Michael Knight
6/22/2022
STAT 627-001

Final Project: Predicting Quality of Portuguese Wine

For my final project I decided to use the Wine Quality Data Set (https://archive.ics.uci.edu/ml/datasets/Wine+Quality) for both my regression and classification model sets. The study sample data consists of 1,599 red wines and 4,898 white wines from the northwest region of Portugal (one of the top ten wine producing countries in the world), a type of wine known as Vinho Verde. This data was collected between May of 2004 to February of 2007. There were zero null values in the data set and thus no observations needed to be dropped. For the sake of this project I focused solely on the white wine data (using a full data set sample size of n = 4,898 observations). I excluded the red wine data because running and reporting on four models on two data sets would have exceeded the page limitation for this assignment, because combining the data sets would have added a categorical predictor variable to my otherwise fully quantitative set of predictor variables, for the fact that red wine is different enough from white wine that they should be done separately, and for the principle of parsimony. The overall goal of my analysis was to try to predict the quality of white wine based on a set of objective quantitative variables using 2 different regression methods (Multiple Linear Regression and Shrinkage Methods, predicting quality as a quantitative variable) and 2 different classification methods (Support Vector Machine and Bayes Classifiers, predicting quality as a qualitative variable), and to determine which models performed better.

There are 11 predictor variables measured which are all qualitative (and all floats/doubles). They are ‘fixed.acidity’ (the fixed acidity of the wine measured in grams of tartaric acid per cubic decimeter or liter), ‘volatile.acidity’ (volatile acidity of the wine measured in grams of acetic acid per cubic decimeter or liter), ‘citric.acid’ (the amount of citric acid in a wine measured in grams per cubic decimeter or liter), ‘residual.sugar’ (the amount of residual sugar in a wine measured in grams per cubic decimeter or liter), ‘chlorides’ (the amount of chlorides in a wine measured in grams per cubic decimeter or liter), ‘free.sulfur.dioxide’ (the amount of free sulfur dioxide in a wine measured in grams per cubic decimeter or liter), ‘total.sulfur.dioxide’ (the amount of total sulfur dioxide in a wine measured in grams per cubic decimeter or liter), ‘density’ (the density of a wine measured as grams of wine per cubic centimeter), ‘pH’ (the pH balance of each wine), ‘sulphates’ (the amount of sulfates in the wine measured in grams of potassium sulfate per cubic decimeter or liter), and ‘alcohol’ (alcohol content as a percentage).

The response variable measured is wine quality, which was based on sensory data provided by professional sommeliers (more specifically by the Comissão de Viticultura da Região dos Vinhos Verdes, or CVRVV; an inter-professional organization with the objective of representing the interests of the professions involved in the Vinho Verde production and trade). Although it is in the data set as a quantitative measure (rating from 0-10, with 0 denoting ‘very bad’ and 10 denoting ‘excellent’), I derived a binary categorical variable, ‘quality.good’, which

has a value of ‘good’ (or 1) for wines with a quality rating from 6-10 and a value of ‘bad’ (or 0) for wines with a quality rating from 0-5 (I had initially explored just using the original ‘quality’ variable with an ‘as.factor()’ run on it to get a categorical variable with more specific ratings, but abandoned this idea largely for parsimony and the page length limitation; this ‘quality.factor’ variable did come in handy for my exploratory data analysis however). For both of my classification models I used this qualitative ‘quality.good’ feature as the response variable, and for my two regression models I used the built-in quantitative ‘quality’ feature as my response variable. On the 4,898 white wines in my full data set I performed a 90/10 train-test split, giving me a training set of 4,408 random white wines (reduced to 4,402 after removal of extreme outliers) and a testing set of 490 random white wines, upon which I respectively used to train and test both my two regression models and my two classification models (making sure to remove the quantitative ‘quality’ variable from my classification models and the qualitative ‘quality.good’ variable from my regression models.

