PART 1

Table of Contents

1.1 INTRODUCTION

Wine is considered to be one of the most popular drinks in the world and is among the favorite drinks of the people of Britain. Wine quality, as the standard measurement, plays an important role in the industry. It affects both the production and consumption of wine. The quality of wine is determined by its various chemical properties which not only affects their taste but also the aroma. There are 5 main types of wines- red wine, white wine, rose wine, dessert or sweet wine and sparkling wine. Each of these types of wines have different chemical compositions which affects their quality and taste.

In this paper we have used "Wine Quality" dataset obtained from UCL machine learning repository.

The **objective** of our study is to classify the quality of white wine as average or good wine based on its 11 chemical properties. This paper is divided into three parts: data preparation, model training and tuning, and model selection.

1.2 DATA PREPARATION

1.2.1 Data Description

The dataset has 4898 observations and consists of 11 numeric variables - fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates and alcohol and 1 categorical variable - wine quality, which takes discrete values from 0-10. However, the minimum and maximum value of wine quality are 3 and 9 respectively. The data points follow normal distribution as shown in **figure1**.

The wine quality has been divided into two classes (based on its distribution) after cleaning the data: wine quality>7 is considered to be "good wine" and wine quality≤7 is considered to be "average wine".

1.2.2 Data cleaning

The data cleaning consists of 3 steps:

Data cleaning steps	Check missing values using is.a() function	Check duplicates using duplicates() function	Check and remove outliers using box plot and Cook's distance
Observation	No missing value	Removed 937 duplicate values	Removed 204 outliers

The boxplot **(figure 2)** represents that residual sugar, free sulfur dioxide, and total sulfur dioxide have long tails. It also depicts that the difference between the third quartile and the maximum value of these features is very large. This difference explains the presence of outliers in our dataset. To remove outliers, we have used Cook's distance. Cook's distance is used to find influential outliers given a set of predictor variables[1].

$$Di = \frac{\sum(\hat{Y}_j - Y_{j(i)})^2}{(p+1)\hat{\sigma}^2}$$

In the graph **(figure 3)** obtained from Cook's distance, the red line is plotted at the point 4*mean. The observations greater than 4 times the mean are considered to be influential outliers.

1.2.3 Correlation

Correlation measures the degree to which two variables are related to each other. **Figure 4** depicts the correlation between all the variables. Observation includes-(a) negative correlation between pH and Alcohol. (b) positive correlation between density and residual sugar. (c) high positive correlation between quality and alcohol.

[1] Prabhankaran 2019

1.2.4 Training and Testing set

Before we split the data, we need to scale the data first, by using the **scale()** function. Then the dataset is divided into two parts - training and testing named as **dat_train** and **dat_test**. The training set contains 80% of the data, while the test set contains 20% of the data. It is observed that the data is unbalanced because the training dataset contains 104 positive observations (good wine) and 2923 negative observations (average wine). To fix this problem, smote algorithm is used in the next step.

1.2.5 SMOTE algorithm for unbalanced classification problems

SMOTE algorithm is used to fix the problem of unbalanced classification problems. The general idea of this method is to artificially generate new examples of the minority class using the nearest neighbors of these cases. Furthermore, the majority class examples are also random sampled, leading to a more balanced dataset[2]. We have used smote algorithm on our training dataset. This gives 3016 positive observations and 2912 negative observations, which solves the unbalanced classification problem.

1.3 MODEL CLASSIFICATION

In this section, decision tree, bagging and random forest models are used on the dataset. The tree model has many advantages because it is easy to interpret and understand by looking at the tree structure.

1.3.1 Decision Tree: Based on the training dataset, a classification tree model is built, in which **wine quality** is the response, and the rest 11 features are the explanatory variables. The **tree()** function is used to fit the classification tree model. The tree has 11 nodes and the variables which are truly used in constructing the tree are alcohol, volatile acidity and free sulfur dioxide. To prune the tree, it is found that the tree size should be equal to 7, which has minimum estimated misclassification error rate 0.1987(**figure 5** and **figure 7) .**

1.3.2 Random Forest: The test error rate for the best-size pruned tree is 0.1987 which is still not small enough. Therefore, random forests and bagging methods are applied, which construct many classifiers and combine these different independent base classifiers to get the final result. In each classifier, several features are selected to fit the model. **RandomForest()** function is used to perform the random forest model. Here, we set the parameter mtry=sqrt(p dimensions). This model has only 6.73% error, which means we can predict with 93.26% accuracy. The results indicate that the alcohol is the most important variable in terms of Model Accuracy and Gini index **(figure 9)**. **Bagging Random Forest:** It is a special case of a random forest that uses all the features to train the classifiers. Here it is observed that the accuracy is 92.6% which is a bit smaller than the random forest model. The alcohol is still the most important variable **(figure 8)**.

1.3.3 XGBoost and Gradient Boosting: XGBoosting has recently become a very popular machine learning techniques. It is an execution of gradient boosting decision tree algorithm[3]. In the randomForest, each tree are independent, but in boosting model, new tree depend on previous tree. The expand.grid() function is used to do the parameter selection which includes the **nrounds, max_depth** and **colsample_bytree,** and it is found that the parameters should be set as {200,15,0.6} to get the best model. XGBoost reduced the error rate to 5.28% with the accuracy equal to 94.71% which is higher than random forest.

