Wine Quality Indicators Project

Vito Leonardo, Jiatao Wang, and Wenjin Liu

8/1/2021

Table of Contents

1. Introduction

This project mainly focuses on analyzing the wine quality data, which is available at the UCI Machine Learning Repository. The data contains a set of observations on numbers of red and white wine varieties from the Vinho Verde region in Portugal, but we only consider the red wine data for this project. Vineyards often use taste testers to determine the quality of wine. To reduce the dependency on taste tasters, the goal of this project is to suggest a model that can accurately predict the taste quality of the wine based on available physicochemical measurements. We will recommend two models: one for regression and one for classification. The goal of the regression model is to predict the wine quality score given a certain set of measurements. The goal of the classification model is to predict whether wine quality is high or low based on given measurements of the wine. We consider 5 different regression models/methods and 3 different classification models/methods. In the proceeding sections, we discuss each model we built, how the parameters were chosen, and the resulting prediction efficiency. In the conclusion, we will compare the models and suggest the best regression and classification model. We performed all analysis using R statistical software.

2. The Data

The primary variable that we are predicting from the red wine dataset is the quality of wine. This variable ranges from 0 to 10 where 0 is the lowest quality wine and 10 is highest quality wine. The sample from this dataset only contains wine quality ranging from 3 to 8. Figure 2.1 displays the percentage of the observations in the dataset that have each quality.

The variables available for prediction are fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, and alcohol content. For regression, we will use these variables to predict quality on the scale of 0 to 10.

For classification, we will divide this quality variable into two groups. Samples with quality level less than 6 were assigned low quality, the rest were assigned high quality. As a result, low quality samples account for 46.5% of all samples, while high quality samples account for 53.5%. This can also be seen in Figure 2.1. The goal of the classification models is to predict whether wine quality is high or low.

Figure 2.2.1: Distribution of the Response Variables

Quality	3	4	5	6	7	8	low	high
Percentage	0.6%	3.3%	42.6%	39.9%	12.4%	1.1%	46.5%	53.5%

In order to test the prediction efficiency of our models, the data was randomly split into a training data set and a test. The test set contains 70% of the data while the training set contains 30% of the data. We use the training set to fit our models and the test set to determine its prediction efficiency. For regression models, the prediction efficiency was measured by the Mean Square Error when predicting quality using the test set. For classification models, the prediction efficiency was measured by the Accuracy of the predictions using the test set. The accuracy is the percentage of predictions that were correctly classified.

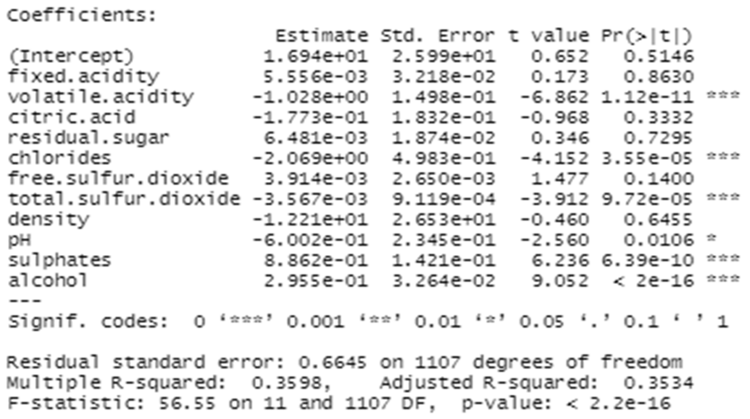
3. Regression Models

3.1. Multiple Linear Regression

3.1.1 Linear Model

We fit the linear model using the training data: response variable(quality) with all other explanatory variables and made a prediction from the model on the test dataset. The test MSE we got from this multiple linear regression is **0.3731942**.

Figure 3.1.1: Linear Model Coefficient Estimates



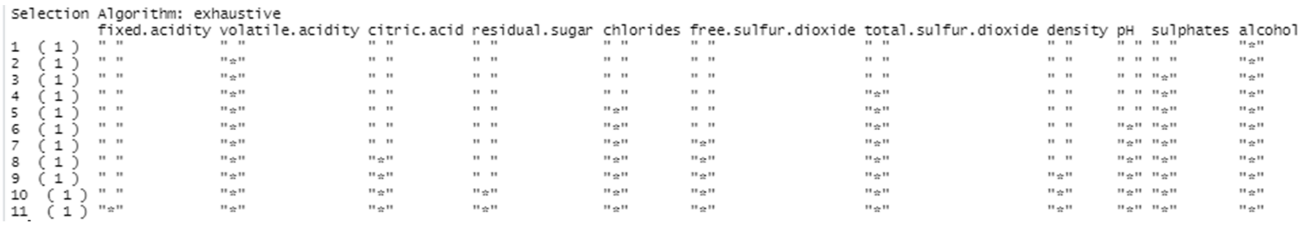
```
Coefficients:
                      Estimate Std. Error t value Pr(>|t|)    
(Intercept)          1.694e+01  2.599e+01   0.652   0.5146    
fixed.acidity        5.556e-03  3.218e-02   0.173   0.8630    
volatile.acidity    -1.028e+00  1.498e-01  -6.862 1.12e-11 ***
citric.acid         -1.773e-01  1.832e-01  -0.968   0.3332    
residual.sugar       6.481e-03  1.874e-02   0.346   0.7295    
chlorides           -2.069e+00  4.983e-01  -4.152 3.55e-05 ***
free.sulfur.dioxide  3.914e-03  2.650e-03   1.477   0.1400    
total.sulfur.dioxide -3.567e-03 9.119e-04  -3.912 9.72e-05 ***
density             -1.221e+01  2.653e+01  -0.460   0.6455    
pH                  -6.002e-01  2.345e-01  -2.560   0.0106 *  
sulphates            8.862e-01  1.421e-01   6.236 6.39e-10 ***
alcohol              2.955e-01  3.264e-02   9.052  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.6645 on 1107 degrees of freedom
Multiple R-squared:  0.3598,	Adjusted R-squared:  0.3534 
F-statistic: 56.55 on 11 and 1107 DF,  p-value: < 2.2e-16
```

Figure 3.1.1 shows that P-values for alcohol, volatile. acidity, sulphates, total sulfur dioxide, chlorides are relatively small, indicating that they are the most significant predictive variables that could explain the response variable for the multiple regression method.

