Analysis of Wine Quality Data

Robert West, Amirreza Sahebi Fakhrabad, Mohsen Sahraei Ardakani, Qiaozhi Bao, Nasrin Alizadeh

November 2020

Executive Summary

Identifying any relation between the chemical components of the wine and its desirability can be interest for wine companies. In this project we analyse the red wine data consisting of several chemical properties of wine and quality ratings to shed light on this question. The data contains observations on the chemical components of red wine from the Vinho Verde region in Portugal.
For the first step, our team did some preprocessing analysis to prepare the data for the different regression and classification methods. For this purpose, we checked the normality of the variables and also detected the outliers to improve the performance of the methods. The team decided to more focus on investigating classification methods since the target variable is an ordinal variable and contains the rates from 1 to 10 by tasters. We considered a variety of models including: logistic regression, LDA and QDA, KNN method, tree based methods and SVM. The models were compared with another based on the test errors.
Random Forest has the best average accuracy on test set with 19.4% error, but bagging works competitively close and in some cases better. SVM and Boosting are other competitive algorithms. The team also figured out some variables such as "residual.sugar", "citric.acid", "fixed acidity" and "chlorides" do not play a significant role in the quality of the wine. On the other side, "alcohol", "sulphates", and "volatile.acidity" are the most significant factors effecting the ratings.

1 Introduction

The quality of wine and capturing the satisfaction of the customers can raise the question if there is any chemical factors influencing the quality of wine from customers' view of point. Our team explore red wine data to answer this question and determine the significant factors to classify the high quality wine.
In the following, we first present the analyse conducted to preprocess the data. Then, the methods are provided with the detailed results. In the results section we compare the methods according to their test errors. And finally, we have the conclusion and appendix.

1.1 Data

Data is collected on 12 different variables and consisted of 1599 observations. The properties of wine that are measured are: fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol and output variable (based on sensory data) quality (score between 1 and 10). All these properties are continuous but not quality, which is categorical variable.

As a part of the data pre-processing for statistical analysis, the team performed several exploratory analysis on the dataset. The dataset has no missing data. Furthermore, we provided the correlation between all the properties in Figure1. There is a high correlation between "fixed.acidity" with "pH", "density" and "citric.acid". That is why our team decided to perform lasso to explore the possibility of removing some of the variables.

After implementing Lasso we decided to keep "volatile.acidity", "chlorides", "pH", "sulphates" and "alcohol" and created a new variable "ratio_sulfor.dioxide" which is ratio of free and total sulfor dioxide.

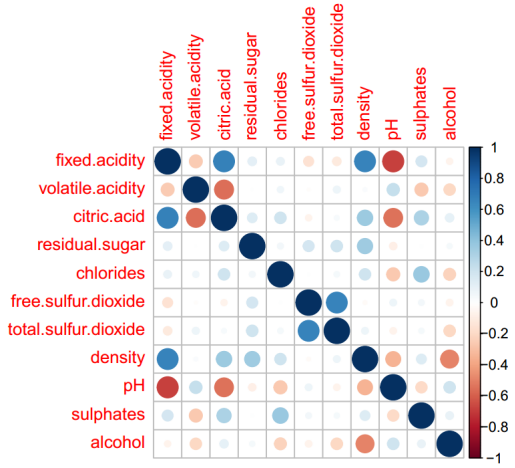


Figure 1: Correlation

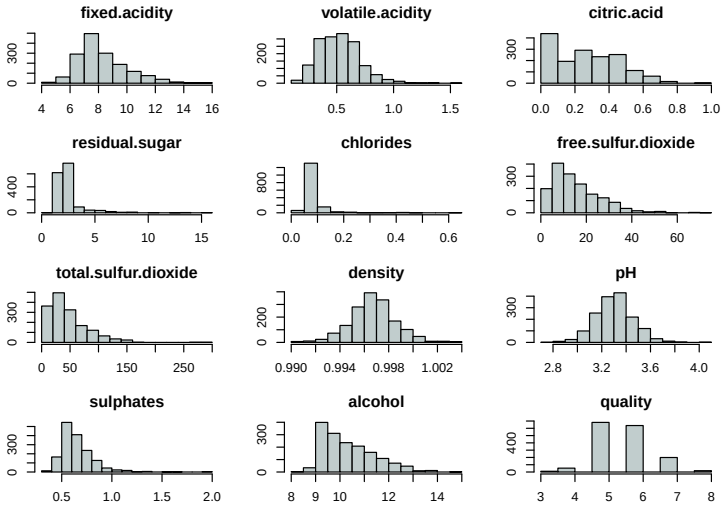


Figure 2: Histogram

We also checked Normality of the predictors which is an assumption for methods like LDA and QDA. The easiest way to check normality is looking at the histograms of variables. As shown in figure (2), "chlorides", "sulphates" and "alcohol" have distributions far from Normal. So we implement some transformations to make them more close to Normal. (We just checked selected variables from lasso because LDA and DQA are the only models require Normality along with variable selection). Specifically, log_{10}, is used for "alcohol" and "sulphates", and square root is used for "alcohol".

Detecting and removing outliers is the last step in our data cleaning process. We found outliers by looking at each selected variable's histogram and in total 37 observations are removed. We put the detailed analysis in the Appendix.

Note that some not all methods require variable selection, outlier detection or normality transformation. Thus we use them as needed.

Regarding the response "quality", we decided to redefine it as two-level factor. For this new qualitative response, classes 3,4,5 are labeled as "Low" and classes 6,7,8 are labeled as "High" quality. We also explored other splits with more than two classes, and different two level ones. However, the best performance achieved in the split we chose. We could also use original quality levels as 6 separated classes, but there were some drawbacks: 1) Some levels (like 3 and 4) have small number of observations so their test error were not reliable 2) Some methods like QDA does not work properly when classes have small number of observations 3) Logistic Regression requires two classes.

2 Methods

In this section, the methods including: logistic regression, LDA, QDA, KNN, decision tree, random forest, bagging, boosting and SVM are presented in detail. For each method, we report the related figures and analysis.

