum(bag_tree3_pred == "M" & winez_test$qual3 == "M")
Btree3_6 <- sum(bag_tree3_pred == "M" & winez_test$qual3 == "H")
Btree3_7 <- sum(bag_tree3_pred == "H" & winez_test$qual3 == "L")
Btree3_8 <- sum(bag_tree3_pred == "H" & winez_test$qual3 == "M")
Btree3_9 <- sum(bag_tree3_pred == "H" & winez_test$qual3 == "H")
Btree3_L <- round(Btree3_1/sum(winez_test$qual3 == "L"), 3)
Btree3_M <- round(Btree3_5/sum(winez_test$qual3 == "M"), 3)
Btree3_H <- round(Btree3_9/sum(winez_test$qual3 == "H"), 3)
# random forest
rfGrid <- expand.grid(mtry = 2:8)
# 2-level variable
rf_tree2 <- train(qual2 ~ fa + va + ca + rs + ch + fsd + tsd + den + ph + sul + al,
    data = winez_train, method = "rf", preProcess = c("center", "scale"), trControl = trainCont
        number = 10), tuneGrid = rfGrid)
rf2_pred <- predict(rf_tree2, newdata = winez_test)
rfMatrix2 <- table(rf2_pred, winez_test$qual2)
rf2_test <- mean(rf2_pred == winez_test$qual2)
# varImp(rf_tree2) 3-level variable
rf_tree3 <- train(qual3 ~ fa + va + ca + rs + ch + fsd + tsd + den + ph + sul + al,
    data = winez_train, method = "rf", preProcess = c("center", "scale"), trControl = trainCont
        number = 10), tuneGrid = rfGrid)
rf3_pred <- predict(rf_tree3, newdata = winez_test)
rfMatrix3 <- table(rf3_pred, winez_test$qual3)
rf3_test <- mean(rf3_pred == winez_test$qual3)
# varImp(rf_tree3) random forest plot: accuracy rates vs number of predictors
rfplot1 <- plot(rf_tree2)
rfplot2 <- plot(rf_tree3)
gridExtra::grid.arrange(rfplot1, rfplot2, nrow = 1, ncol = 2)
# values for creating confusion matrices for random forest
Rtree2_1 <- sum(rf2_pred == "low" & winez_test$qual2 == "low")
Rtree2_2 <- sum(rf2_pred == "low" & winez_test$qual2 == "high")
Rtree2_3 <- sum(rf2_pred == "high" & winez_test$qual2 == "low")
Rtree2_4 <- sum(rf2_pred == "high" & winez_test$qual2 == "high")
Rtree2_low <- round(Rtree2_1/sum(winez_test$qual2 == "low"), 3)
Rtree2_high <- round(Rtree2_4/sum(winez_test$qual2 == "high"), 3)
Rtree3_1 <- sum(rf3_pred == "L" & winez_test$qual3 == "L")
```