3.1.2 Best Subset Selection

We used the best subset selection method for the regression model.

Figure 3.1.2: Best Models for Each Level of p



```
Selection Algorithm: exhaustive
          fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide total.sulfur.dioxide density pH  sulphates alcohol
1  ( 1 )  " "           " "              " "         " "            " "       " "                 " "                  " "     " " " "       "*"    
2  ( 1 )  " "           "*"              " "         " "            " "       " "                 " "                  " "     " " " "       "*"    
3  ( 1 )  " "           "*"              " "         " "            " "       " "                 " "                  " "     " " "*"       "*"    
4  ( 1 )  " "           "*"              " "         " "            " "       " "                 "*"                  " "     " " "*"       "*"    
5  ( 1 )  " "           "*"              " "         " "            "*"       " "                 "*"                  " "     " " "*"       "*"    
6  ( 1 )  " "           "*"              " "         " "            "*"       " "                 "*"                  " "     "*" "*"       "*"    
7  ( 1 )  " "           "*"              " "         " "            "*"       "*"                 "*"                  " "     "*" "*"       "*"    
8  ( 1 )  " "           "*"              "*"         " "            "*"       "*"                 "*"                  " "     "*" "*"       "*"    
9  ( 1 )  " "           "*"              "*"         " "            "*"       "*"                 "*"                  "*"     "*" "*"       "*"    
10  ( 1 ) " "           "*"              "*"         "*"            "*"       "*"                 "*"                  "*"     "*" "*"       "*"    
11  ( 1 ) "*"           "*"              "*"         "*"            "*"       "*"                 "*"                  "*"     "*" "*"       "*"    
```

Figure 3.1.2 shows that the best three-variable model contains alcohol, volatile acidity, and sulphates.

For the next step, the training MSE and test MSE for the best subset method are conducted.

Figure 3.1.3: Training and Test Error

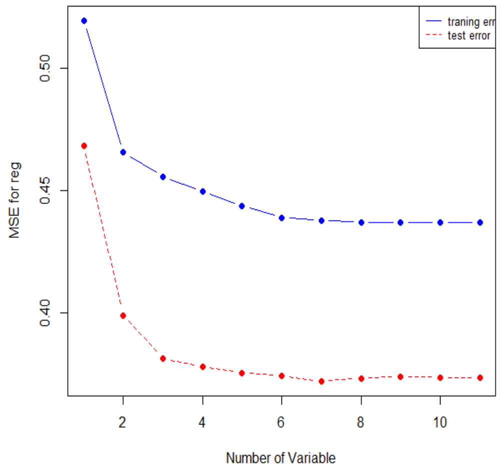


Figure 3.1.3 shows that the response variable is best explained by the number of variables 7 with measuring of MSE. We can see that the training MSE goes horizontally after x-axis reaches 7 and test MSE reaches the minimum at number of variables 7. The minimum test MSE is **0.3719347**.

Figure 3.1.4: Selection Criteria

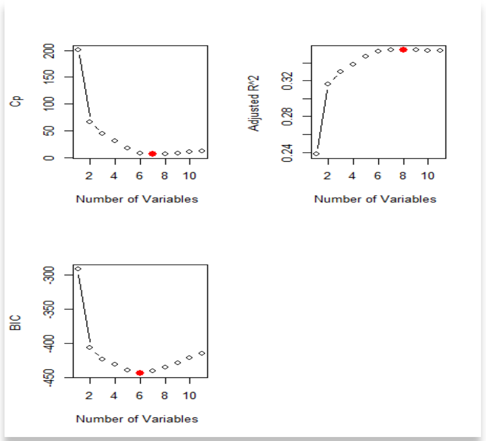


Figure 3.1.4 shows the resulting Adjusted R^2, Mallow's Cp and BIC used to determine the number of variables.

As you can see from the Figure 3.1.4, red dots are identified to be the selected variables for the best model with a given number of predictors. For Mallow's Cp and BIC, they are chosen by the lowest values and for Adjusted R^2, it is indicated by the maximum value. Mallow's Cp gives number of variables 7, while BIC gives 6 and Adjusted R^2 gives 8.

3.1.3 Choosing Among models using the validation set approach.

We also performed the best subset selection for the regression model and make a model matrix for the test dataset. By extracting the coefficients from the best model of that size and multiplying them into the proper columns of the test model matrix, we got the predictions and computed the test MSE.

The test MSE are displaying with the ascending order of number of variables:

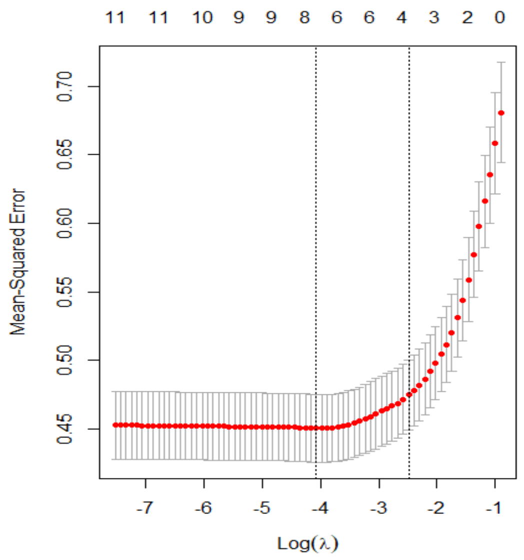
0.4683391 0.3988560 0.3812330 0.3779221 0.3752782 0.3741211 0.3719347 0.3731440 0.3738247 0.3735179 0.3731942 we find that the best model is the one that contains 7 variables. (Indicated in red) This is valid because from the best subset selection method by R^2, Mallow's cp and BIC, the results are 7,8,6 correspondingly. If we want to choose 7 variables from all the explanatory variables, by looking at the results from the summary part. They should be alcohol, volatile acidity, sulphates, total sulfur dioxide, chlorides, PH, free sulfur dioxide.