2.1 Logistic Regression

Logistic Regression is a productive classification technique to allow us predict a qualitative response. In this section, we use the predictors selected by lasso analysis,then look for the best model. The performance of the optimal model evaluated with accuracy and test error rates. The team investigated six different models,using cross validation method to get accuracy measurement which describes how well each model works on the training set.Our candidate models are as follows:

Model 1	$qualityBinary \sim volatile.acidity + chlorides + pH + sulphates + alcohol + ratio\_sulfur.dioxide$
Model 2	$qualityBinary \sim volatile.acidity + chlorides + pH + sulphates + I(sulphates^2) + I(sulphates^3) + alcohol + ratio\_sulfur.dioxide$
Model 3	$qualityBinary \sim chlorides + pH + sulphates + I(sulphates^2) + I(sulphates^3) + alcohol + ratio\_sulfur.dioxide + I(ratio\_sulfur.dioxide^2)$
Model 4	$qualityBinary \sim chlorides * pH + chlorides * sulphates + chlorides * alcohol + ratio\_sulfur.dioxide$
Model 5	$qualityBinary \sim volatile.acidity + pH + sulphates + I(sulphates^2) + I(sulphates^3) + alcohol + ratio\_sulfur.dioxide$
Model 6	$qualityBinary \sim volatile.acidity + chlorides + pH + sulphates + I(sulphates^2) + I(sulphates^3) + alcohol + ratio\_sulfur.dioxide + I(ratio\_sulfur.dioxide^2)$

Table 1: Explored models for logistic regression

For investigating main effects we add the variables simply. Combination terms and high order terms are also involved in the six models for investigating multiple effect.As the high-correlated variables are removed in the data exploration, the six candidate models actually have similar performance with very close accuracy values.So it would be meaningless to discuss tiny discrepancies between them.

-	Accuracy	Kappa	Accuracy SD	KappaSD
Model 1	0.7292410	0.4565879	0.0465328	0.0919803
Model 2	0.7405079	0.4780093	0.0637134	0.1283770
Model 3	0.7156635	0.4265812	0.0228495	0.0466375
Model 4	0.7228656	0.4423137	0.0568100	0.1139057
Model 5	0.7411394	0.4784766	0.0355358	0.0731631
Model 6	0.7388798	0.4750547	0.0445406	0.0866685

Table 2: Logistic Models Accuracy for Training set

For the next step, we test models on the test set and get the accuracy of each candidate model for the test set. The model with the highest Accuracy and Kappa would be the best model.

-	Accuracy	Kappa
Model 1	0.7476038	0.4935794
Model 2	0.7603834	0.5192209
Model 3	0.7380192	0.4732154
Model 4	0.7284345	0.4551170
Model 5	0.7635783	0.5254272
Model 6	0.7603834	0.5192209

Table 3: Logistic Models Accuracy for Test set

According to the accuracy and Kappa which are indicators of model performance, we found that Model 5 performs well on both train and test sets. We choose Model 5 as the optimal model and report its accuracy and test error in the results section.

2.2 Linear and Quadratic Discriminant Analysis

LDA and QDA are two classification techniques that can be used when response has multiple classes. These methods are closely related to Bayes' classification and work well when predictors follow normal or close to normal distribution. The difference between the methods is that LDA assumes a common covariance over all

classes but QDA has specific covariance for each class. As discussed in section 2.1, we have implemented some transformation to make them approximately Normal. Although multi-level classification is one of the motivations to perform LDA and QDA, our analysis showed that two class response (3,4,5 as "low" and 6,7,8 as "high") has the best performance with error rates of 25% for LDA and 28% for QDA.

2.3 K-Nearest Neighbors

KNN can be used both in regression and classification problems which is especially appealing since it has only one model parameter that needs tuning and balances the complexity and interpretability. The response in the case of regression or class label in the classification problem is determined by response in the K nearest samples in the training set. The KNN classification takes the majority vote of the labels in the k nearest neighbors and assigns the label of the test point. In the regression problem the only difference is that instead of majority vote average of the responses in the nearest neighbors decides the quality of test wine sample. After only keeping the variables mentioned in section 2.1 both KNN classification and regression are explored on the dataset. The performance is cross validated with different values of K and best K is reported.

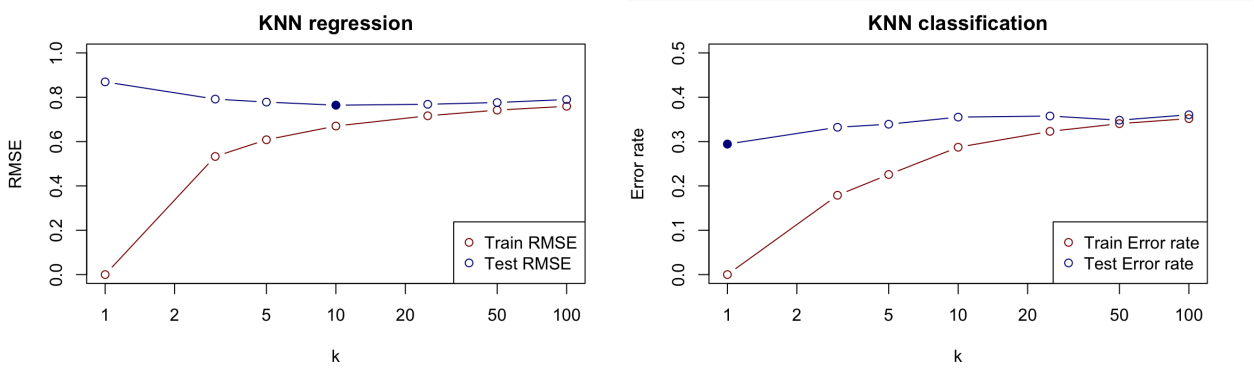


Figure 3: The performance of KNN vs the value of K

As depicted in Figure 3 the lowest test RMSE for regression happens at $k = 10$ with the $RMSE = 0.76$. and for the classification with two classes of high quality and low quality the minimum test error rate takes place at $k = 1$ with the error rate of 29%.

2.4 Tree based Methods

Tree based methods are some supervised learning algorithms that are mostly used for regression and classification. These methods are easy to interpret and competitive with other classification methods. In this section, we discuss four of these methods including decision trees, bagging, random forest and boosting. The data used in these algorithms is the original data without outliers.

2.4.1 Decision Trees

Decision Tree uses the if-then rule to classify and also predict the variables, and enables us to present the decisions virtually. The team applied the decision tree method to the data with two-level quality response and provide the Figure 4. The tree has 9 internal nodes and 10 terminal nodes and shows the frequency of the observations in each terminal node. We can observe the most important feature in determining the quality of red wine is alcohol which stands at the top of the tree. Furthermore, tree has same number of terminal nodes for Low and High quality.

