STSCI 5740 Final Project Report - Group 4

Huimin Wang (hw447), Xingyu Zhang (xz385), Yuxiao Feng (yf254), Zhizhou Yin (zy222)

Introduction

From UCI Machine Learning website, we have the wine-quality-red-and-white dataset. Our goal is to predict the wine qualities. We used the following methods: simple linear regression, binomial logistic regression, multinomial logistic regression, SVM, Decision Tree, XGBoost, Random Forest. We found that non-linear classification method lead to a higher prediction accuracy, with the highest validation accuracy of 69.20% produced by XGBoost.

1 Data Preprocessing

1.1 Data Analysis

Table 1: Quality Distribution

Quality	3	4	5	6	7	8	9
Quantity	30	216	2138	2836	1079	193	5

According to Tabel 1, we see that the distribution of quality is not balanced, with most of them clustered at 5,6,7, few in 3,4,8, and fewest in 9. The imbalance of data would impact prediction accuracy. In the discussion part of this report, we address the imbalance issue.

1.2 Data Preprocessing and Feature Engineering

We do one-hot encoding for the categorical “type”, mapping data to values 0 and 1 (white, red). We then plot the correlation matrix. As we can see from the correlation matrix in Figure 1:

(1) “alcohol” has the highest positive correlation (0.444) with quality

(2) “density” has the highest negative correlation (-0.306) with quality.

Intuitively, these two might be important predictors for predicting the quality of wine.

We can also see that:

(1) “free.sulfur.dioxide” has a high positive correlation (0.721) with “total.sulfur.dioxide”.

(2) “free.sulfur.dioxide” has a high negative correlation (-0.7) with “typered”.

(3) “total.sulfur.dioxide” has a high negative correlation(-0.687) with “alcohol”.

We could remove variables with high correlation since they represent similar information. Besides, if we don’t do so, multicollinearity would lead to inaccurate results related to standard error and confidence interval. Consider the above information, from (3), we remove “total.sulfur.dioxide” since “alcohol” has the highest positive correlation (0.444) with quality; from (1), we could keep “free.sulfur.dioxide” since we remove “total.sulfur.dioxide”; from (2), we remove “typered” since we keep“free.sulfur.dioxide”.

To conclude, we remove predictors “typered” and “total.sulfur.dioxide” and keep the rest 10 predictors. All the models analyzed below use 10 predictors to predict quality.

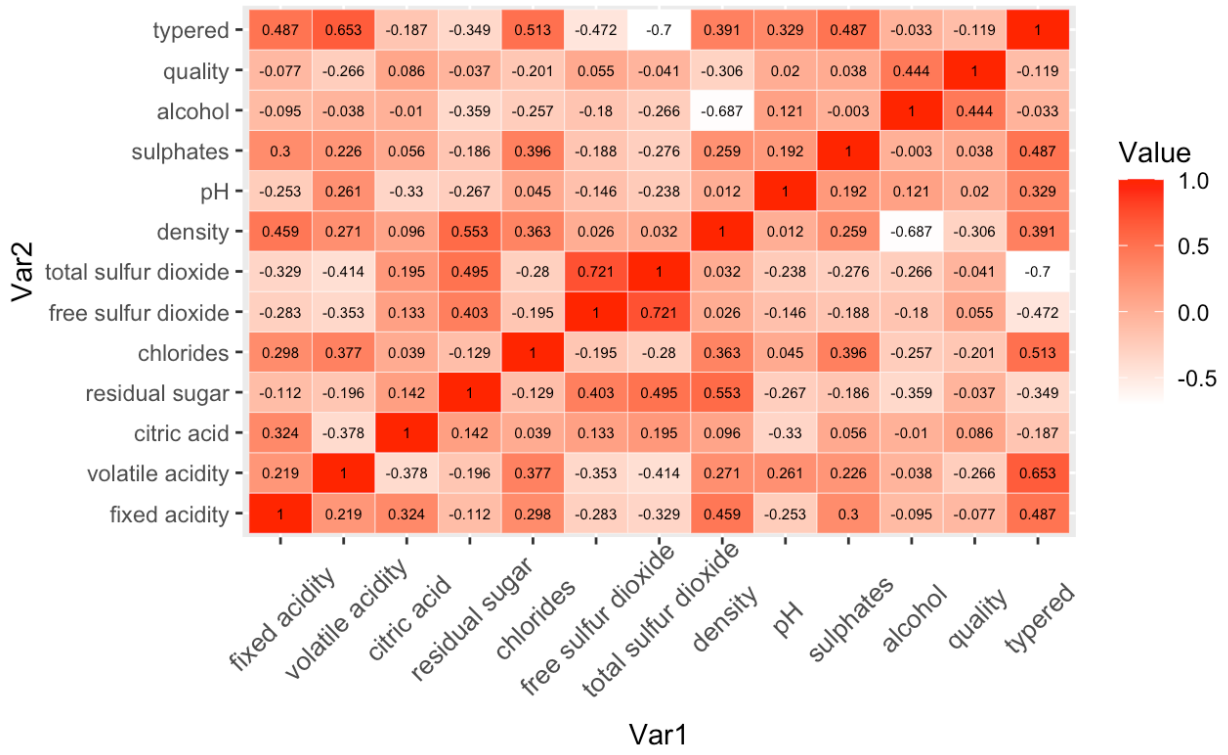


Figure 1: Correlation Matrix

2. Modeling

We conduct 10-fold cv to train each model and use the validation accuracy to compare the result.