3.2 Ridge/Lasso Regression

In this section, we discuss the results from the Ridge Regression method and the Lasso Regression method. Both methods use a penalty term to shrink the coefficients closer to zero, increasing bias, and reducing variance with a goal of minimizing MSE. The Ridge Regression penalty term uses the L2 norm which shrinks all coefficients towards zero but does not shrink any of them to zero. The Lasso Regression penalty norm uses the L1 norm which gives Lasso the ability to perform dimension reduction and variable selection by setting coefficients to zero. Typically, these methods are used in high dimensional data. For both models, to choose the best tuning parameter for the penalty term, we used Cross Validation and chose the tuning parameter that resulted in the lowest Cross Validation error.

3.2.1 Ridge Regression

Figure 3.2.1: Ridge Lambda and MSE



From the Figure 3.2.1, the lowest MSE is achieved at around number of variables 7 (indicated by the dotted vertical line on the left side of the figure)

By computing the best lambda for ridge regression, we get the value lambda = 0.006731691

This is very small and it's close to 0. The tuning parameter is very close to 0, indicating that the ridge regression almost equals the least square regression.

As we can see above, the test MSE for the least square's regression is 0.3731942.

The test MSE for the ridge regression is **0.3727866**. This is almost the same as the test MSE for the least square regression. As a result, the ridge regression does not improve the model.

3.2.2 Lasso Regression

Figure 3.2.2 shows the results for the cross validation. The lambda that minimizes the CV error is 0.009669876, which is very close to zero. The coefficients can be found in figure 3.2.3.

Figure 3.2.2: Lasso Lambda and MSE

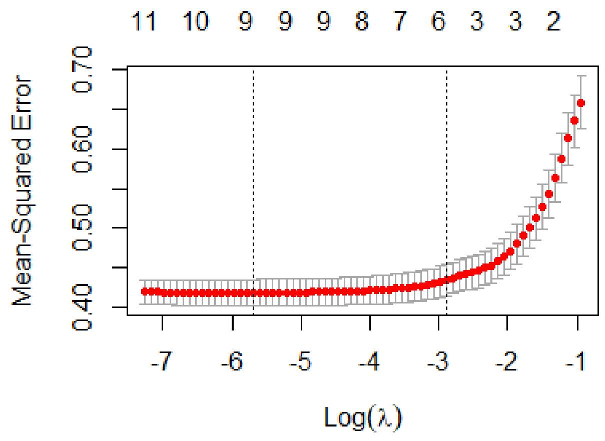


Figure 3.2.3: Lasso Coefficient Estimates

Variables	Coefficients
(Intercept)	9.9684150
fixed acidity	0.0000000
volatile acidity	0.0010006
citric acid	0.0000000
residual sugar	0.0000000
chlorides	-1.0929585
free sulfur dioxide	-1.4491211
total sulfur dioxide	0.0032855
density	-0.0029824
pH	-6.2157870
sulphates	-0.2376072
alcohol	0.7584966

Despite minimizing the CV error and setting 3 unimportant predictors to zero, the test MSE from this model is **0.3886784** which is no better than the Ridge Regression or the Best Subset selection method for Least Squares.

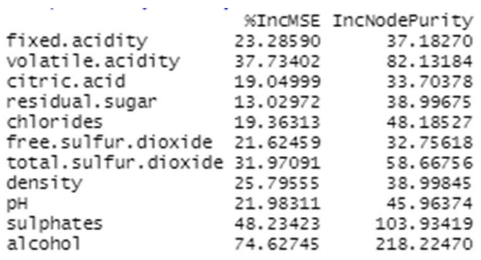
3.3 Random Forest Regression

3.3.1 All Predictors Model

We put all predictors in the model for each split of the tree for the training dataset and considered how this bagged model performs on the test set. The test set MSE associated with the bagged regression tree is **0.3046107**. This is relatively small compared to the linear regression model.

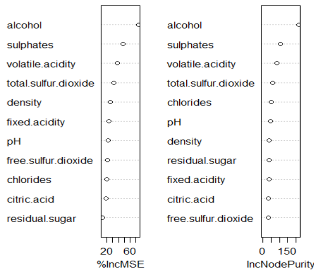
3.3.2 Results

Figure 3.3.1: Random Forest Node Purity 1



```
                      %IncMSE IncNodePurity
fixed.acidity        23.28590      37.18270
volatile.acidity     37.73402      82.13184
citric.acid          19.04999      33.70378
residual.sugar       13.02972      38.99675
chlorides            19.36313      48.18527
free.sulfur.dioxide  21.62459      32.75618
total.sulfur.dioxide 31.97091      58.66756
density              25.79555      38.99845
pH                   21.98311      45.96374
sulphates            48.23423     103.93419
alcohol              74.62745     218.22470
```

Figure 3.3.2: Random Forest Node Purity 2



By viewing the importance of each variable for the all-predictors model, from figure 3.3.1, and figure 3.3.2, we can see that the first measurement: mean decrease of accuracy in predictions on the out of bag samples when a given variable is excluded from the model. Higher percentage means more importance of the variable. We can see from the output that alcohol, sulphates, volatile acidity and total sulfur dioxide are the most important ones.