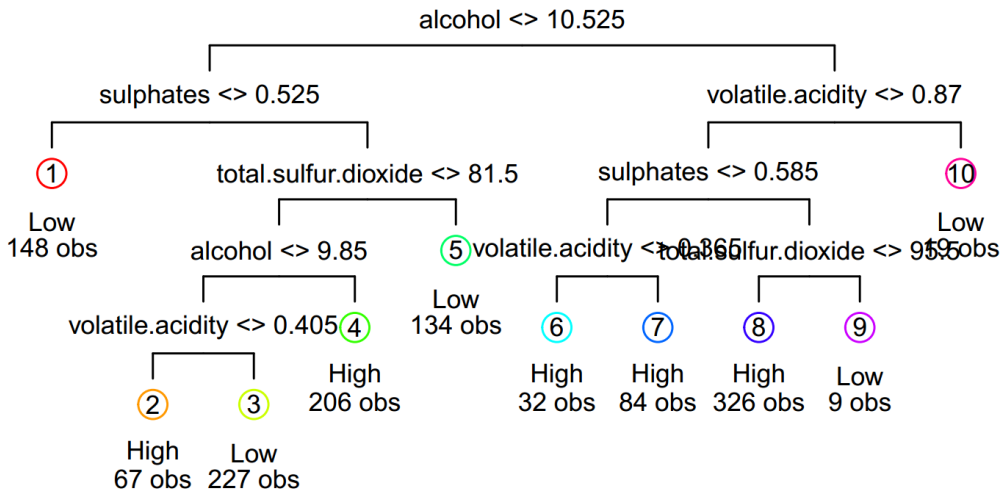


Figure 4: Decision Tree

2.4.2 Bagging and Random Forest Models

Bagging is a technique to create many decision trees while ensuring their variety with random sampling. In fact, the method reduces the high variance we may counter in the decision tree method. We implement this method along with random forest. For the bagging part, we used all the predictors and the results are as below.

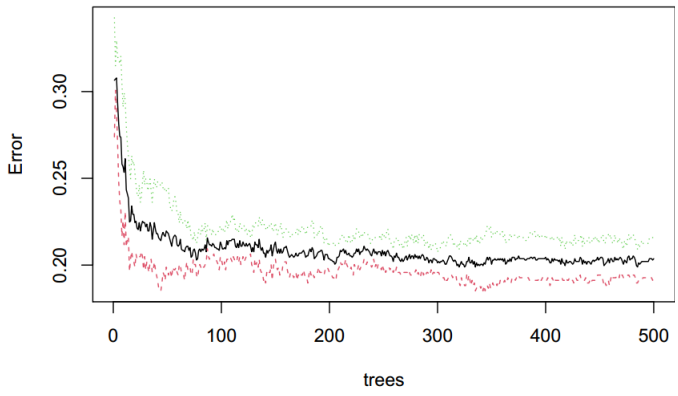


Figure 5: Tree vs Error for bagging model

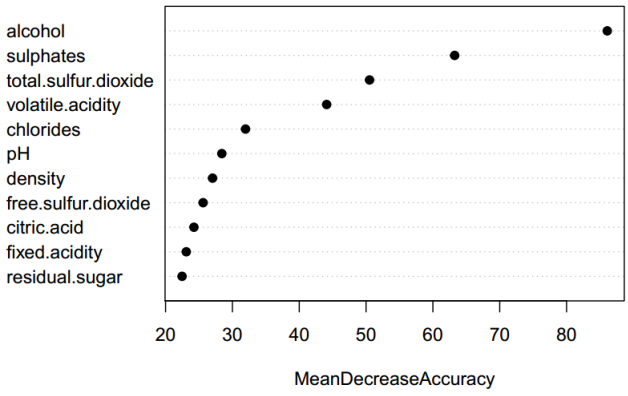


Figure 6: Important variables according to bagging

We can observe that four important features are "alcohol", "sulphates", "total.sulfur.dioxide" and "volatile.acidity". It seems that the error is almost constant after 250 trees. We will present the test errors for the methods in the results section.
We performed a cross-validation to figure out the best number of predictors in random forest model. Although there is not a significant difference in the accuracy of the models, 3 predictor model shows better results, so we set $mtry = 3$. We have almost the same important predictors except density comparing with bagging. We also set the number of trees equal to 10000 but there is not much difference after less that 1000 trees.