[2] Performance Estimation documentation 2019
[3] Reinstein 2019

1.3.4 Support Vector Machine and Other Models: After trying the tree models, we tried some other models like Support Vector Machine, which is also a good prediction tool for classification and regression. It uses machine learning theory to maximize predictive accuracy and avoids overfitting of data. The model gives the accuracy of 84.8% which is less than Random Forest and XGBoost model. We also tried the logistic regression model and K nearest neighbor model to fit the data. In the KNN model, we assign the values from 5 to 20 to the parameter k and select k=5 because it gives the highest accuracy. The two models return a lower accuracy value compared to the tree models.

1.4 MODEL SELECTION

Accuracy, error rate, precision, recall and AUROC curve play a very important role in determining the best model for dataset. Among the above measurements, accuracy rate is the most important factor for finding the best model. However, other measurements have been used to see the performance of different models. From the **figure 11**, it is observed that ROC curves of bagging and random forests are very close. So, we calculate the area under the curve for these two models. AUC for bagging is 0.7574 and AUC for random forests is 0.7998. In order to calculate accuracy, error, precision and recall, a confusion matrix of predicted and true values is constructed. All these measurements for every model are presented in the table below.

	Error Rate	Accuracy	Precision	Recall
XG Boosting	0.0528	0.9471	0.9727	0.9727
Random Forest	0.0673	0.9326	0.9762	0.9536
Logistic Regression	0.2589	0.7410	0.9872	0.7421
KNN	0.1677	0.8322	0.9764	0.8472
Decision Tree	0.1987	0.7564	0.9841	0.7613
Bagging	0.0739	0.9260	0.9762	0.9536
SVM	0.1519	0.8480	0.9768	0.8635

From the above table, it is observed that logistic regression and decision tree can be classified as worst models for our dataset since they have extremely low accuracy rate and recall. On the other hand, XGBoost, Randomforest and Bagging Randomforest have a very high accuracy and recall which is greater than 90%. Among these three models, XGBoost can be considered as the best model for our dataset as it has the highest accuracy rate of 94.71%. It also has a high precision rate equal to 97.27% and recall equal to 97.27%.

1.5 CONCLUSION

In this paper, the study shows that XGBoost is the best model for classifying the quality of wine as good or average, since it gives the highest accuracy rate (94.71%) and lowest error rate. XGBoost has become a widely used and very popular machine earning technique among Data Scientists in industry. It has proven to push the limits of computing power for boosted trees algorithms as it was built and developed for the sole purpose of model performance and computational speed.[4] Our study indicates that XGBoost outperforms all other models that have been in the classification of dataset and it is highly scalable/ parallelizable and fast to implement. For future studies, a better result can be obtained by collecting a more balanced data.

[4] Reinstein 2019

1.6 REFERENCE

[1] Reinstein, Ilan. 2019. "Xgboost, A Top Machine Learning Method On Kaggle, Explained - Kdnuggets". *Kdnuggets*.https://www.kdnuggets.com/2017/10/xgboost-top-machine-learning-method-kaggle-explained.html.

[2] Documentation, performance Estimation documentation, R docs, and Run browser. 2019. "Smote: SMOTE Algorithm For Unbalanced Classification Problems In Performanceestimation: An Infra-Structure For Performance Estimation Of Predictive Models". *Rdrr.Io*. https://rdrr.io/cran/performanceEstimation/man/smote.html.

[3] Prabhakaran, Selva. 2019. "Outlier Detection And Treatment With R". *R-Bloggers*. https://www.r-bloggers.com/outlier-detection-and-treatment-with-r/.