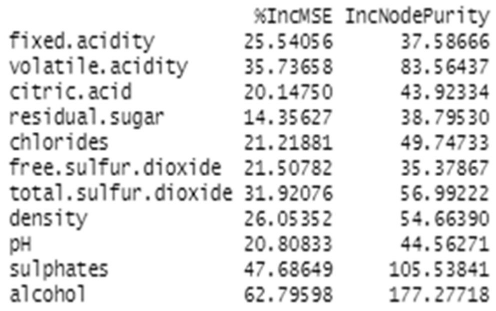
3.3.3 5-Predictors model

We change the number of trees using 5 predictors. And then see how this model performs on the test set. The test set MSE associated with the bagged regression tree is **0.3056664**. There is no significant difference between those two models.

3.3.4 Results

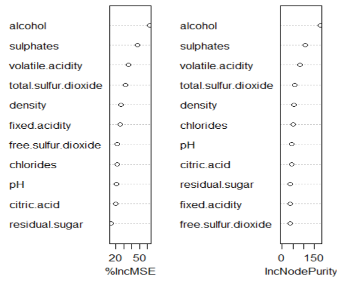
By viewing the Figure 3.3.3 and 3.3.4 for the 5-predictors model. It shows basically the same result as the full-predictor one. It shows basically the same result as the full-predictor one.

Figure 3.3.3: 5 Predictor RF Node Purity 1



```
                     %IncMSE IncNodePurity
fixed.acidity       25.54056      37.58666
volatile.acidity    35.73658      83.56437
citric.acid         20.14750      43.92334
residual.sugar      14.35627      38.79530
chlorides           21.21881      49.74733
free.sulfur.dioxide 21.50782      35.37867
total.sulfur.dioxide 31.92076     56.99222
density             26.05352      54.66390
pH                  20.80833      44.56271
sulphates           47.68649     105.53841
alcohol             62.79598     177.27718
```

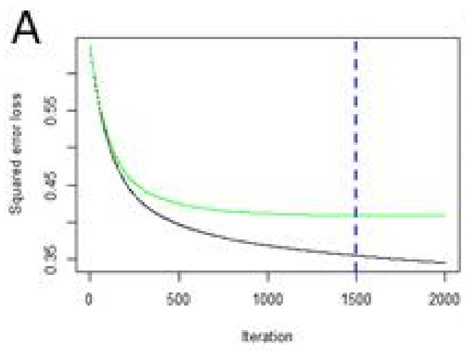
Figure 3.3.4: Predictor RF Node Purity 2

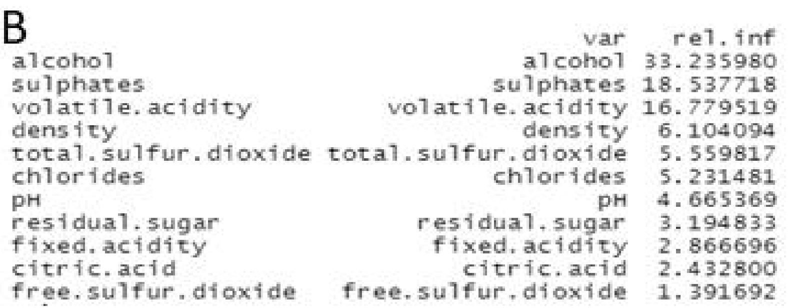


3.4 Generalized Boosted Regression Modeling (GBM)

Boosted regression model builds many sub models, and each model is trying to solve the hard problems the previous models failed to solve. Although boosting algorithm is rather resisting to overfitting, it could still slowly overfit if we apply too many iterations in the training process. Therefore, we used cross validation to choose the best iteration number as shown in Figure 3.4.1A. We used the best iteration number to fit the GBM model. As shown in Figure 3.4.1B, the three most important variables in the model are alcohol, sulphates, and volatile acidity. The model produced training MSE 0.355, and the test MSE is **0.401**.

Figure 3.4.1 Iteration Curve (A) and Feature Importance (B) in GBM Regression Model




B

```
                                      var   rel.inf
alcohol                           alcohol 33.235980
sulphates                       sulphates 18.537718
volatile.acidity         volatile.acidity 16.779519
density                           density  6.104094
total.sulfur.dioxide total.sulfur.dioxide  5.559817
chlorides                       chlorides  5.231481
pH                                     pH  4.665369
residual.sugar             residual.sugar  3.194833
fixed.acidity               fixed.acidity  2.866696
citric.acid                   citric.acid  2.432800
free.sulfur.dioxide   free.sulfur.dioxide  1.391692
```

4. Classification Models

4.1 Logistic Regression

The first classification method we used to predict red wine quality as either "high" or "low" was the logistic regression. The logistic regression is like the standard linear regression except the response variable is a binary or follows a binomial distribution. Because of this, the standard linear model cannot be effectively used to predict it. Instead, the logistic regression connects the predictors using the logit link function and predicts the predicts the probability that an observation falls into a class instead of the class itself. The model is as follows:

$$\log\left(\frac{P(success)}{1-P(success)}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p \qquad then, \qquad P(success) = \frac{e^{\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p}}{1 + e^{\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p}}$$

We use this model to find the probability that an observation belongs to the "success" class. In our case, if probability was greater than 0.5, then we classified the observation to "high." If the observation was less than 0.5, then we classified the observation to "low."

We fit the logistic regression model with all predictors then performed both backwards and forwards stepwise model selection by AIC to choose the best set of predictors.

Figure 4.1.1 shows the coefficient estimates from our model. With a classification threshold of 0.5, we used the model to classify the test data into either the "high" or "low" category. Figure 4.1.2 shows the resulting Confusion Matrix from our predictions using the model.

Figure 4.1.1: AIC Logistic Regression Coefficients

	Coefficients
(Intercept)	-9.2440590
fixed acidity	0.1395021
volatile acidity	-3.5609174
citric acid	-1.4860347
free sulfur dioxide	0.0243792
total sulfur dioxide	-0.0181569
sulphates	2.0004621
alcohol	0.9341621

Figure 4.1.2: Logistic Regression Confusion Matrix

		Reference	
		High	Low
Prediction	High	169	72
	Low	60	179

The resulting Accuracy of the model was **0.75** and the Misclassification Rate of the model was **0.25**.