Looking at the later abandoned ‘quality.factor’ variable, I noticed some interesting things. It is interesting to me that for white wines, there are no wines with a rating less than 3 (and there were only 19 white with a rating of 3) and no whites with a rating higher than 9 (and there were only 5 wines with a rating of 9). This was another reason why I ultimately went with the binary ‘quality.good’ as the response variable for my classification models. Comparing the scatterplots of the predictors to the qualitative ‘quality.good’ response variable with the scatterplots of the predictors to the quantitative ‘quality’ response variable showed very similar patterns, even though ‘quality.good’ was binary and thus not as specific as the unused qualitative ‘quality.factor’ would-be response variable, which also led to the decision to go with ‘quality.good’. Looking at scatterplots of the predictors plotted against the response (for both the qualitative and quantitative versions of quality), I found 6 observations to be significantly skewing the data for fixed acidity, citric acid, residual sugar, free sulfur dioxide, and density, so I removed them. Observing these plots I also noticed some curvilinearity in all but two of the plots, so I tried to see if transforming them would affect the fit/residuals when building my MLR model.

```
fixed.acidity    volatile.acidity  citric.acid     residual.sugar     chlorides       free.sulfur.dioxide
Min.   : 3.800   Min.   :0.0800   Min.   :0.0000   Min.   : 0.600   Min.   :0.00900   Min.   :  2.00
1st Qu.: 6.300   1st Qu.:0.2100   1st Qu.:0.2700   1st Qu.: 1.700   1st Qu.:0.03600   1st Qu.: 23.38
Median : 6.800   Median :0.2600   Median :0.3200   Median : 5.200   Median :0.04300   Median : 34.00
Mean   : 6.855   Mean   :0.2782   Mean   :0.3345   Mean   : 6.403   Mean   :0.04593   Mean   : 35.38
3rd Qu.: 7.300   3rd Qu.:0.3200   3rd Qu.:0.3900   3rd Qu.: 9.900   3rd Qu.:0.05000   3rd Qu.: 46.00
Max.   :14.200   Max.   :1.1000   Max.   :1.6600   Max.   :65.800   Max.   :0.34600   Max.   :289.00

total.sulfur.dioxide    density             pH          sulphates         alcohol         quality      quality.factor
Min.   :  9.0        Min.   :0.9871   Min.   :2.720   Min.   :0.2200   Min.   : 8.00   Min.   :3.000   3:  19
1st Qu.:108.0        1st Qu.:0.9917   1st Qu.:3.090   1st Qu.:0.4100   1st Qu.: 9.50   1st Qu.:5.000   4: 143
Median :134.0        Median :0.9938   Median :3.180   Median :0.4700   Median :10.40   Median :6.000   5:1322
Mean   :138.5        Mean   :0.9940   Mean   :3.188   Mean   :0.4895   Mean   :10.51   Mean   :5.878   6:1967
3rd Qu.:168.0        3rd Qu.:0.9961   3rd Qu.:3.280   3rd Qu.:0.5500   3rd Qu.:11.40   3rd Qu.:6.000   7: 791
Max.   :440.0        Max.   :1.0390   Max.   :3.810   Max.   :1.0800   Max.   :14.05   Max.   :9.000   8: 161
                                                                                                     9:   5

quality.good
0:1484
1:2924
```