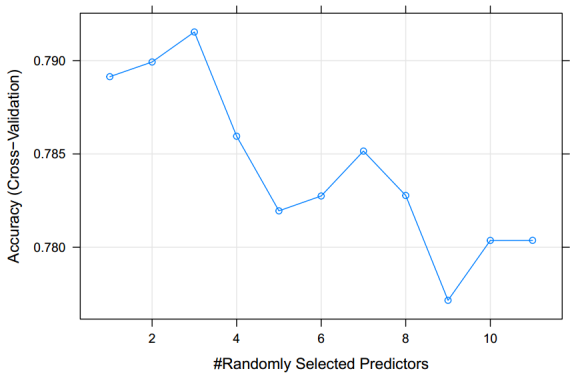


Figure 7: Accuracy vs number of predictors

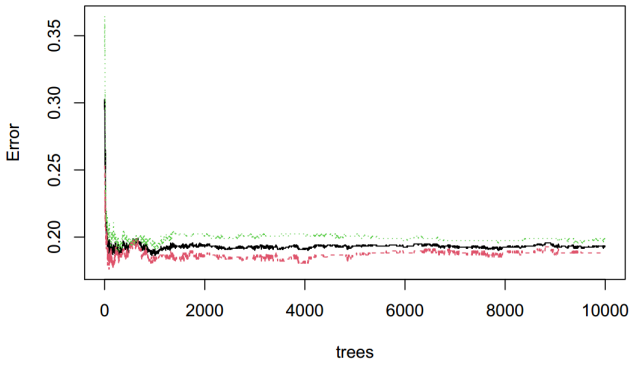


Figure 8: Trees vs Error for random forest

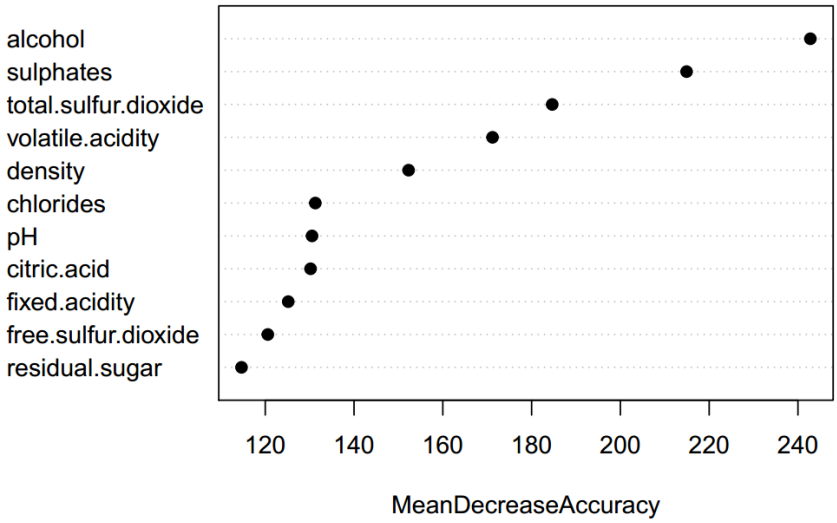


Figure 9: Important factors based on random forest

2.4.3 Boosting

Another powerful and accurate method for classification is boosting which creates consecutive small trees. That is, the method fit tree based on the residuals to ensure the balance between the over-fitting and variance. We also explored the best shrinkage level with cross validation on the train set, and set it as the shrinkage level in our analysis, but in fact it does not guarantee better performance on the test set. Figure 10 shows the importance of the variables based on this model.

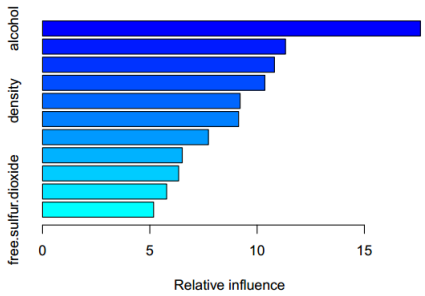


Figure 10: Importance of the variables with boosting

To summarize, the tree based models have competitive performances except the decision tree. Bagging and random forest perform very close and in some cases bagging works better but its average test error is a little bit worse than random forest. Although, decision tree has the worst performance among the methods, it is one of the interpretable models along with KNN.

2.5 Support Vector Machines

Support Vector Machines attempt to separate the data through use of a hyperplane in the same dimension as the data. SVMs with Linear, Polynomial, and Radial kernels were fit to the training set. In a simulation in which new train and test sets were formed each time, the test error was consistently lower for the Radial kernel. Figure 11 only includes Linear and Radial kernels as the Polynomial kernel improved on the test error for a Linear kernel by a small amount relative to the decrease in test error for Radial kernels.

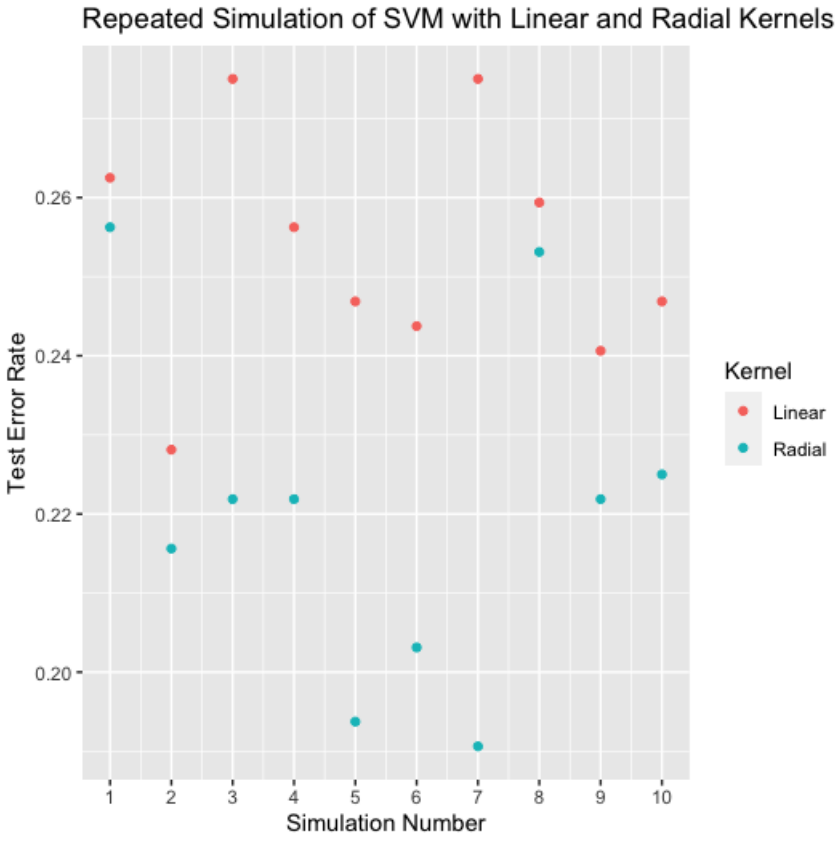


Figure 11: Simulated Test Error rates by Kernel

For each iteration of this simulation (and for the model used to compare to other methods), the parameter associated with penalizing the SVM for training misclassification was fit using grid search cross validation for values: (0.001, 0.01, 0.1, 1, 5, 10, 100) for both Radial and Linear kernels. For Radial kernels, the additional parameter of gamma was fit using grid search cross validation on values: (0.5, 1, 2, 3, 4).

On average, the Radial kernel decreased test error by about 3% compared to an SVM using a Linear kernel on the same data. This suggests that we need a little more complicated surface than a hyperplane to capture the separation of our data into high and low quality. In practice, this may be an unfortunate result since the Radial kernel SVM is a little more costly to fit for data in higher dimensions. If the dataset were much larger with more variables, we would probably have selected the Linear kernel SVM. However, with this dataset, we selected the Radial Kernel SVM to report in the Results section.

The group considered the removal of variables as well. When the pair of kernels were trained on the reduced model, the resulting test errors were significantly higher than that of using all 11 features. Because there

are relatively few variables, the models fit are not likely to suffer due to dimensionality. It was therefore concluded that we should use all variables on the SVM classification.

3 Results

In the previous sections several classification and regression methods were explored. The models and their hyper parameters were tuned so the best performance of each model were obtained. In this section the error rate of different classification methods are compared in multiple simulations. The setup for all the methods are similar in terms of the split of training and testing sets. Additionally the outlier data points were removed for the tree based methods.