4.2 Support Vector Classifier

The second classification method that we used was the Support Vector Classifier. The SVC attempts to classify observations into a category by splitting the data into two or more classes using a hyperplane plus a margin that allows for errors in predictions. The hyperplane and margin are chosen using the following constrained optimization.

$$\max_{\beta_0,\beta_1,\dots\beta_p,\epsilon_1,\dots,\epsilon_n} M \ subject\ to \sum_{j=1}^{p} \beta_j = 1\,,\ \ y(\beta_0 + \beta_1 x_{i1} + \dots + \beta_p x_{ip}) \geq M(1-\epsilon_i), \epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C$$

Using Cross Validation, we chose C = 1. After fitting the SVC model with C=1 and using it to predict on the test set. Figure 4.2.1 displays the resulting Confusion Matrix.

Figure 4.2.1: Support Vector Classifier Confusion Matrix

	Reference	
Prediction	High	Low
High	169	72
Low	60	179

The resulting Accuracy of the model was **0.7583333** and the Misclassification Rate of the model was **0.2416667** which offers a slight improvement over the logistic regression. However, the logistic regression offers a more interpretable model that may be preferred by the vineyard.

4.3 Generalized Boosted Classification Modeling (GBM)

4.3.1 Fit a default GBM model

Boosting model builds multiple trees in sequential and less restrictive. By fitting a GBM model with default parameters, we got training accuracy 0.772 and test accuracy 0.760.

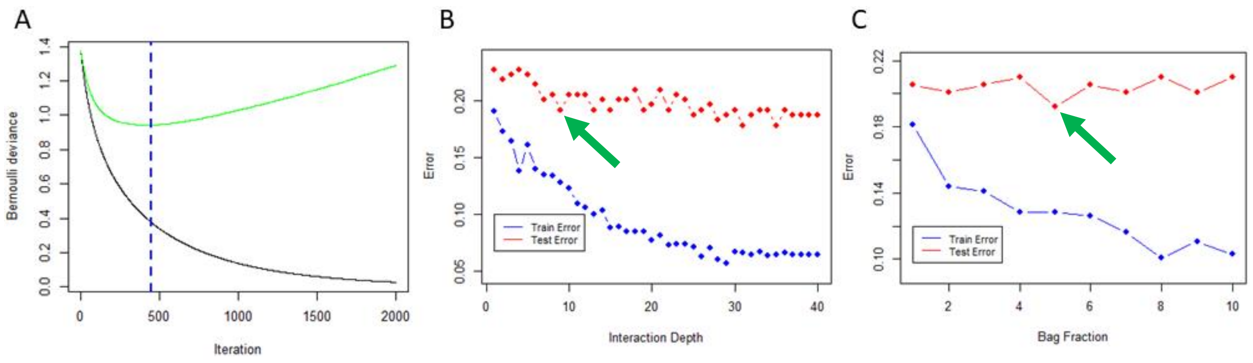
4.3.2 Tuning parameters for GBM model

Boosting method can overfit with more iterations. As shown in Figure 4.3.2A, training error continuously decreases with iteration, while test error shows a U shape. We chose the best iteration when Bernoulli deviance is the lowest marked by the blue dotted line (Figure 4.3.2A).

The interaction depth determines the size of each subtree, and therefore is an important parameter. The boosting tree could overfit with large subtree. We used cross validation to determine the best subtree size. As shown in Figure 4.3.2B, training error continuously decreases with larger subtree, while test error decreases initially before reaching a plateau. We chose the best interaction depth 9 marked by green arrow.

Bag fraction introduces randomness into the model fit. We chose best bag fraction 0.5 by cross validation as marked by green arrow in Figure 4.3.2C.

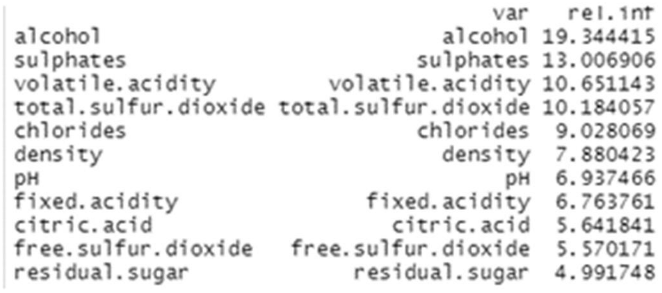
Figure 4.3.2 Cross Validation to Choose Best Iteration (A), Interaction Depth (B) and Bag Fraction (C)



4.3.3 Fit a tuned GBM model

We fit a tuned GBM model with the above chosen best values for parameters. The accuracies for both training and test data set improved (training accuracy 0.881, test accuracy 0.800). The three most important variables are alcohol, sulphates, and volatile acidity (Figure 4.3.3).

Figure 4.3.3 Feature importance in GBM model

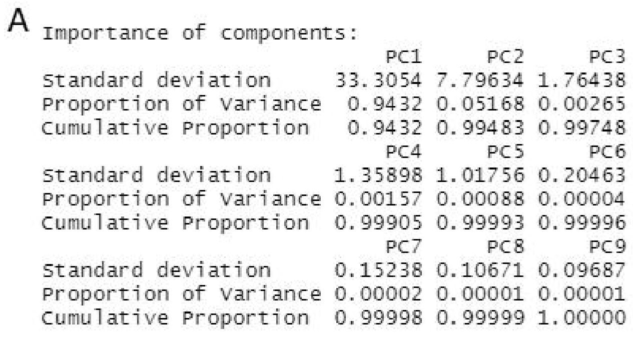


```
                                      var   rel.inf
alcohol                           alcohol 19.344415
sulphates                       sulphates 13.006906
volatile.acidity         volatile.acidity 10.651143
total.sulfur.dioxide total.sulfur.dioxide 10.184057
chlorides                       chlorides  9.028069
density                           density  7.880423
pH                                     pH  6.937466
fixed.acidity               fixed.acidity  6.763761
citric.acid                   citric.acid  5.641841
free.sulfur.dioxide   free.sulfur.dioxide  5.570171
residual.sugar             residual.sugar  4.991748
```