Regression

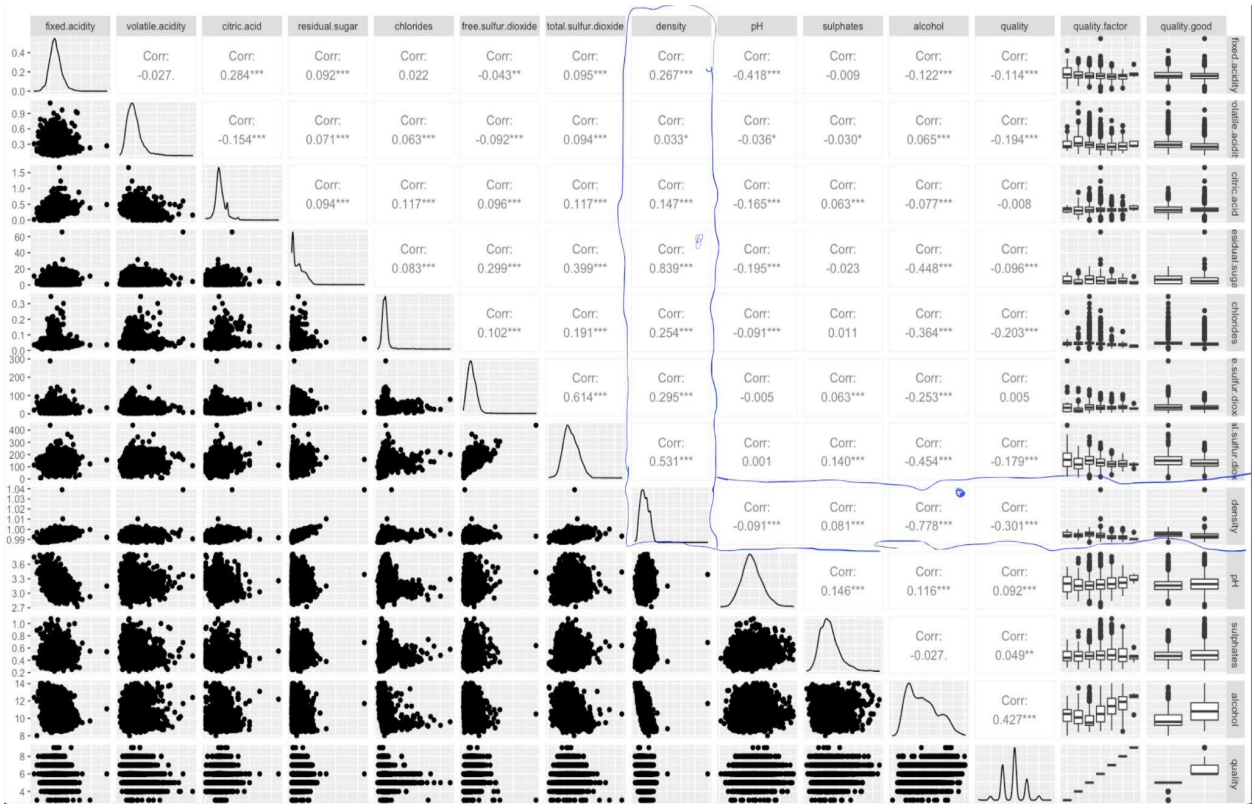
MLR (Using Best Subsets) (Regression 1)

The first regression method I chose to use on my data was Multiple Linear Regression (MLR). MLR starts with setting up a model based on a data set of n observations (sample size of the data) and p variables (here there are 12) with a response variable (Y = wine quality, as a vector of n y-values) and p-1 predictor variables (X = x_1, x_2, …, x_{11} = 'fixed.acidity', 'volatile.acidity', … 'alcohol', as a matrix of n*p-1 x values). The model is then set up as $Y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \ldots + \beta_{p-1} x_{i,p-1} + \varepsilon_i$, where i = 1, 2, …, n and ε represents an error term, or in matrix terms as $\mathbf{Y} = \mathbf{X}\beta + \varepsilon$, where

$$\mathbf{Y} = \begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ \vdots \\ Y_n \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} 1 & X_{1,1} & X_{1,2} & \dots & X_{1,p-1} \\ 1 & X_{2,1} & X_{2,2} & \dots & X_{2,p-1} \\ 1 & X_{3,1} & X_{3,2} & \dots & X_{3,p-1} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & X_{n,1} & X_{n,2} & \dots & X_{n,p-1} \end{bmatrix} \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{p-1} \end{bmatrix} \quad \textbf{and} \quad \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

The tuning parameters for this method are the variables that you choose to include or exclude in the model, and how you may want to transform (or mathematically alter) the variables before including them in the fit of the model.