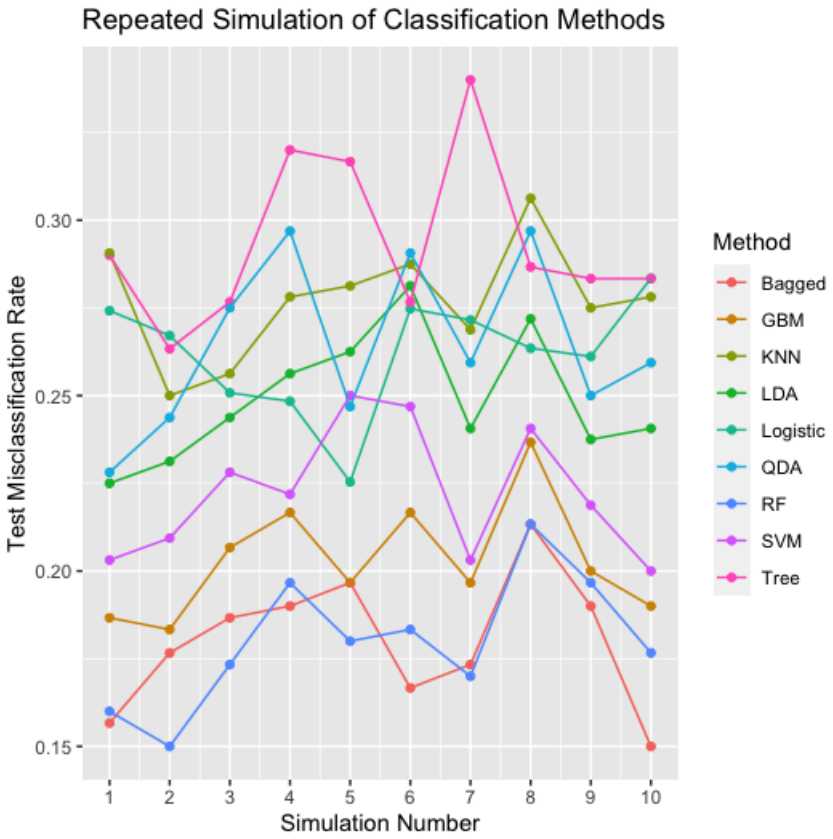


Figure 12: Error rates of methods in 10 simulations

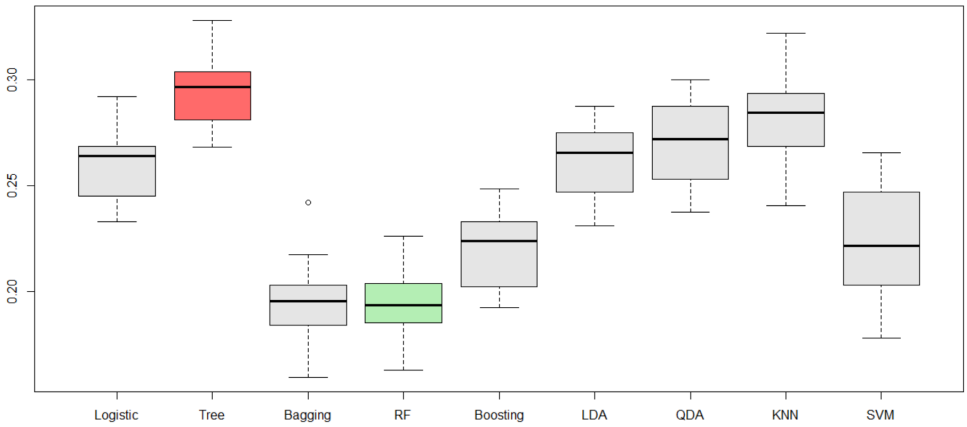


Figure 13: Error rates of methods

As shown in Figure 12 the error rates for each method is plotted in the simulations. It is clear the bagged trees and random forest consistently come first and therefore are the most reliable models for the wine classification. On the other side of the spectrum KNN and decision tree tend to perform poorly. As discussed in section 2.3 the best value for K turned out to be one, which means the model is not learning complex relations and is predicting based on just local data. The same line of argument is true for decision tree.

Model	Error rate	Model	Error rate
Logistic Regression	0.260	KNN	0.285
LDA	0.262	QDA	0.270
Decision Tree	0.295	Bagged Trees	0.196
GBM	0.220	Random Forest	0.194
SVM	0.222		

Table 4: Average classification error rates of the methods

The summary of classification error rates are shown in Table 4. Among all the methods random forest achieves the best accuracy with the error rate of 19.4% and the worst performance belongs to decision tree with error rate of 29.5%. It is worth mentioning that even though SVM is one of the better performers it is the most computationally expensive method.

4 Conclusion

In this project the red wine data was analysed via several methods, namely, logistic regression, LDA, QDA, KNN, tree based methods and SVM. The data was investigated with respect to the histogram of different variables and quality. The wine samples were labeled as "High" quality and "Low" quality forming a two class data set. Additionally, variable selection was performed on the data in order to reduce the dimension of data set. The outlier data points were recognized in order to exclude from tree methods training. All the methods were thoroughly investigated and fine tuned. The models were compared with respect to test set error rates and random forest and bagged trees tended to be more accurate with better performance in terms of training time, while decision tree and KNN are least accurate.

5 Appendix

We used `R` for this study and you may find the related codes as well as some analysis in this section.

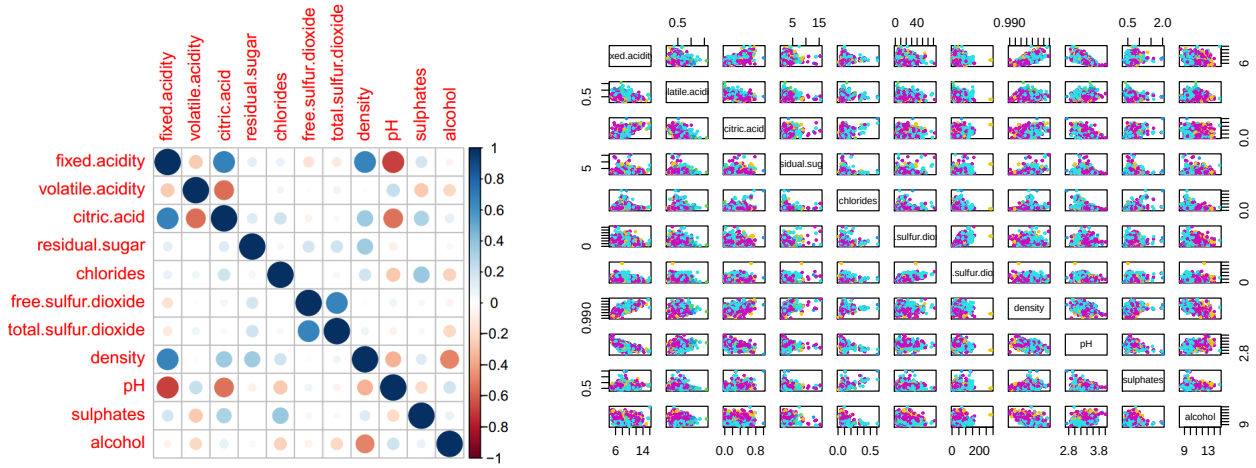
Data Pre-Processing

The first part of our study was data pre-processing which includes: 1) variable selection 2) outlier detection 3) variable transformation. As mentioned we used Lasso for variable selection. We can get a rough idea of important variables by looking at the correlation between all predictors:

Variable Selection

```
## Read Data
set.seed(123)
Red_wine = read.csv("winequality-red.csv")
Missed.data = sum(is.na(Red_wine))
var.name=colnames(Red_wine)
```

```
split = Red_wine$quality
plot(Red_wine[,-c(12)], pch = 19, col = split, cex = 0.4)
corrplot(cor(Red_wine[,-c(12)]),method = "circle")
```



As the left plot shows, `fixed.acidity`, `citric.acid`, `density` and `pH` are highly correlated. Ther is also high correlation among `free.sulfur.dioxide` and `total.sulfur.dioxide`. So we may remove some of them to avoid co-linearity. Pair-plot of the predictors is shown on the right hand side. In this plot, observations are colored according to their quality level. The plot indicates that no single predictor is able to classify wines based on their quality. Now we may use lasso to select useful variables in a more systematic way.

```
# Splitting the Data
train=sample(1:nrow(Red_wine), floor(nrow(Red_wine) * 0.8))
test=(1:nrow(Red_wine))[-train]
Red_wine.train = Red_wine[train,]
Red_wine.test = Red_wine[test,]
```

```r
library(glmnet)
data.matrix = model.matrix(quality ~ .,
                           Red_wine.train)
xtrain = data.matrix[,-1]
ytrain = Red_wine.train$quality
set.seed(123)
cv.lasso <- cv.glmnet(xtrain, ytrain, alpha = 1)
lasso.cv.mod <- glmnet(xtrain, ytrain, alpha = 1,
                       lambda = cv.lasso$lambda.min)
# Display regression coefficients
lasso.reg.coef <- coef(lasso.cv.mod)
lasso.reg.coef
```

```
## 12 x 1 sparse Matrix of class "dgCMatrix"
##                                s0
## (Intercept)           4.457856710
## fixed.acidity         .
## volatile.acidity     -0.977901830
## citric.acid           .
## residual.sugar        .
## chlorides            -1.833001596
## free.sulfur.dioxide   0.001848130
## total.sulfur.dioxide -0.002864311
## density               .
## pH                   -0.483454243
## sulphates             0.778211389
## alcohol               0.293004480
```

Lasso also indicates three of the four highly correlated variables should be removed. It also removes `residual.sugar`. Another interesting observation is that `free.sulfur.dioxide` and `total.sulfur.dioxide` have much smaller coefficients relative to other variables. (Although they are not removed). So we decide to combine these two variables and define a new variable as follows:

$$ratio\_sulfur.dioxide = \frac{free.sulfur.dioxide}{total.sulfur.dioxide}$$

Finally following variables are kept:

"volatile.acidity", "chlorides", "pH", "sulphates" and "alcohol" and created a new variable "ratio_sulfor.dioxide" which is ratio of free and total sulfor dioxide.

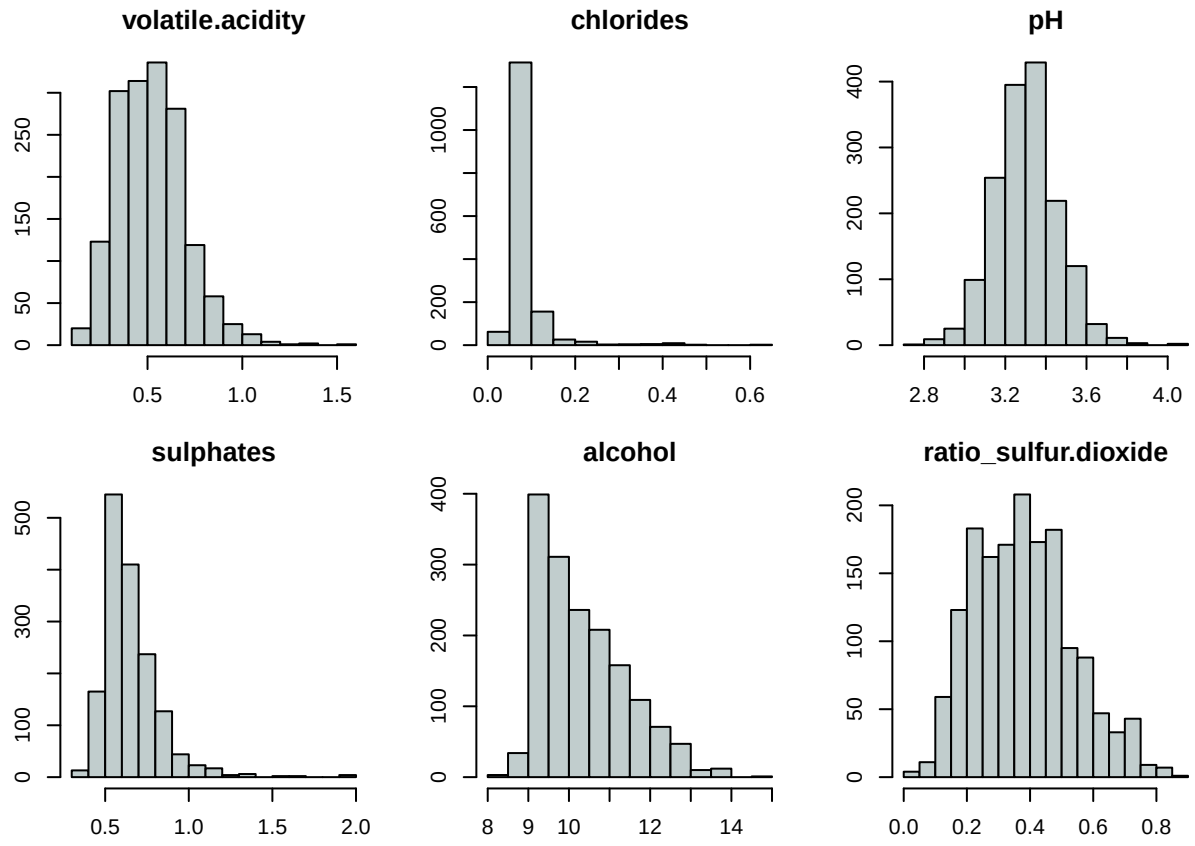
Variables Transformation and Outlier Detection