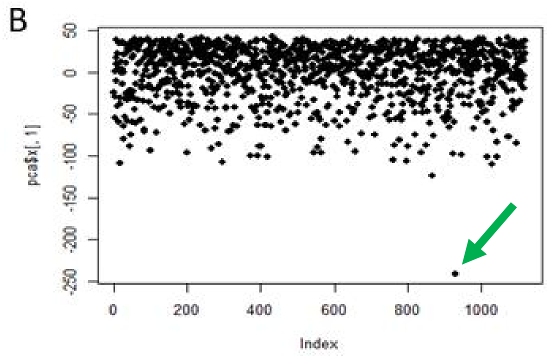
4.3.4 The effect of removing outliers

Boosting method is sensitive to outliers. We performed principal component analysis. As shown in Figure 4.3.4A, the first component alone captures 94% variance. We used principal component analysis to remove outlier indicated by green arrow (Figure 4.3.4B). We then fit a GBM model, the accuracy did not improve (training accuracy 0.890, test accuracy 0.785), likely because there is only one outlier.

Figure 4.3.4 Principal Component Analysis



```
Importance of components:
                           PC1     PC2     PC3
Standard deviation     33.3054 7.79634 1.76438
Proportion of Variance  0.9432 0.05168 0.00265
Cumulative Proportion   0.9432 0.99483 0.99748
                           PC4     PC5     PC6
Standard deviation     1.35898 1.01756 0.20463
Proportion of Variance 0.00157 0.00088 0.00004
Cumulative Proportion  0.99905 0.99993 0.99996
                           PC7     PC8     PC9
Standard deviation     0.15238 0.10671 0.09687
Proportion of Variance 0.00002 0.00001 0.00001
Cumulative Proportion  0.99998 0.99999 1.00000
```



4.3.5 The final boosting model

The final GBM model produced greatly improved accuracies (training accuracy 0.881, test accuracy **0.800**).

5. Conclusions

In this project, we tried six different regression models: linear regression (full, best subset, ridge, and lasso), random forest, and generalized boosted modeling. Ridge regression (test MSE 0.373) performs better than lasso regression (test MSE 0.389). Best subset performs exhaustive search of all possible models, and therefore usually have best performance (test MSE 0.372) in linear regression models. Linear regression models have the same limitation that they work best when the relationship is linear and response variable is continuous. In the wine data set, quality is an ordinal variable which is not a strictly continuous variable. The magnitude of the difference between adjacent levels (e.g., levels 4-5, 5-6) could vary greatly. The other two regression models we tried are random forest and generalized boosted modeling. These are more flexible models which requires neither linear relationship nor continuous response variable, which seems better fit for this wine data set. This is exactly what we observed in the analysis. Best performance was achieved by random forest (test MSE 0.305), which is better than boosting model (test MSE 0.401). To predict quality of wine, we recommend the 5-predictor random forest model discussed in section 3.3.

We also explored three different classification models: logistic regression, the support vector classifier, and generalized boosted modeling. They achieved 75%, 75.8%, 80.0% accuracy respectively. Logistic regression is a more restrictive model which works better if the true decision boundary is linear. Support vector machine works better when samples can be well separated by the curve defined by the kernel function. Boosting model builds multiple trees in sequential and less restrictive. Each subsequent tree tries to solve hard problems left by previous trees. The default classifier, when we simply assign every sample to high quality, has an accuracy of 53.5%. All of our classification models have an accuracy significantly greater than 53.5%, therefore, they are all reasonably good models. If prediction accuracy is the primary concern, then the GBM model from section 4.3 is the model that we suggest. However, since these models all produce similar accuracy, they are all reasonable choices. If interpretability is of concern, the logistic model may be better. If there is area of concern that may want to be minimized or maximized, such as Specificity or Sensitivity, the models would have to be explored my thoroughly.

Overall, the three most important variables identified by all our regression and classification models are alcohol, sulphates, and volatile acidity. The results produced by all the models are reasonably good, but not excellent. There are several possible causes. It is likely that some important chemical variables are not measured and not included in this data set. It is also likely that wine quality is a rather subjective measure. Different tasters may have different preferences and therefore assign different quality values for the same wine, which complicates the relationship for wine quality. The models could potentially be greatly improved if the data set includes taster information which is likely an important variable.

6. Appendix (R Code)

Multiple Linear Model

```
setwd("C:\\Users\\CKA\\Downloads")
set.seed(123)
#read data
data.red <- read.csv("C:/Users/CKA/Downloads/winequality-red.csv",header=TRUE,sep = ";", fill =
TRUE)
sample <- sample.int(n = nrow(data.red), size = floor(.7*nrow(data.red)), replace = F)
train <- data.red[sample, ]
test <- data.red[-sample, ]
#regression method contaning cp, adjR^2 BIC
lm <- lm(quality ~., data =train)
summary(lm)
p <- predict(lm, newdata= test)
test.MSE <- mean((p-test$quality)^2)
```