There are 6 assumptions that must be met in order to run a MLR model; linearity (i.e. it can be represented in the form of the formula provided above, independence of the error terms, normal distribution of the error terms, equal variance of error terms (or homoscedasticity), independence of the predictor variables i.e. no significant correlation amongst the predictor variables (known as multicollinearity), and the nonstochastic (x values not chosen randomly) and many more observations then there are features (n >> p). Based on the data I chose I already know that the x values are not random, and that our number of observations far exceeds the number of variables. In terms of multicollinearity, correlation coefficients whose magnitudes are between 0.9 and 1.0 indicate variables which can be considered very highly correlated. Correlation coefficients whose magnitudes are between 0.7 and 0.9 indicate variables which can be considered highly correlated. Looking at the correlation between all of the variables (untransformed) using ggpairs, I observed only one instance of multicollinearity; there was an .833% correlation between density and residual.sugars. Density also has the second highest correlation (-.803) with alcohol, and the fourth highest correlation (0.550) with total sulfur dioxide. I addressed the issue by deciding to remove the density variable from my MLR model. Normality of error terms can be checked by observing histogram plots of the residuals given by the model and checking to make sure that there is a bell curve; this was the case for all of the predictor variables.

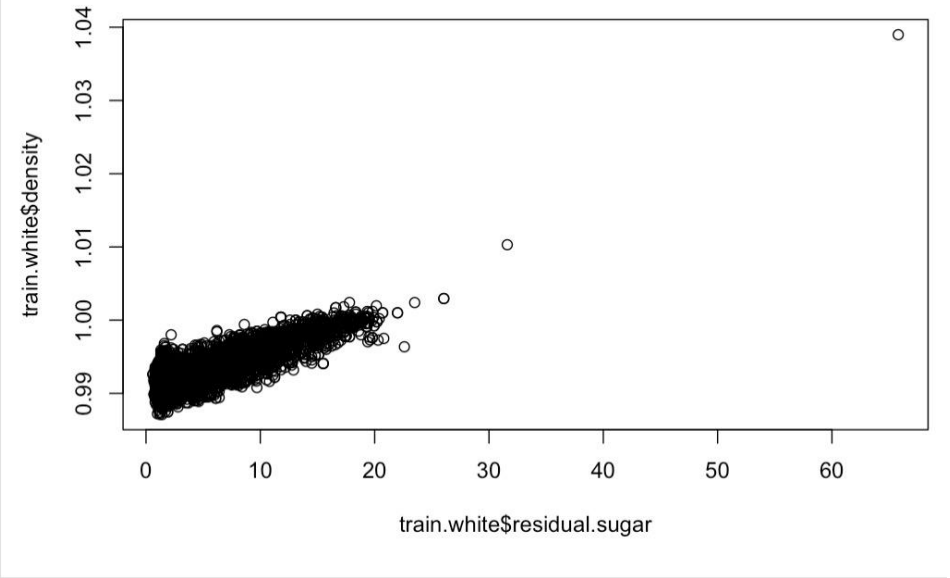


146

147 ```{r}

148 plot(train.white$residual.sugar, train.white$density)

149 ```



The remaining assumptions of the MLR model (linearity, independence, and equal variance of the error terms) can all be checked by fitting a model and observing a scatterplot of the residuals of the model graphed against the fitted values for each predictor variable of the model (see R Markdown). Linearity can be assured if there is no discernable pattern in the plot, independence can be determined if the residuals are centered around 0, and equal variance can be determined if the plot displays no evidence of 'funneling' (narrowing as the fitted values move left or right). Observing the graphs of the untransformed variables, there were no issues of equal variance or independence, but a many of the graphs (notably for 'volatile.acidity', 'residual.sugar', 'chlorides', 'free.sulfur.dioxide', 'pH', 'sulphates', and 'alcohol') did display some possible patterns. I addressed this by playing around with transforming each variable and found the best results when I made the transformations log(volatile.acidity), poly(residual.sugar, 3), poly(chlorides,6), poly(free.sulfur.dioxide, 3), log(pH), log(sulphates), and poly(alcohol,3). After dropping density and making these transformations, I am left with 21 predictor variables for my MLR model.