```
Rtree3_2 <- sum(rf3_pred == "L" & winez_test$qual3 == "M")
Rtree3_3 <- sum(rf3_pred == "L" & winez_test$qual3 == "H")
Rtree3_4 <- sum(rf3_pred == "M" & winez_test$qual3 == "L")
Rtree3_5 <- sum(rf3_pred == "M" & winez_test$qual3 == "M")
Rtree3_6 <- sum(rf3_pred == "M" & winez_test$qual3 == "H")
Rtree3_7 <- sum(rf3_pred == "H" & winez_test$qual3 == "L")
Rtree3_8 <- sum(rf3_pred == "H" & winez_test$qual3 == "M")
Rtree3_9 <- sum(rf3_pred == "H" & winez_test$qual3 == "H")
Rtree3_L <- round(Rtree3_1/sum(winez_test$qual3 == "L"), 3)
Rtree3_M <- round(Rtree3_5/sum(winez_test$qual3 == "M"), 3)
Rtree3_H <- round(Rtree3_9/sum(winez_test$qual3 == "H"), 3)
# boosted tree
gbmGrid <- expand.grid(interaction.depth = c(1, 2, 4), n.trees = (1:30) * 50, shrinkage = 0.1,
    n.minobsinnode = 10)
# 2-level variable
boost_tree2 <- train(qual2 ~ fa + va + ca + rs + ch + fsd + tsd + den + ph + sul +
    al, data = winez_train, method = "gbm", preProcess = c("center", "scale"), trControl = tra
    number = 10), tuneGrid = gbmGrid, verbose = FALSE)
boost2_pred <- predict(rf_tree2, newdata = winez_test)
boostMatrix2 <- table(boost2_pred, winez_test$qual2)
boost2_test <- mean(boost2_pred == winez_test$qual2)
# 3-level variable
boost_tree3 <- train(qual3 ~ fa + va + ca + rs + ch + fsd + tsd + den + ph + sul +
    al, data = winez_train, method = "gbm", preProcess = c("center", "scale"), trControl = tra
    number = 10), tuneGrid = gbmGrid, verbose = FALSE)
boost3_pred <- predict(boost_tree3, newdata = winez_test)
boostMatrix3 <- table(boost3_pred, winez_test$qual3)
boost3_test <- mean(boost3_pred == winez_test$qual3)
# boosted tree plot: accuracy rates vs number of iterations
boostPlot1 <- ggplot(boost_tree2) + theme(legend.position = "bottom")
boostPlot2 <- ggplot(boost_tree3) + theme(legend.position = "bottom")
gridExtra::grid.arrange(boostPlot1, boostPlot2, nrow = 1, ncol = 2)
# values for creating confusion matrices for boosted trees
BStree2_1 <- sum(boost2_pred == "low" & winez_test$qual2 == "low")
BStree2_2 <- sum(boost2_pred == "low" & winez_test$qual2 == "high")
BStree2_3 <- sum(boost2_pred == "high" & winez_test$qual2 == "low")
BStree2_4 <- sum(boost2_pred == "high" & winez_test$qual2 == "high")
BStree2_low <- round(BStree2_1/sum(winez_test$qual2 == "low"), 3)
BStree2_high <- round(BStree2_4/sum(winez_test$qual2 == "high"), 3)
BStree3_1 <- sum(boost3_pred == "L" & winez_test$qual3 == "L")
BStree3_2 <- sum(boost3_pred == "L" & winez_test$qual3 == "M")
BStree3_3 <- sum(boost3_pred == "L" & winez_test$qual3 == "H")
BStree3_4 <- sum(boost3_pred == "M" & winez_test$qual3 == "L")
BStree3_5 <- sum(boost3_pred == "M" & winez_test$qual3 == "M")
BStree3_6 <- sum(boost3_pred == "M" & winez_test$qual3 == "H")
BStree3_7 <- sum(boost3_pred == "H" & winez_test$qual3 == "L")
BStree3_8 <- sum(boost3_pred == "H" & winez_test$qual3 == "M")
```

```
BStree3_9 <- sum(boost3_pred == "H" & winez_test$qual3 == "H")
BStree3_L <- round(BStree3_1/sum(winez_test$qual3 == "L"), 3)
BStree3_M <- round(BStree3_5/sum(winez_test$qual3 == "M"), 3)
BStree3_H <- round(BStree3_9/sum(winez_test$qual3 == "H"), 3)
# multiple linear regression
WineLR <- lm(qual ~ fa + va + ca + rs + ch + fsd + tsd + den + ph + sul + al, data = winez_tra
WineLR.testMSE <- mean((winez_test$qual - predict(WineLR, winez_test))^2)
Wine6 <- lm(qual ~ al + va + sul + tsd + ch + ph, data = winez_train)
Wine6.testMSE <- mean((winez_test$qual - predict(Wine6, winez_test))^2)
Wine7 <- lm(qual ~ al + va + sul + tsd + ch + ph + fsd, data = winez_train)
Wine7.testMSE <- mean((winez_test$qual - predict(Wine7, winez_test))^2)
Wine8 <- lm(qual ~ al + va + sul + tsd + ch + ph + fsd + ca, data = winez_train)
Wine8.testMSE <- mean((winez_test$qual - predict(Wine8, winez_test))^2)
# MLR plots
par(mfrow = c(2, 2))
plot(WineLR)
# regression tree
tree.Wine <- tree(qual ~ fa + va + ca + rs + ch + fsd + tsd + den + ph + sul + al,
    data = winez_train)
yhat.Wine <- predict(tree.Wine, newdata = winez_test)
up.tree.testMSE <- mean((yhat.Wine - winez_test$qual)^2)
# regression tree plot
knitr::include_graphics("regressT.png")
# plot(tree.Wine) text(tree.Wine, pretty = 50) Huge improvement on test MSE with
# bagging
bag.Wine <- randomForest(qual ~ fa + va + ca + rs + ch + fsd + tsd + den + ph + sul +
    al, data = winez_train, mtry = 11, importance = TRUE)
yhat.Wine <- predict(bag.Wine, newdata = winez_test)
bag.testMSE <- mean((yhat.Wine - winez_test$qual)^2)
# variable importance plot
knitr::include_graphics("varImpP.png")
# varImpPlot(bag.Wine, main = NULL) randomforests, mtry = sqrt(11)
rf.def.Wine <- randomForest(qual ~ fa + va + ca + rs + ch + fsd + tsd + den + ph +
    sul + al, data = winez_train, importance = TRUE)
yhat.rf.Wine <- predict(rf.def.Wine, newdata = winez_test)
rf.mtry3.testMSE <- mean((yhat.rf.Wine - winez_test$qual)^2)