Best Subset Selection

```
library(leaps)
regression1 <- regsubsets(quality ~., data = train,nvmax=11)
hh1<-summary(regression1)
#test for training and test mse
MSE1 <- rep(NA,11)
train_matrix <- model.matrix(quality ~ ., data = train, nvmax = 11)
for (i in 1:11){
  coefi <- coef(regression1, id = i)
  pred1 <- train_matrix[, names(coefi)] %*% coefi
  MSE1[i] <- mean((pred1 - train$quality)^2)
}
plot(MSE1, xlab = "Number of Variable", ylab = "MSE for reg", pch = 19, type = "b",col="blue")
MSE2 <- rep(NA,11)
test_matrix <- model.matrix(quality ~ ., data = test, nvmax = 11)
for (i in 1:11){
  coefi <- coef(regression1, id = i)
  pred2 <- test_matrix[, names(coefi)] %*% coefi
  MSE2[i] <- mean((pred2 - test$quality)^2)
}
plot(MSE2, xlab = "Number of Variable", ylab = "MSE for reg", pch = 19, type = "b",col="red")

matplot( cbind(MSE1,MSE2),type="b",col=c("blue","red"),pch=19,xlab = "Number of Variable", ylab =
"MSE for reg" )
legend("topright", legend =c("traning error", "test error"), col=c("blue","red"),lty=1:2, cex=0.8)

#plots for the number of variable associated with mallow cp, adjusted R^2 and BIC.
par(mfrow=c(2,2))
which.min(hh1$cp)
plot(hh1$cp ,xlab="Number of Variables ",ylab="Cp", type='b')
points (7,hh1$cp [7], col ="red",cex=2,pch =20)
```

```
which.max(hh1$adjr2)
plot(hh1$adjr2 ,xlab="Number of Variables ",ylab="Adjusted R^2 ", type='b')
points (8,hh1$adjr2 [8], col ="red",cex=2,pch =20)
which.min(hh1$bic)
#return value = 3
plot(hh1$bic ,xlab="Number of Variables ",ylab="BIC ", type='b')
points (6,hh1$bic [6], col ="red",cex=2,pch =20)
```

Validation Set Approach

```
#The same as test mse for the best subset approach
regfit.best=regsubsets (quality~.,data=train,
                      nvmax=11)
test.mat=model.matrix(quality~.,data=test)
val.errors =rep(NA ,11)
for(i in 1:11){
   coefi=coef(regfit.best ,id=i)
  pred=test.mat[,names(coefi)]%*%coefi
 val.errors[i]=mean(( test$quality-pred)^2) }
```

Ridge/Lasso regression

```
set.seed(1)
#ridge regression
library(Matrix)
library(glmnet)
train.ridge <- model.matrix(quality ~ ., data = train)
test.ridge <- model.matrix(quality ~ ., data = test)
ridge <- glmnet(train.ridge, train$quality, alpha = 0)
cv.ridge <- cv.glmnet(train.ridge, train$quality, alpha = 0)
plot(cv.ridge)
bestlam.ridge <- cv.ridge$lambda.min
#0.006731691
best.ridge <- glmnet(train.ridge, train$quality, alpha = 0, lambda = 0.006133666 )
coef(best.ridge)
pred2 <- predict(best.ridge, s = bestlam.ridge, newx = test.ridge)
mean((pred2 - test$quality)^2)
cv.out <- cv.glmnet(X.train, Y, alpha=1)
plot(cv.out)
bestlam <- cv.out$lambda.min
lasso.fit <- glmnet(X.train,Y, alpha = 1, lambda = bestlam)
lasso.coef <- predict(lasso.fit, type = "coefficients")
print(lasso.coef)
lasso.pred <- predict(lasso.fit, newx= X.test)
lasso.MSE <- mean((lasso.pred - RedWine.Test$quality)^2)
```

Random Forest Regression

```
library("randomForest")
bag.red <- randomForest(quality ~ ., data = train, mtry = 11,
                      importance = T)
random1<-randomForest(formula = quality ~ ., data = train, mtry = 11, importance = T)
random2<-randomForest(formula = quality ~ ., data = train, mtry = 5, importance = T)
random.pred1 <- predict(random1, test)
random.pred2 <- predict(random2, test)
MSE.random1 <- mean((random.pred1 - test$quality)^2)
MSE.random2 <- mean((random.pred2 - test$quality)^2)
importance(random1)
importance(random2)
varImpPlot(random1)
varImpPlot(random2)
```

Generalized Boosted Regression Modeling (GBM)

```
library(class)
setwd("E:\\courses\\ST563\\Final_project")
rawdata <- read.csv("winequality-red.csv")
raw_x = rawdata[-12]
raw_y = rawdata[12]
set.seed(8)
trainid <- sample(1:nrow(rawdata), nrow(rawdata)*0.7, replace=F)
train <- rawdata[trainid,]
test <- rawdata[-trainid,]
set.seed(1)
par(mfrow=c(2,2))
gbm_model <- gbm(quality~., data=train, distribution="gaussian", n.trees=2000, shrinkage=0.01,
cv.folds=5)
best_tree_n <- gbm.perf(gbm_model)
test_predict <- predict(gbm_model, newdata=test, n.trees=best_tree_n, type="response")
mean((test_predict - test$quality)^2)
summary(gbm_model)
train_predict <- predict(gbm_model, newdata=train, n.trees=best_tree_n, type="response")
mean((train_predict - train$quality)^2)
```

Logistic Regression

```
glm.fit <- glm(QualityCat ~ . - quality, data = RedWine.Train, family = binomial)
glm.fit2 <- stepAIC(glm.fit, direction = "both", trace = FALSE)
glm.prob <- predict(glm.fit2, newdata = RedWine.Test[,-c(12,13)], type = "response")
glm.pred <- factor(ifelse(glm.prob >.5, 1,0), labels = c("high", "low"))
kable(data.frame(Coefficients = glm.fit2$coefficients))
(ErrorMatrix <- confusionMatrix(data =  glm.pred, reference = RedWine.Test$QualityCat))
print(ErrorMatrix$table)
ErrorMatrix$overall[1]
(glm.missclass <- 1 - ErrorMatrix$overall[1])
```

Supper Vector Classifier

```
library(e1071)
svmfit <- svm(QualityCat ~ ., data= RedWine.Train[,-12], kernal = "linear", cost = 10, scale = FALSE)
summary(svmfit)
tune.out <- tune(svm,QualityCat ~ ., data= RedWine.Train[,-12], kernal = "linear",
    ranges = list(0.001, 0.01, 0.1,0.5,.75, 1,1.25, 2, 5, 10, 25, 50, 100))
summary(tune.out)
```

```
bestmod <-tune.out$best.model
summary(bestmod)
svm.pred <- predict(bestmod, RedWine.Test)
ErrorMatrix <- confusionMatrix(data = svm.pred, reference = RedWine.Test$QualityCat)
print(ErrorMatrix$table)
ErrorMatrix$overall[1]
(SVC.missclass <- 1 - ErrorMatrix$overall[1])
```