After removing variables we may check Normality of the remaining ones. It is crucial for LDA and QDA. The easiest way is looking at the histograms of the remaining variables. We may also detect outlier observations by histograms.

```
Red_wine$ratio_sulfur.dioxide <- Red_wine$free.sulfur.dioxide /
                                 Red_wine$total.sulfur.dioxide
Red_wine.Selected <- subset(Red_wine,
                            select = c(volatile.acidity, chlorides,
                                       pH, sulphates, alcohol,
                                       ratio_sulfur.dioxide, quality))
var.name=colnames(Red_wine.Selected)

library(GGally)

colors = c("azure3")
par(mar=c(2.5,2.5,2.5,1.8))
par(mfrow=c(2,3))
for (i in 1:6){
  hist(Red_wine.Selected[,i],
       main = var.name[i], col = colors,
       freq = T, border = 'black')
}
```



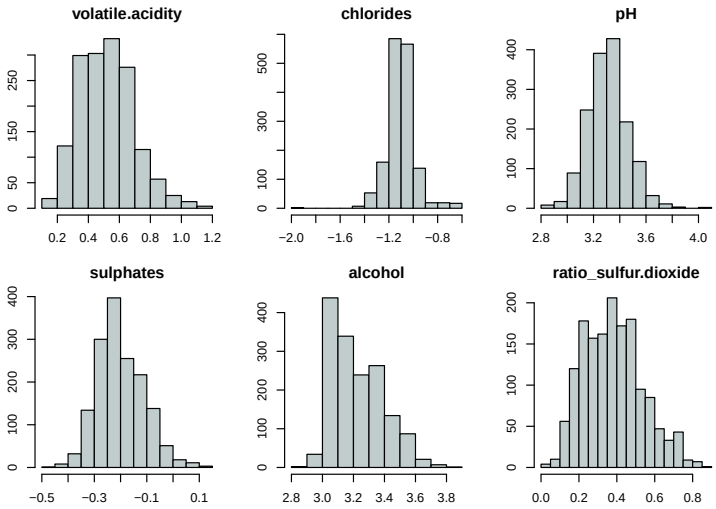
Histograms indicate that `chlorides`, `alcohol` and `sulphates` are far from normal density. log_{10} and square root transformations used to make them close to Normal. Based on the plot, we could say observations related to `volatile.acidity` > 1.2, `chlorides` > 0.25 and `sulphates` > 1.5 are outliers. Data looks like the following plots after removing outliers and transforming variables:

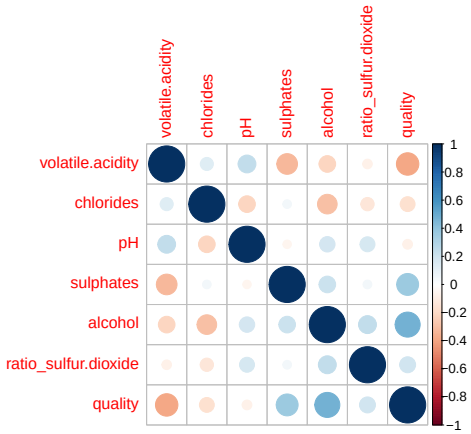
```r
Data <- Red_wine.Selected
Data$chlorides <- log10(Red_wine$chlorides)
Data$sulphates <- log10(Red_wine$sulphates)
Data$alcohol <- sqrt(Red_wine$alcohol)

outliers <- which(Red_wine$volatile.acidity > 1.2)
outliers <- append(outliers, which(Red_wine$chlorides > 0.25))
outliers <- append(outliers, which(Red_wine$sulphates > 1.5))
Data <- Data[-outliers,]
var.name=colnames(Data)
#Red_wine$quality = as.factor(as.character(Red_wine$quality))

## Assessing Normality
library(GGally)
colors = c("azure3")

par(mar=c(2.5,2.5,2.5,1.8))
par(mfrow=c(2,3))
for (i in 1:6){
  hist(Data[,i], main = var.name[i], col = colors, freq = T, border = 'black')
}
corrplot(cor(Data),method = "circle")
```





Models

```
knitr::opts_chunk$set(echo = TRUE, message = F, warning = F)

#Load required libraries
library(ggplot2)
library(e1071)
library(randomForest)
library(MASS)
library(caret)
library(dplyr)
library(class)
library(FNN)
library(tree)
library(gbm)


#KNN classification
make_knn_pred = function(k = 1, train_X, test_X, train_Y, test_Y) {
  pred = knn(train_X, test_X, train_Y, k = k)
  mean(test_Y!=pred)}


#KNN Regression
rmse = function(actual, predicted) {
  sqrt(mean((actual - predicted) ^ 2))
}
make_knn_pred_Reg = function(k = 1, train_X, test_X, train_Y, test_Y) {
  pred = knn.reg(train = train_X,
                       test = test_X,
                       y = train_Y, k = k)$pred
  act = test_Y
  rmse(predicted = pred, actual = act)
}

#Read in data
wine = read.csv("winequality-red.csv", sep=";")

#Split data groups based on Quality
split = factor((wine$quality > 5), labels = c("Bad", "Good"))
mean(split=="Good")
```

```
## [1] 0.5347092
```

```
wineTotal = wine %>%
  mutate(split=split)

train = sample(1:nrow(wineTotal), size=nrow(wineTotal)*0.8, replace = F)

wineTrain = wineTotal[train, ]%>%dplyr::select(-quality)
wineTest = wineTotal[-train, ]%>%dplyr::select(-quality)
```

Logistic Regression

```
outliers = which(wineTotal$volatile.acidity > 1.2)
outliers = append(outliers, which(wineTotal$chlorides > 0.25))
outliers = append(outliers, which(wineTotal$sulphates > 1.5))

logistic = wineTotal[-outliers,]

# Create a new variable and remove variables
logistic = logistic %>%
  mutate(ratio.sulfur.dioxide = free.sulfur.dioxide/total.sulfur.dioxide) %>%
  dplyr::select(-c(fixed.acidity,free.sulfur.dioxide,total.sulfur.dioxide,
                   citric.acid,density,residual.sugar))

train.logistic = logistic[train,]
test.logistic = logistic[-train,]

#Create the logistic regression model
glm.fit.wine = glm(split ~ volatile.acidity+pH+sulphates+I(sulphates^2)+
                           I(sulphates^3)+alcohol+ratio.sulfur.dioxide,
                   data=logistic, family = binomial)

glm.probs = predict(glm.fit.wine,type='response', newdata = test.logistic)

glm.pred = factor((glm.probs > 0.5), labels = c("Bad", "Good"))

mean(glm.pred != test.logistic$split)
```

```
## [1] 0.2875399
```