To further tune my MLR model and narrow the now much larger amount of variables to fit my final model on, I employed the use of variable selection using the method of best subsets by way of regsubsets() to see which of my newly transformed variables I should include in my model. Best subsets works by, based on a chosen criterion, the method determines how many and which predictor variables will give a model the best result based on said chosen criterion. I tried this on three different criteria; adjusted Cp, adjusted R^2 score, and BIC, which gave me 18 variables, 12 variables, and 20 variables (respectively). Fitting all three of these models and then running them on the testing data to get test MSEs as my metric for best MLR model, (Cp test MSE = 0.4487168, adjusted R^2 test MSE = 0.4539624, BIC test MSE = 0.4565343), I observed that the 18 variable model determined by the best Cp was the best choice. which were fixed.acidity, log(volatile.acidity), residual.sugar, residual.sugar^2, chlorides, chlorides^2, chlorides^3, chlorides^5, chlorides^6, free.sulfur.dioxide, free.sulfur.dioxide^2, free.sulfur.dioxide^3, total.sulfur.dioxide, log(pH), log(sulphates), alcohol, alcohol^2, and alcohol^3. So, given the coefficients provided by the summary of this model, the optimal MLR model can be represented as:

Quality = 5.1266931 - 0.0401307*fixed.acidity -0.5931897*log(volatile.acidity) + 6.2975681*residual.sugar -3.3238367*residual.sugar^2 - 1.9318351*chlorides + 2.6632498*chlorides^2 - 2.4968086*chlorides^3 + 1.3744869*chlorides^5 - 1.1204375*chlorides^6 + 6.4939960*free.sulfur.dioxide - 9.1820104*free.sulfur.dioxide^2 + 4.4594110*free.sulfur.dioxide^3 - 0.0012945*total.sulfur.dioxide + 0.5344207*log(pH) + 0.2795037*log(sulphates) + 26.1029883*alcohol + 4.1650825*alcohol^2 - 3.0991054*alcohol^3.

Based on these coefficients, it is clear to see that alcohol by far has the strongest correlation with wine quality (only to the first power), followed by free sulfur dioxide (taken to the power of 1, 2 and 3).

MLR optimal model: best subset by Cp criterion; train MSE = 0.542259, test MSE = 0.4487168

Shrinkage Methods (Ridge and LASSO) (Regression 2)

The second regression method I used to address the overall goal of the analysis was Shrinkage. While subsetting means choosing a subset from available variables to include in the model to reduce its dimensionality, Shrinkage, on the other hand, means reducing the size of the coefficient estimates (shrinking them towards zero) with respect to the orthonormal basis formed by the principal components. Coordinates with respect to principal components with smaller variance are shrunk more. Shrinking the coefficient estimates significantly reduces their variance, which prevents overfitting (but also may cause underfitting by introducing bias). Shrinkage also reduces multicollinearity by replacing $(X^TX)^{-1}$ with $(X^TX + \lambda I)^{-1}$, where λ(lambda) is the tuning (shrinkage) parameter and I is the identity matrix (recall that $\beta_{hat} = (X^TX)^{-1}X^TY$). There are two types of Shrinkage Methods; Ridge Regression and LASSO (least absolute shrinkage and selection operator) regression, which we can execute by running glmnet() with alpha = 0 for ridge and with alpha = 1 for LASSO (so in a sense, alpha is another tuning parameter for Shrinkage). The difference between Ridge Regression and LASSO regression is that, while both methods put a similar constraint on the coefficients by introducing a penalty factor, while lasso regression takes the magnitude of the coefficients, ridge regression takes the square. This is a little easier to understand when you compare the tail end of the mathematical equations for the estimates of the coefficients from both methods:

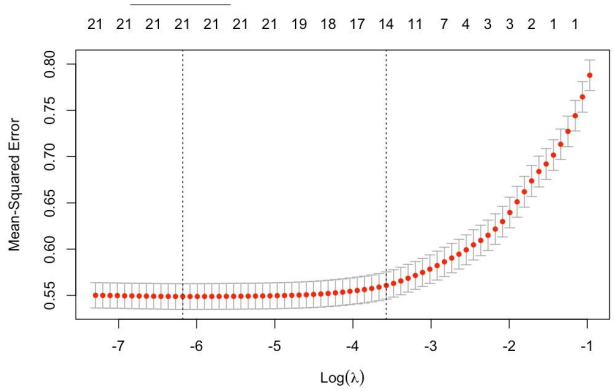
$$L_{lasso}(\hat{\beta}) = \sum_{i-1}^{n}(y_i - x_i'\hat{\beta})^2 + \lambda \sum_{j-1}^{m} |\hat{\beta}_j|.$$

$$L_{hridge}(\hat{\beta}) = \sum_{i-1}^{n}(y_i - x_i'\hat{\beta})^2 + \lambda \sum_{j-1}^{m} w_j \hat{\beta}_j^2.$$

The assumptions of Shrinkage methods are the same as that of linear regression: linearity, constant variance, and independence. However, as ridge regression does not provide confidence limits, the distribution of errors to be normal need not be assumed. This is easily achieved at this point by using the transformed variables I used for my MLR model before performing best subsets. We assume also that the predictor variables (X's) and response variable (Y) have been centered so that we have no need for a constant term in the regression. This can be achieved by converting the predictor variables (transformed as they were for the MLR method) of our training data into an n by p matrix with centered columns, which we can easily do in R using model.matrix() and then throwing out the first column (all 1's in the design matrix) and saving the response variable as a centered n-vector (which I did simply by running y.shrinkage <- train.white$quality).

I tuned my Shinkage method on alpha and lambda. Since there are only two values of alpha (0 for ridge and 1 for LASSO), I tuned the alpha parameter simply by running the code tuning for lambda once for each alpha. To tune for the lambda parameter, I created a grid of lambda values from 10^{10} down to 10^{-2} and performed 10-fold cross-validation to choose the lamda from this grid. This gave me an optimal lambda of 0.0379651 for the Ridge regression and an optimal lambda of 0.002073759 for the LASSO regression. When fitting the LASSO model on it's optimal lambda, the citric.acid and chlorides[4] were the variables dropped, and I got

a test MSE of 1.097064, and fitting the Ridge model on it's optimal lambda dropped the none of the variables and gave me a test MSE of 1.116491, so between the optimal Ridge and the optimal LASSO models, the optimal LASSO model performed better. The fact that the Ridge model dropped zero variables and got a lower score than LASSO was not surprising; LASSO regression will always zero out at least one beta coefficient, whereas Ridge regression can shrink beta coefficients close to zero, but never equal to zero. **Shrinkage optimal model: LASSO model with lambda tuned by 10-fold Cross Validation; MSE = 1.097064**



(*Visual representation of the lambda tuning for the LASSO model*)

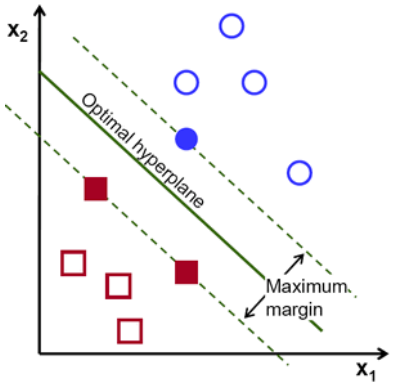
Regression conclusion: For the optimal MLR model using Best Subsets and optimizing by Cp, the train MSE was 0.542259 and the test MSE was 0.4487168; For the optimal Shrinkage method using a LASSO model with a lambda tuned by 10-fold Cross Validation, the train MSE was 0.5442927 and the test MSE was 1.097064, making the Multiple Linear Regression superior to the Shrinkage in the predicting the quality of white wine based of the features included in the White Wine dataset. Since both models are essentially running on the same MLR model, it may be more accurate to say that variable selection via Shrinkage outperformed variable selection via Best Subsets. It was interesting that the train scores for both models were so close, which suggests that the models fit the training data almost equally well. Having a lower test error score than a train error score, which happened with the MLR regressions, suggests overfitting.