# mtry = 4
rf.fit <- randomForest(qual ~ fa + va + ca + rs + ch + fsd + tsd + den + ph + sul +
    al, data = winez_train, mtry = 4, importance = TRUE)
rf.pred <- predict(rf.fit, newdata = winez_test)
rf.mtry4.testMSE <- mean((rf.pred - winez_test$qual)^2)

# mtry = 7
rf.def.Wine <- randomForest(qual ~ fa + va + ca + rs + ch + fsd + tsd + den + ph +
    sul + al, data = winez_train, mtry = 7, importance = TRUE)
yhat.rf.Wine <- predict(rf.def.Wine, newdata = winez_test)
```

```
rf.mtry7.testMSE <- mean((yhat.rf.Wine - winez_test$qual)^2)
# comparison table for all classification models
model_comp <- rbind(c(round(noInfo2, 3), round(decisionTree2_test, 3), round(bag2_test,
    3), round(rf2_test, 3), round(boost2_test, 3)), c(round(noInfo3, 3), round(prunedTree3_test
    3), round(bag3_test, 3), round(rf3_test, 3), round(boost3_test, 3)))
colnames(model_comp) <- c("no-information rate", "classification tree", "bagged tree",
    "random forest", "boosted tree")
rownames(model_comp) <- c("2-level", "3-level")
comp <- knitr::kable(model_comp, caption = "Training Accuracy Rates for All Classification Mode
kableExtra::kable_styling(comp, latex_options = "hold_position")
# we tried a LASSO model, but it doesn't really make sense to include this in our
# analysis because there was very little improvement with variable selection
Wine.x = model.matrix(quality ~ ., WineRed)
Wine.y = WineRed$quality

grid = 10^seq(10, -2, length = 100)

lasso.Wine = glmnet(Wine.x[train, ], Wine.y[train], alpha = 1, lambda = grid)
plot(lasso.Wine)

set.seed(2)
cv.Wine.lasso = cv.glmnet(Wine.x[train, ], Wine.y[train], alpha = 1)
plot(cv.Wine.lasso)
bestlam = cv.Wine.lasso$lambda.min
lasso.Wine.pred = predict(lasso.Wine, s = bestlam, newx = Wine.x[-train, ])
lasso.testMSE = mean((lasso.Wine.pred - Wine.y[-train])^2)
```

```
# we tried a LASSO model, but it doesn't really make sense to include this in our
# analysis because there was very little improvement with variable selection
Wine.x = model.matrix(quality ~ ., WineRed)
Wine.y = WineRed$quality

grid = 10^seq(10, -2, length = 100)

lasso.Wine = glmnet(Wine.x[train, ], Wine.y[train], alpha = 1, lambda = grid)
plot(lasso.Wine)

set.seed(2)
cv.Wine.lasso = cv.glmnet(Wine.x[train, ], Wine.y[train], alpha = 1)
plot(cv.Wine.lasso)
bestlam = cv.Wine.lasso$lambda.min
lasso.Wine.pred = predict(lasso.Wine, s = bestlam, newx = Wine.x[-train, ])
lasso.testMSE = mean((lasso.Wine.pred - Wine.y[-train])^2)
```