Tree Method

```
#Logistic regression and tree methods removed the same observations
trees = wineTotal[-outliers, ]%>%dplyr::select(-quality)
treeTrain = trees[train, ]%>%na.omit()
treeTest = trees[-train, ]%>%na.omit()

treeMod= tree(split~., data=treeTrain)

treePred = predict(treeMod, newdata=treeTest, type = "class")

mean(treePred != treeTest$split)
```

```
## [1] 0.3290735
```

Bagged Trees

```
bagMod = randomForest(split ~ ., data=treeTrain, mtry=11, importance=T)
bagPred = predict(bagMod, newdata=treeTest, type="class")
mean(bagPred != treeTest$split)
```

```
## [1] 0.1916933
```

GBM

```
control=trainControl(method="cv", number=5, search="grid")

tunegrid=expand.grid(n.trees=1000,
                     interaction.depth=5,
                     shrinkage=c(0.001,0.005,0.01,0.015,0.02, 0.03, 0.05, 0.1),
                     n.minobsinnode=3)

gb_gridsearch=train(split~., data=treeTrain, method="gbm", metric="Accuracy",
                              tuneGrid=tunegrid, trControl=control, verbose=F)


#Cross Validation over train set which increases the test error
boost.data=gbm(split~., data=treeTrain, distribution="multinomial", n.trees=10000,
               shrinkage=gb_gridsearch$bestTune$shrinkage, interaction.depth=4)

#Prediction
gbmPred = predict.gbm(object = boost.data,
                   newdata = treeTest,
                   n.trees = 500,
                   type = "response")
labels = colnames(gbmPred)[apply(gbmPred, 1, which.max)]

mean(labels != treeTest$split)
```

```
## [1] 0.2204473
```

Random Forest

```
control=trainControl(method="cv", number=5, search="grid")
tunegrid=expand.grid(mtry=c(1:11))

rf_gridsearch=train(split~., data=treeTrain, method="rf", metric="Accuracy",
                              tuneGrid=tunegrid, trControl=control)

rfMod=rf_gridsearch$finalModel

rfPred = predict(rfMod, newdata=treeTest)
mean(rfPred != treeTest$split)
```

```
## [1] 0.1853035
```

LDA-QDA

Multi-level classification is one of the feature of LDA and QDA. Although we split wine quality into two groups, but in this section we considered original 6 class wine quality as well. As results indicate, two class test error is much lower than multi class.

```
#Transformations in an attempt to meet Normality assumptions
LDA = wineTotal
LDA$split = wineTotal$split
LDA$volatile.acidity = wineTotal$volatile.acidity
LDA$chlorides = log10(wineTotal$chlorides)
LDA$pH = wineTotal$pH
LDA$sulphates = log10(wineTotal$sulphates)
LDA$alcohol = sqrt(wineTotal$alcohol)
```

```
LDA$ratio_sulfur.dioxide = wineTotal$free.sulfur.dioxide /
  wineTotal$total.sulfur.dioxide
LDA = subset(LDA, select = c(volatile.acidity, chlorides,
                                pH, sulphates, alcohol,
                                ratio_sulfur.dioxide, split))
## 2 classes
LDA.train = LDA[train,]
LDA.test = LDA[-train,]

#LDA
lda.fit = lda(split~. , data = LDA.train)
lda.pred = predict(lda.fit, LDA.test)$class
mean(lda.pred != LDA.test$split)
```

```
## [1] 0.296875
```

```
#QDA
qda.fit = qda(split~. , data = LDA.train)
qda.pred <- predict(qda.fit, LDA.test)$class
mean(qda.pred != LDA.test$split)
```

```
## [1] 0.31875
```

```
## All classes
LDA$split <- wine$quality
LDA.train = LDA[train,]
LDA.test = LDA[-train,]


#LDA
lda.fit = lda(split~. , data = LDA.train)
lda.pred = predict(lda.fit, LDA.test)$class
mean(lda.pred != LDA.test$split)
```

```
## [1] 0.434375
```

```
#QDA
qda.fit = qda(split~. , data = LDA.train)
qda.pred <- predict(qda.fit, LDA.test)$class
mean(qda.pred != LDA.test$split)
```

```
## [1] 0.471875
```

KNN Classification

```
#Variables to include after selection performed
preds = c(2,5,6,7,9,10,11)

X_trn = wineTrain[ ,preds]
X_tst  = wineTest[ ,preds]

#Grid over which to search for optimal K
k = c(1, 3, 5, 10, 25, 50, 100)
knn_tst_rmse = sapply(k, make_knn_pred,
                      train_X = X_trn,
                      test_X = X_tst,
                      train_Y = wineTrain$split,
                      test_Y = wineTest$split)
```

```
# determine "best" k
best_k = k[which.min(knn_tst_rmse)]

#Build the optimal model
make_knn_pred(k=best_k,
              train_X = X_trn,
              test_X = X_tst,
              train_Y = wineTrain$split,
              test_Y = wineTest$split)
```

```
## [1] 0.290625
```

KNN Regression

```
k = c(1, 3, 5, 10, 25, 50, 100)
knn_tst_rmse = sapply(k, make_knn_pred_Reg,
                      train_X = X_trn,
                      test_X = X_tst,
                      train_Y = wineTotal[train, 12],
                      test_Y = wineTotal[-train, 12])

# determine "best" k
best_k = k[which.min(knn_tst_rmse)]

#Build the optimal model
make_knn_pred_Reg(k=best_k,
                  train_X = X_trn,
                  test_X = X_tst,
                  train_Y = wineTotal[train, 12],
                  test_Y = wineTotal[-train, 12])
```

```
## [1] 0.7352083
```

SVM-Radial

```
radialSVM = tune(svm, split ~., data=wineTrain, kernel="radial", scale=T,
                 ranges=list(
                   cost=c(0.001, 0.01, 0.1, 1),
                   gamma=c(0.5, 1, 2, 3, 4)
                 ))
radialSVMPred = predict(radialSVM$best.model, newdata=wineTest)
mean(radialSVMPred != wineTest$split)
```

```
## [1] 0.1875
```

Simulation

In order to get a more reliable test error, we run our codes 10 times with different test sets and report the average error.