Generalized Boosted Classification Modeling (GBM)

```
library(class)
rawdata <- read.csv("winequality-red.csv")
prop.table(table(rawdata$quality))
rawdata$quality <- ifelse(rawdata$quality<6, 0, 1)
prop.table(table(rawdata$quality))
raw_x = rawdata[-12]
raw_y = rawdata[12]
set.seed(8)
trainid <- sample(1:nrow(rawdata), nrow(rawdata)*0.7, replace=F)
train_x <- raw_x[trainid,]
train_y <- raw_y[trainid,]
train_y <- raw_y[trainid,]
test_y <- raw_y[-trainid,]
train <- rawdata[trainid,]
test <- rawdata[-trainid,]
get_accuracy_1 <- function(model, train, test) {
  train_predict <- predict(model, newdata=train, type="response")
  train_predict <- ifelse(train_predict > 0.5, 1, 0)
  train_error <- mean(train_predict != train$quality)
  test_predict <- predict(model, newdata=test, type="response")
  test_predict <- ifelse(test_predict > 0.5, 1, 0)
  test_error <- mean(test_predict != test$quality)
  return(c(1-train_error, 1-test_error))
}
get_accuracy_2 <- function(model, best_tree_n, train, test) {
  train_predict <- predict(model, newdata=train, n.trees=best_tree_n, type="response")
  train_predict <- ifelse(train_predict > 0.5, 1, 0)
  train_error <- mean(train_predict != train$quality)
  test_predict <- predict(model, newdata=test, n.trees=best_tree_n, type="response")
  test_predict <- ifelse(test_predict > 0.5, 1, 0)
  test_error <- mean(test_predict != test$quality)
  return(c(1-train_error, 1-test_error))
}
# Default gbm parameters
require(gbm)
set.seed(1)
gbm_model <- gbm(quality~., data=train)
default_accuracy <- get_accuracy_1(gbm_model, train, test)
print(default_accuracy)
# Future divide train set into train and test
set.seed(1)
```

```
trainid <- sample(1:nrow(train), nrow(train)*0.8, replace=F)
train_train <- train[trainid,]
train_test <- train[-trainid,]
# Tune parameter interaction.depth
set.seed(1)
max_depth <- 40
train_errors <- rep(NA, max_depth)
test_errors <- rep(NA, max_depth)
for (i in 1:max_depth) {
  set.seed(1)
  gbm_model <- gbm(quality~., data=train_train, n.trees=2000, shrinkage=0.01, interaction.depth=i,
                 class.stratify.cv=T, cv.folds=5)
  best_tree_n <- gbm.perf(gbm_model)
  accuracy <- get_accuracy_2(gbm_model, best_tree_n, train_train, train_test)
  train_errors[i] <- 1 - accuracy[1]
  test_errors[i] <- 1 - accuracy[2]
}
par(mfrow=c(2,2))
y_min <- min(min(train_errors), min(test_errors)) - 0.01
y_max <- max(max(train_errors), max(test_errors)) + 0.01
plot(train_errors, xlab='Interaction Depth', ylab="Error", type="b", pch=16, col="blue", ylim=c(y_min,
y_max))
lines(test_errors, type="b", pch=16, col="red")
legend(1, 0.1, legend=c("Train Error", "Test Error"), col=c("blue", "red"), lty=1:1, cex=0.8)
# Tune parameter bag.fraction
set.seed(1)
train_errors <- rep(NA, 10)
test_errors <- rep(NA, 10)
for (i in 1:10) {
  set.seed(1)
  gbm_model <- gbm(quality~., data=train_train, n.trees=2000, shrinkage=0.01, interaction.depth=9,
                 bag.fraction=i/10, class.stratify.cv=T, cv.folds=5)
  best_tree_n <- gbm.perf(gbm_model)
  accuracy <- get_accuracy_2(gbm_model, best_tree_n, train_train, train_test)
  train_errors[i] <- 1 - accuracy[1]
  test_errors[i] <- 1 - accuracy[2]
}
par(mfrow=c(2,2))
y_min <- min(min(train_errors), min(test_errors)) - 0.01
y_max <- max(max(train_errors), max(test_errors)) + 0.01
plot(train_errors, xlab='Bag Fraction', ylab="Error", type="b", pch=16, col="blue", ylim=c(y_min,
y_max))
lines(test_errors, type="b", pch=16, col="red")
legend(1, 0.12, legend=c("Train Error", "Test Error"), col=c("blue", "red"), lty=1:1, cex=0.8)
# Tuned model
set.seed(4)
gbm_model <- gbm(quality~., data=train, n.trees=2000, shrinkage=0.01, interaction.depth=9,
               bag.fraction=0.5, class.stratify.cv=T, cv.folds=5)
best_tree_n <- gbm.perf(gbm_model)
accuracy_1 <- get_accuracy_2(gbm_model, best_tree_n, train, test)
print(accuracy_1)
```

```
summary(gbm_model)
# PCA
pca <- prcomp(train_x)
summary(pca)
plot(pca$x[, 1], pch=19)
outlier <- which(pca$x[, 1] < -150)
train2 <- train[-c(outlier),]
# PCA + Tuned model
set.seed(4)
gbm_model <- gbm(quality~., data=train2, n.trees=2000, shrinkage=0.01, interaction.depth=9,
            bag.fraction=0.5, class.stratify.cv=T, cv.folds=5)
best_tree_n <- gbm.perf(gbm_model)
accuracy_2 <- get_accuracy_2(gbm_model, best_tree_n, train2, test)
print(accuracy_2)
summary(gbm_model)
print(c(default_accuracy, accuracy_1, accuracy_2))
```