Classification

SVM (Classification 1)

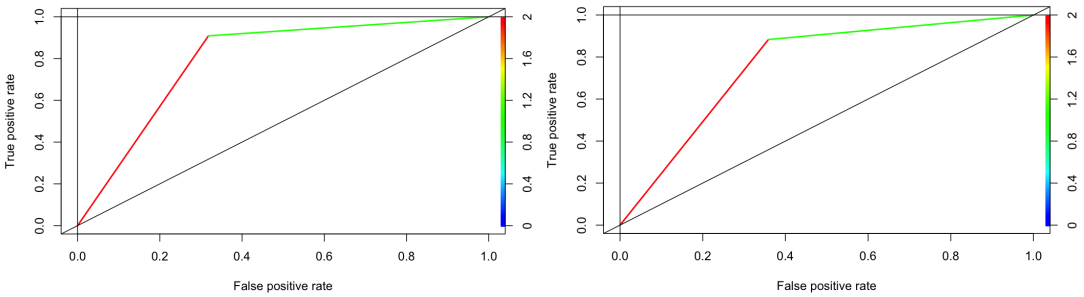
The first classification method I used for my analysis was Support Vector Machine (SVM). The objective of the support vector machine algorithm is to find a hyperplane in an p-dimensional space (p being the number of features) that distinctly classifies the data points. To separate the two classes of data points, there are many possible hyperplanes that could be chosen. The objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some

reinforcement so that future data points can be classified with more confidence. Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM. The parameters that I tuned over were kernel (the choices being "linear", "polynomial", "radial", and "sigmoid") and cost (cost of constraints violation). “Kernel” is used due to a set of mathematical functions used in Support Vector Machine providing the window to manipulate the data. So, Kernel Function generally transforms the training set of data so that a non-linear decision surface is able to transform to a linear equation in a higher number of dimension spaces. The cost parameter decides how much an SVM should be allowed to “bend” with the data. For a low cost, you aim for a smooth decision surface and for a higher cost, you aim to classify more points correctly.



The only assumptions of Support Vector Machines are independent and identically distributed data. Since we know that all of the predictor variables are normally distributed, this implies identical distribution, and so nothing needs to be addressed. To remedy for dependent data we can just drop the ‘density’ variable again.

I tuned kernel and cost at the same time (to ensure that I wasn’t tuning either on the fixed setting for the other parameter) using the svm package’s tune() function, making sure to leave ‘density’ out of the model. I tested over a list of all four of the existing kernels and a list of cost values from 1 to 3 in increments of 0.5 (i.e. 1, 1.5, 2, 2.5, 3). The optimal kernel/cost combination was a radial kernel with a cost of 3. I then trained an SVM model with these parameters on the training data, which gave me a train error rate of 0.1678782; when I ran this model to predict the binary value of quality.good on the testing data, I got back a test error rate of 0.1938776. **SVM optimal model: Radial kernel with cost = 3; Test Error Rate = 0.1938776**



(*Left: ROC graph for SVM training data. Right ROC graph for SVM testing data*)

Bayes Classifiers (LDA, QDA, and Naive Bayes) (Classification 2)

The second classification method I utilized to address the goal of my analysis was Bayes Classifiers. Unlike other classification methods, Bayes Classifiers rely on the Bayes Theorem framework. Bayes Theorem describes the probability of an event, based on prior knowledge of conditions that might be related to the event. Bayes Theorem can be represented mathematically as P(A|B) = P(B|A)P(A)/P(B), where P(A|B) (the posterior probability) is the probability of event A occurring given that B is true, P(A) is the probability of A occurring (known as the prior probability), P(B|A) is the probability of event B occurring given that A is true (typically the Probability Density Function aka PDF when dealing with continuous variables as we are here), and P(B) is the probability of event B occurring (or the total probability formula). For classification, "A" represents "y=k" or the value of the response variable at an observation and "B" represents "X=x" or the value of a single predictor variable. There are three types of Bayes Classifier methods: Linear Discriminant Analysis (LDA) , Quadratic Discriminant Analysis (QDA), and Naive Bayes Classifier. The major difference between LDA and QDA is that LDA assumes that the feature covariance matrices of both classes are the same, which results in a linear decision boundary; in contrast, QDA is less strict and allows different feature covariance matrices for different classes, which leads to a quadratic decision boundary. The difference between QDA and Naive Bayes is that Naive Bayes assumes independence of the features, which means the covariance matrices are diagonal matrices. And while LDA has a shared covariance matrix, Naive Bayes has class-specific covariance matrices. The two tuning parameters are which variables are to be used and which method of Bayes Classifier to use (LDA, QDA, or Naive Bayes).