1.7 APPENDIX

Graphs for our data

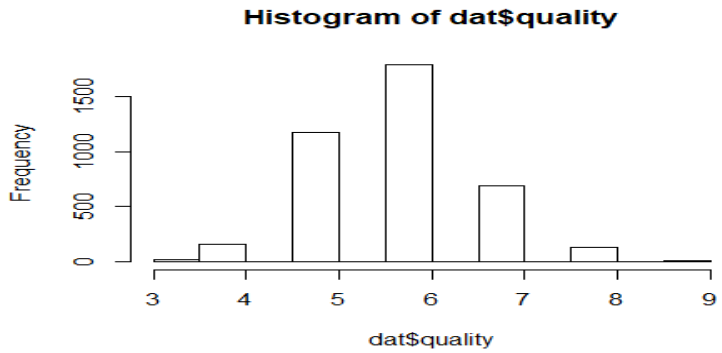

Figure 1

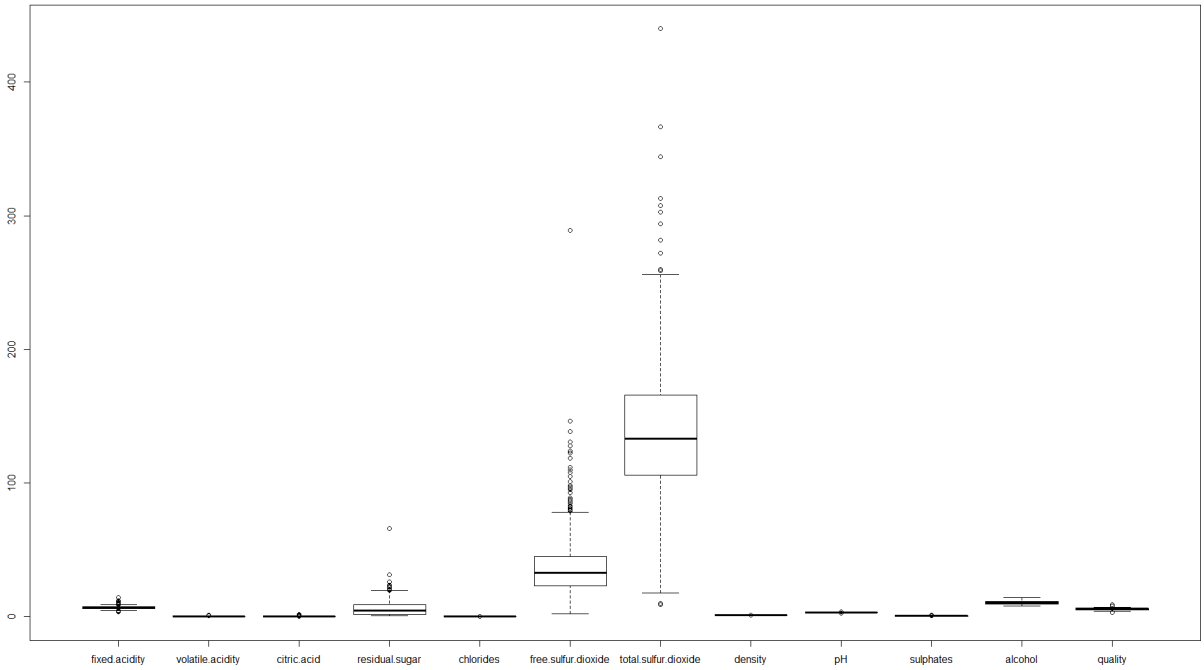

Figure 2

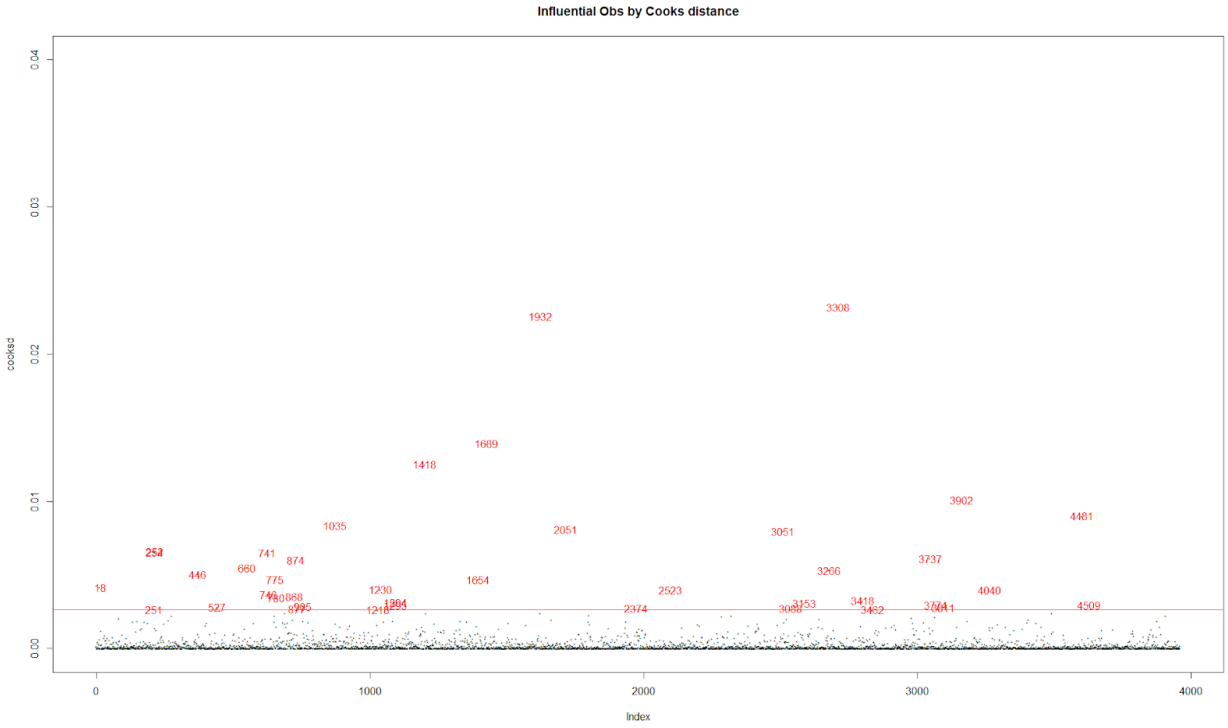

Figure 3

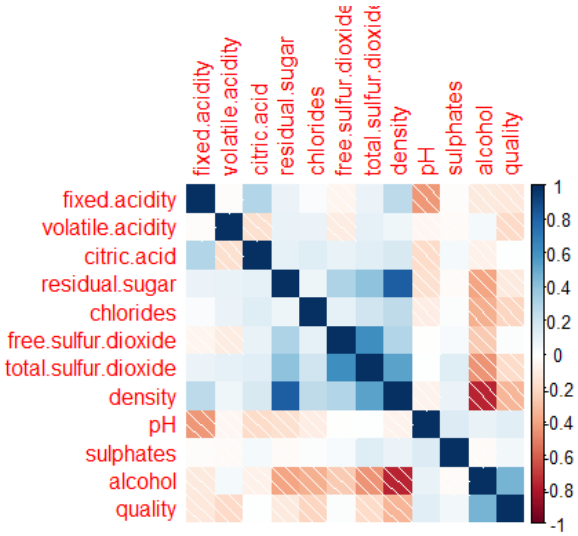

Figure 4

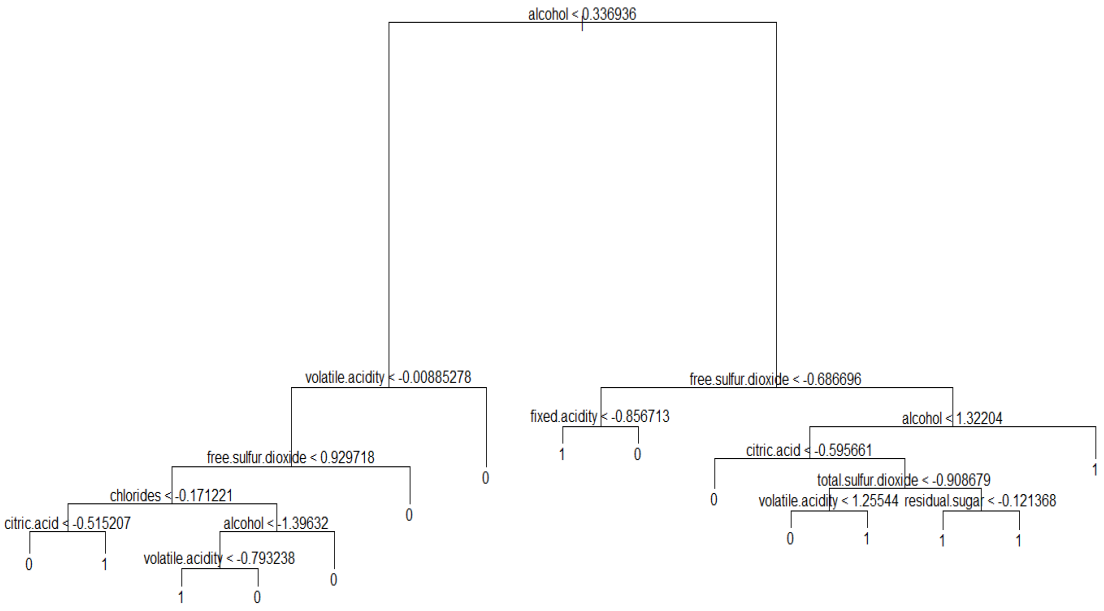

Figure 5

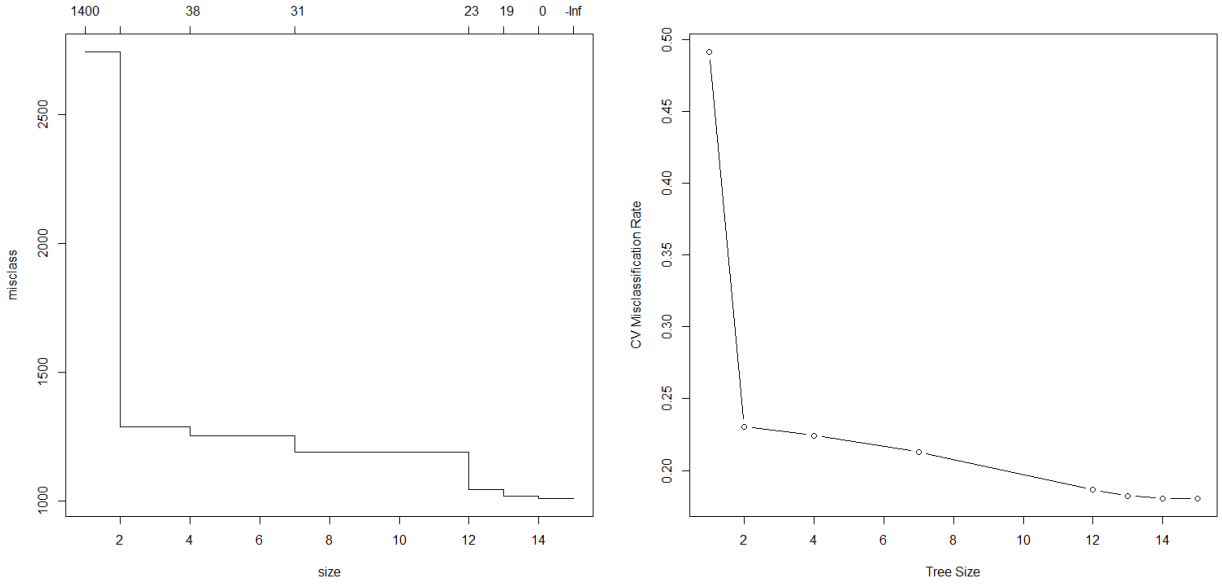

Figure 6

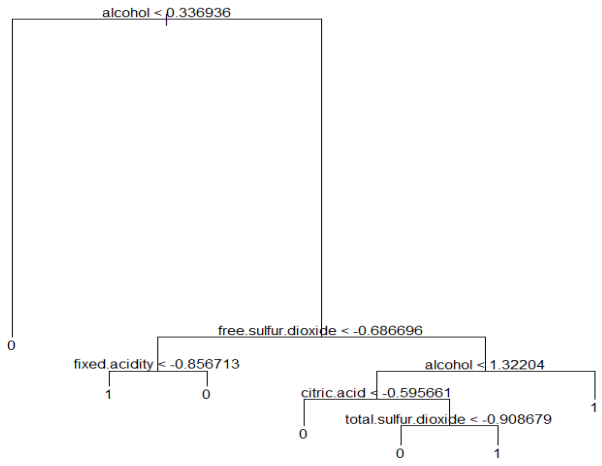

Figure 7

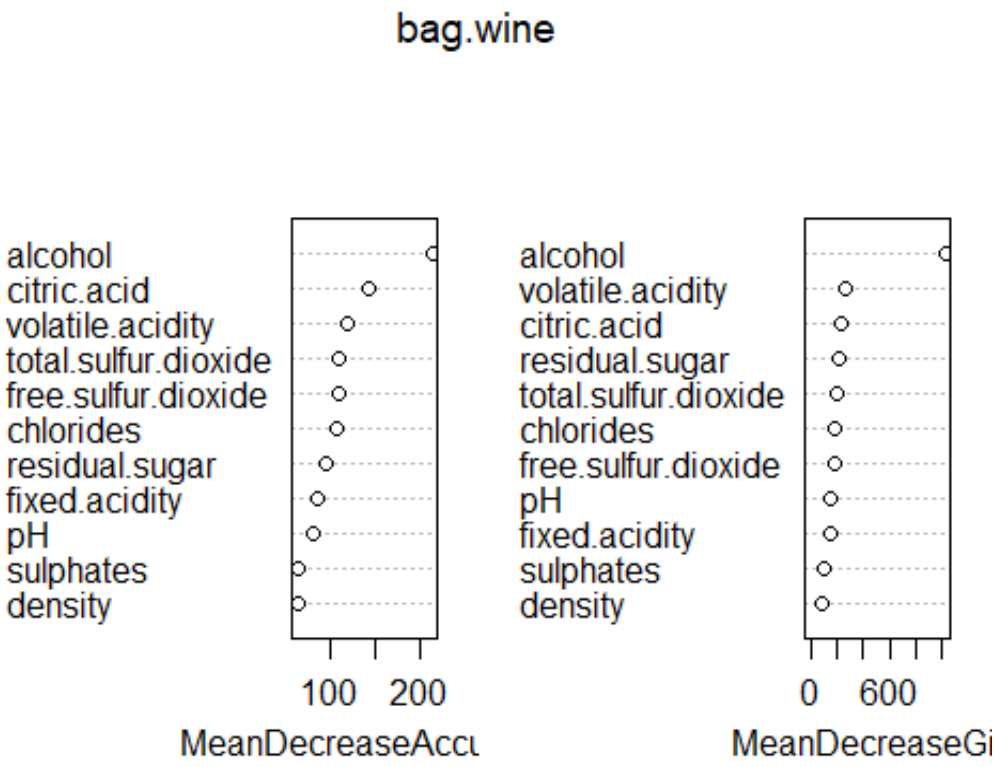

Figure 8

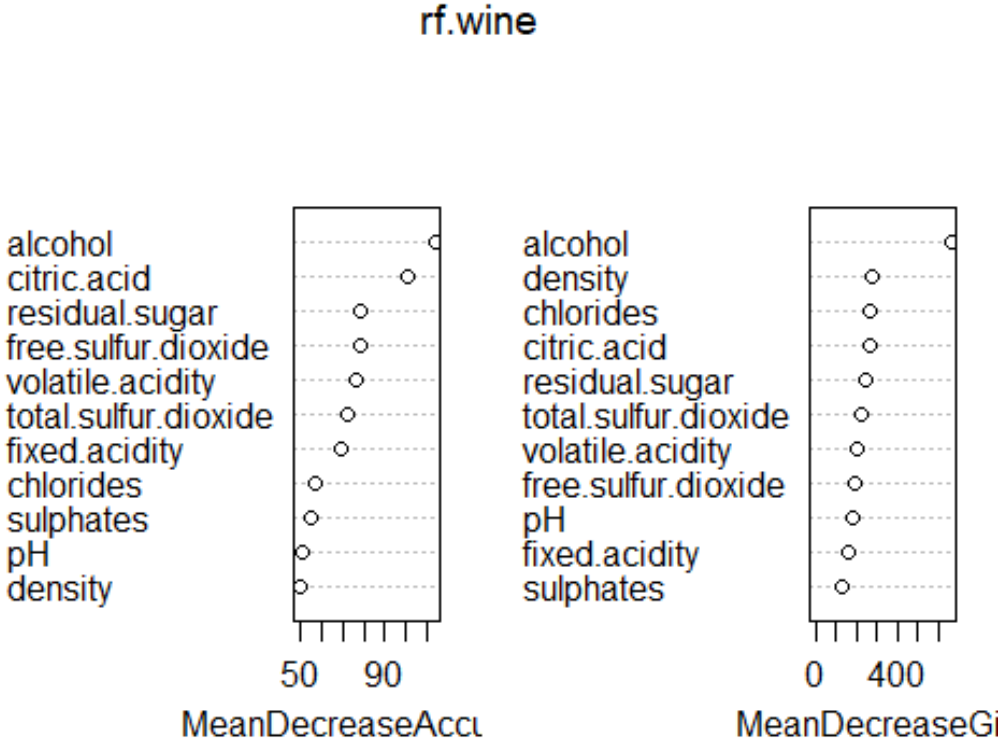

Figure 9

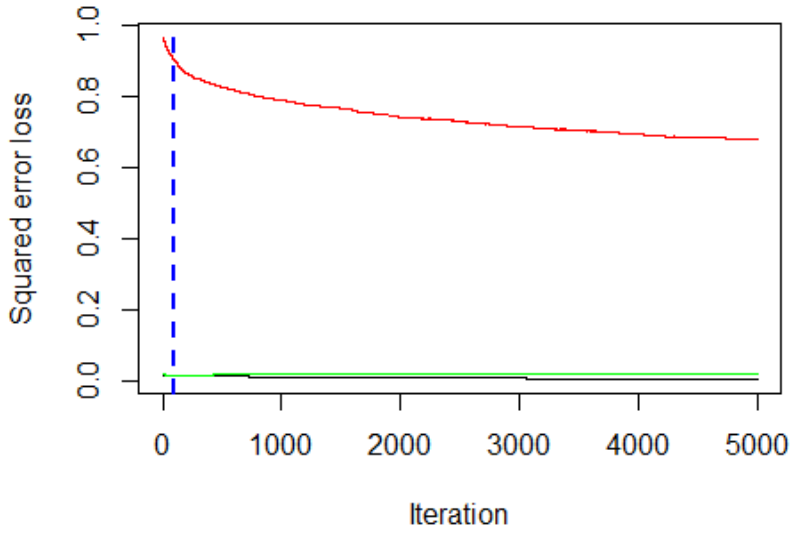

Figure 10

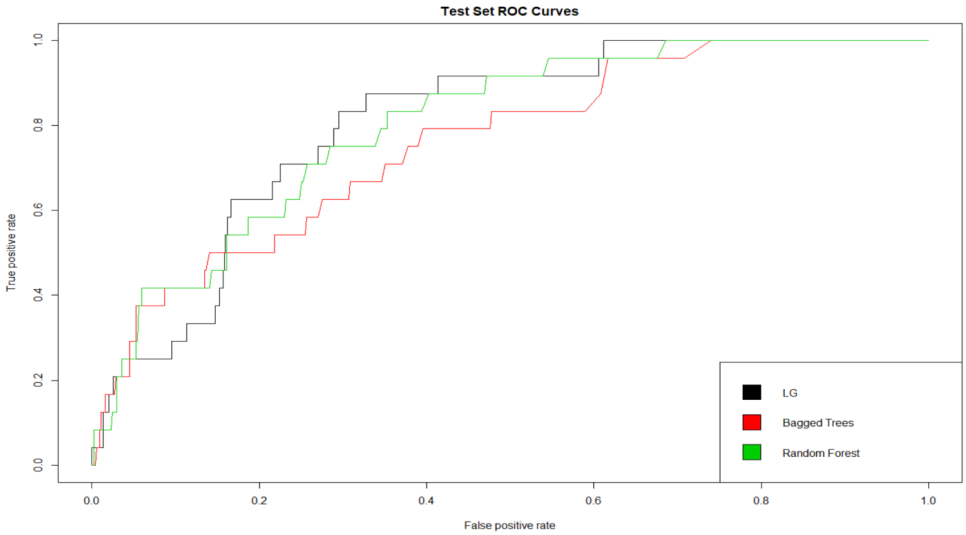

Figure 11

Code for model selection

```
library(tidyverse)
library(tree)
library(randomForest)
library(gbm)
library(ROCR)
library(e1071)
library(imager)
library(dplyr)
library(readr)
library(ISLR)
library(ggplot2)
library(reshape2)
library(plyr)
library(class)
library(caret)
```

#Data Summary

```
#import data
data<-read.csv(file ="C:/Users/13645/Desktop/LSE/ST443/group project/winequality-
white.csv",sep = ";")

#1. see class of each column:11 numeric & 1 interger
lapply(data, class)

#2. summary the data
summary(data)

#3. see missing values in each column; Then sum them.
sapply(data, function(x) sum(is.na(x)))
sum(sapply(data, function(x) sum(is.na(x))))

#4. find correlations of each pair
cor(data)
```

```r
#duplicate
dat=data[!duplicated(data),]
```

##Data Visualizing

```r
#plot to see the relationship between each pairs; and count them;
hist(dat$quality)
table(dat$quality)

#find correlations of each pair;
round(cor(dat),3)
library(corrplot)
corrplot(cor(dat), method = "shade")
```