Both LDA and QDA assume that the predictor variables X are drawn from a multivariate Gaussian (aka normal) distribution. There was no problem with normality in the untransformed data set so I didn't have to adjust for anything there. LDA assumes equal variance; this assumption is relaxed with the QDA model. As determined for the MLR model, the data's untransformed data showed no sign of funneling on scatterplots of the residuals vs the fitted values for each predictor, and thus no adjustments need to be made for this issue with neither LDA nor QDA. LDA and QDA require the number of predictor variables (p) to be less than the sample size (n). This was a given with my data so no correction was necessary. Naive Bayes has only the assumption of independence among predictors. I addressed this issue of

multicollinearity by dropping the ‘density’ variable for the Naive Bayes model, just as I had done for the MLR, Shrinkage and SVM methods.

I tuned the variables to be modeled on by running a 10-fold Cross Validation, which I did manually by splitting my training data into 10 equally measured random mini train datasets and 10 equally measured random mini validation datasets, and plotting each predictor variable against the quality.good response variable 10 times. The strongest correlations I observed in these plots were between quality.good and the predictors ‘alcohol’, ‘total.sulfur.dioxide’, ‘volatile.acidity’, and ‘density’ (the latter which I had to drop for the Naive Bayes model in order to satisfy the assumption of independence among predictors. I tuned on the classifier type simply by running the QDA, LDA, and Naive Bayes models on the same variables, and assessed by finding the mean Train Error rate among all 10 cross validated models. The LDA model returned a Train Error Rate of 0.2554545 and a Test Error Rate of 0.2510204; the QDA model returned a Train Error Rate of 0.2565909 and a Test Error Rate of 0.2367347; the Naive Bayes model returned a Train Error Rate of 0.2747727 and a Test Error Rate of 0.2857143. Although the Naive Bayes model was the only of the three with a test error rate higher then the test error rate (and thus was the only model not overfit), it also had the worst Test Error Rate of the three models. Ultimately I assessed that the QDA model, with the lowest Test Error Rate of 0.2367347 was the optimal Bayes Classifier model. **Bayes Classifier optimal model: QDA model on predictors ‘alcohol’, ‘total.sulfur.dioxide’, ‘volatile.acidity’, and ‘density’; Test Error Rate = 0.2367347**

Classification conclusion: For the optimal SVM model tuning on kernel and cost, the Train Error Rate was 0.1678782 and the Test Error Rate was 0.1938776; For the optimal Bayes Classifier method using a QDA model with the choice of variables tuned by 10-fold Cross Validation, the Train Error Rate was 0.2565909 and the Test Error Rate was 0.2367347, making the Support Vector Machine superior to the Bayes Classifier method in the predicting the binary quality of white wine based of the features included in the White Wine dataset. I was impressed with how well the SVM performed, especially given that the Test Error Rate was larger than the Train Error Rate, suggesting no overfitting. And while the QDA also performed quite well, having a lower test error score than a train error score, which happened with the MLR regressions as well, does suggest overfitting.

Overall conclusion: although it is hard to compare the classification error rate to the regression mean squared error, it does look like the classification models both performed better than the regression models. I potentially attribute this to the classification metric ‘quality.good’ breaking down the 10 ratings of ‘quality’ into two ratings of either ‘good’ or ‘bad’.