Outlier

We use cooksd to calculate the outliers. (http://r-statistics.co/Outlier-Treatment-With-R.html)

```r
boxplot(dat)

outliers <- lm(quality ~ ., data=dat)
cooksd <- cooks.distance(outliers)

plot(cooksd, pch="*", cex=0.5, main="Influential Obs by Cooks distance",ylim=c(0,0.04))  #
plot cook's distance
abline(h = 4*mean(cooksd, na.rm=T), col="red")  # add cutoff line
text(x=1:length(cooksd)+1, y=cooksd, labels=ifelse(cooksd>4*mean(cooksd,
na.rm=T),names(cooksd),""), col="red")  # add labels

# Outliers' index
names(cooksd)[(cooksd > (4/nrow(dat)))]
# record outliers
influential <- as.numeric(names(cooksd)[(cooksd > (4/nrow(dat)))])
```

add label

we decided quality>7 ——"good", quality<=7 ——"normal"

```r
library(dplyr)
# delect the outliers
dat2 = dat[-influential, ]

#add label
dat2 = dat2 %>%
  mutate(winequality=as.factor(ifelse(quality<=7, 0, 1)))
dat2=dat2[,-12]

summary(dat2$winequality)
```

Training set and Test set

```r
# change the dat_new into data.frame and mutate winequality.

#data=data.frame(dat_new)%>%
#  mutate(winequality=dat2$winequality)

data=data.frame(scale(dat2[-12], center=T,scale=T))%>%
  mutate(winequality=dat2$winequality)
#train & test 8:2
set.seed(2)
train.indices = sample(1:nrow(data),nrow(data)*0.8)
dat_train=data[train.indices,]
```

```
dat_test=data[-train.indices,]
#Applying Standard scaling to get optimized result

dim(dat_train)
dim(dat_test)
table(dat_train$winequality)
library(performanceEstimation)

## A small example with a data set created artificially from the IRIS
## data
#data(iris)
#data <- iris[, c(1, 2, 5)]
#data$Species <- factor(ifelse(data$Species == "setosa","rare","common"))
## checking the class distribution of this artificial data set
#table(data$Species)

## now using SMOTE to create a more "balanced problem"
newData <- smote(winequality ~ ., dat_train, perc.over = 28,perc.under=1)
table(newData$winequality)
#table(dat_test$winequality)
## End(Not run)
```

logistic regression

```
glm_fit = glm(winequality ~.,
              data = newData,
              family = binomial)
summary(glm_fit)
glm_probs = predict(glm_fit, dat_test[-12],type = "response")

contrasts(dat_train$winequality)
glm_pred = rep(1, 757)
glm_pred[glm_probs < .5] = 0

#error rate
mean(glm_pred!=dat_test$winequality)
#accuary
tst_tab <- table(predicted = glm_pred, actual = dat_test$winequality)
sum(diag(tst_tab))/length(dat_test$winequality)
precision(tst_tab)
recall(tst_tab)
```

##KNN

```
knn_pred <- knn(newData[,-12], dat_test[,-12], cl=newData$winequality, k=5)

#error rate
table(knn_pred,dat_test$winequality)
mean(knn_pred!=dat_test$winequality)
#accuary
sum(dat_test$winequality == knn_pred)/NROW(dat_test$winequality)



trctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
set.seed(3333)
knn_fit <- train(winequality ~., data = newData, method = "knn",
trControl=trctrl,
preProcess = c("center", "scale"),
tuneLength = 10)
knn_fit
```

```
knn_pred <- predict(knn_fit, newdata = dat_test)

confusionMatrix(knn_pred, dat_test$winequality)

#error rate
mean(knn_pred!=dat_test$winequality)
#accuary
tst_tab <- table(predicted = knn_pred, actual = dat_test$winequality)
sum(diag(tst_tab))/length(dat_test$winequality)
precision(tst_tab)
recall(tst_tab)
```

##cv-decision tree

```
library(tree)
tree.wine <- tree(winequality ~ . , data=newData)
plot(tree.wine)
text(tree.wine,pretty=0)

set.seed(3)
cv.wine <-cv.tree(tree.wine, FUN = prune.misclass)
min_idx = which.min(cv.wine$dev)
min_idx
cv.wine$size[min_idx]

par(mfrow = c(1, 2))
# default plot
plot(cv.wine)
# better plot
plot(cv.wine$size, cv.wine$dev / nrow(newData), type = "b",
    xlab = "Tree Size", ylab = "CV Misclassification Rate")

prune.wine = prune.misclass(tree.wine, best = 7)
summary(prune.wine)
plot(prune.wine)
text(prune.wine,pretty=0)

tree_pred = predict(prune.wine, dat_test, type = "class")
table(predicted = tree_pred, actual = dat_test$winequality)
library(MLmetrics)
Accuracy(y_pred=tree_pred,y_true=dat_test$winequality)
Precision(y_pred=tree_pred,y_true=dat_test$winequality)
Recall(y_pred=tree_pred,y_true=dat_test$winequality)
```

Bagging and Random Forest

Recall that bagging is simply a special case of a random forest with m=p, here we use `mtry=8`.

```
library(MASS)
bag.wine <- randomForest(winequality~., data=newData, mtry=11, importance=TRUE)
bag.wine
yhat.bag <-predict(bag.wine, newdata=dat_test)
bag.probs <-predict(bag.wine, newdata=dat_test,type="prob")
#error rate
mean(yhat.bag!=dat_test$winequality)
#accuary
tst_tab <- table(predicted = yhat.bag, actual = dat_test$winequality)
sum(diag(tst_tab))/length(dat_test$winequality)

cm_bag <- confusionMatrix(yhat.bag, dat_test$winequality)
```

```
cm_bag$overall[1]
precision(tst_tab)
recall(tst_tab)
```

We can view the importance of each variable:

```
importance(bag.wine)
varImpPlot(bag.wine)
```

The plot indicates `lstat` and `rm` are two most important variables.

Growing a random forest is exactly the same way. By default, `randomForest()` uses p/3 variables when building a random forest of regression trees and p variables when building a random forest of classiciation trees. We use `mtry=6` here:

```
set.seed(1)
rf.wine <-randomForest(winequality~., data=newData, mtry=, importance=TRUE, n.tree = 5000)
yhat.rf <-predict(rf.wine, newdata=dat_test)
rf.probs<-predict(rf.wine, newdata=dat_test,type="prob")
#error rate
mean(yhat.rf!=dat_test$winequality)
#accuary
tst_tab <- table(predicted = yhat.rf, actual = dat_test$winequality)
sum(diag(tst_tab))/length(dat_test$winequality)
#recall
precision(tst_tab)
recall(tst_tab)
importance(rf.wine)
varImpPlot(rf.wine)
```

##SVM

```
learn_svm <- svm(winequality~., data=newData)
pre_svm <- predict(learn_svm, dat_test)
cm_svm <- confusionMatrix(pre_svm, dat_test$winequality)
cm_svm

cm_svm$overall[1]
#error rate
mean(pre_svm!=dat_test$winequality)


tst_tab <- table(predicted = pre_svm, actual = dat_test$winequality)
sum(diag(tst_tab))/length(dat_test$winequality)
precision(tst_tab)
recall(tst_tab)
library(gbm)
test_gbm <- gbm(winequality~., data=newData, distribution="gaussian",n.trees = 5000,
                shrinkage = 0.01, interaction.depth = 4, bag.fraction=0.5,
train.fraction=0.5,n.minobsinnode=10,cv.folds=3,keep.data=TRUE,verbose=FALSE,n.cores=1)
best.iter <- gbm.perf(test_gbm, method="cv",plot.it=TRUE)
fitControl = trainControl(method="cv", number=5, returnResamp="all")
learn_gbm = train(winequality~., data=dat_train, method="gbm", distribution="bernoulli",
trControl=fitControl, verbose=F, tuneGrid=data.frame(.n.trees=best.iter, .shrinkage=0.01,
.interaction.depth=1, .n.minobsinnode=1))
pre_gbm <- predict(learn_gbm, dat_test[,-12])
cm_gbm <- confusionMatrix(pre_gbm, dat_test$winequality)
cm_gbm
#error rate
mean(pre_gbm!=dat_test$winequality)
```

```
tst_tab <- table(predicted = pre_gbm, actual = dat_test$winequality)
sum(diag(tst_tab))/length(dat_test$winequality)
precision(tst_tab)
recall(tst_tab)
```

##xgboost

```
library(data.table)
library(mlr)
library(xgboost)
library(caret)
# set caret tuning configuration: 5-Fold CV, 3 repetitions
X_train = xgb.DMatrix(as.matrix(newData[-12]))
y_train = newData$winequality
X_test = xgb.DMatrix(as.matrix(dat_test[-12]))
y_test = dat_test$winequality
xgb_trcontrol = trainControl(
  method = "cv",
  number = 5,
  allowParallel = TRUE,
  verboseIter = FALSE,
  returnData = FALSE
)
xgbGrid <- expand.grid(nrounds = c(100,200),  # this is n_estimators in the python code above
                       max_depth = c(10, 15, 20, 25),
                       colsample_bytree = seq(0.5, 0.9, length.out = 5),
                       ## The values below are default values in the sklearn-api.
                       eta = 0.1,
                       gamma=0,
                       min_child_weight = 1,
                       subsample = 1
                       )
# Training sample

set.seed(0)
xgb_model = train(
  X_train, y_train,
  trControl = xgb_trcontrol,
  tuneGrid = xgbGrid,
  method = "xgbTree"
)

xgb_model$bestTune
#> xgb_model$bestTune
#   nrounds max_depth eta gamma colsample_bytree min_child_weight subsample
#14     200   15 0.1 0              0.6 1   1
xg_pred = predict(xgb_model,X_test)
Accuracy(y_pred=xg_pred,y_true=y_test)

#error rate
mean(xg_pred!=dat_test$winequality)


tst_tab <- table(predicted = xg_pred, actual = dat_test$winequality)
sum(diag(tst_tab))/length(dat_test$winequality)
precision(tst_tab)
recall(tst_tab)
##AUROC
# List of predictions
preds_list <- list(glm_probs,bag.probs[,-1],rf.probs[,-1])
#, knn_pred, yhat.bag, yhat.boost,yhat.rf,pre_gbm,pre_svm
```

```r
# List of actual values (same for all)
m <- length(preds_list)
actuals_list <- rep(list(dat_test$winequality), m)

# Plot the ROC curves
pred <- prediction(preds_list, actuals_list)
rocs <- performance(pred, "tpr", "fpr")
plot(rocs, col = as.list(1:m), main = "Test Set ROC Curves")
legend(x = "bottomright",
       legend = c("LG", "Bagged Trees", "Random Forest"),
       fill = 1:m)
library(pROC)
ROC1 <-roc(dat_test$winequality,glm_probs)
ROC2 <-roc(dat_test$winequality,bag.probs[,-1])
ROC3 <-roc(dat_test$winequality,rf.probs[,-1])
auc(ROC1)
auc(ROC2)
auc(ROC3)
```