2.1 Regression Analysis: Linear Regression

Even though this problem is a classification problem since the response variable takes discrete values, these values have order meanings : quality from 3 to 9 represents quality from low to high. We can model this problem as a regression one, and transform the result into a classification problem. To be more specific, we transform the continuous predicted result into discrete values by first rounding the value, and then treat all round values < 3 as 3, and round values > 9 as 9. For example, for predicted values 1.4, 3.6, 6.5, 10.7 would be transformed to 3, 4, 7, 9. Based on transformed predicted values, we calculate the validation accuracy.
We try the simplest regression model, linear regression. The regression result is listed in Table 2. The cross validation accuracy is 53.10%.

2.2 Classification

In the classification setting, we treat each label of quality as the same level, therefore we may lose some information due to the logical ranking of different labels. However, we see that classification can still lead to a good cross validation accuracy.

2.2.1 Logistic Regression

The simplest classification model is multinomial logistic regression. We use the nnet package to implement. We achieved a cross validation accuracy of 53.83%.

Table 2: Linear Regression Result

	quality		
Predictors	*Estimates*	*CI*	*p*
(Intercept)	35.96	12.91 – 59.01	**0.002**
fixed acidity	0.07	0.04 – 0.10	**<0.001**
volatile acidity	-1.33	-1.48 – -1.18	**<0.001**
citric acid	-0.23	-0.39 – -0.08	**0.004**
residual sugar	0.03	0.02 – 0.04	**<0.001**
chlorides	-0.14	-0.79 – 0.52	0.684
free sulfur dioxide	0.00	0.00 – 0.00	**0.001**
density	-35.52	-59.07 – -11.96	**0.003**
pH	0.42	0.24 – 0.60	**<0.001**
sulphates	0.75	0.60 – 0.90	**<0.001**
alcohol	0.31	0.28 – 0.34	**<0.001**
Observations	6497		
R^2 / R^2 adjusted	0.283 / 0.282		

2.2.2 SVM

Traditional SVM is used for binary classification. We generalize it to multi-class by using the one-against-one technique: we fit all-pairs binary classifiers and decide the final class by voting. We use the e1071 package to implement.

We have tried both the linear kernel and non-linear Gaussian RBF (radial kernel) kernel.

For the linear kernel, we use cross-validation to tune the parameter "cost". We get cost = 15 leads to the highest cross validation error: 59.23%.

For the radial kernel, by using grid search, we find gamma=1.5,cost=15 leads to the highest CV accuracy. The final validation accuracy is 66.66%.

2.2.3 Tree Models

2.2.3.1 Decision Tree

First we use decision tree classifier to model the data. We use rpart package to apply. In the rpart function, there are several parameters to tune: minimum split, complexity parameters, split method. Using 10 fold cross validation, we get the minimum split = 5 works well. For the complexity parameter, we plot the X-val Relative Error to get the best complexity parameter. As we can see from Figure 2, after cp=0.0018, the validation error remains stable, so we choose cp=0.0018. For the split method, we try both Gini impurity and information gain. Gini impurity leads to a higher validation accuracy.

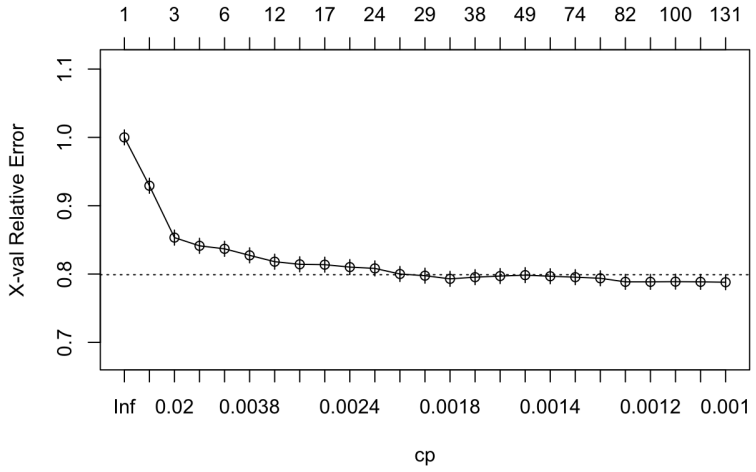


Figure 2： X-val Relative Error for Decision Tree

Using the above parameter, we get the validation accuracy: 54.90%.
The variable importance is listed in Table 3.

Table 3：Varialbe Importance for Decision Tree

alcohol	Density	volatile acidity	residual sugar	chlorides
450.67852	313.41202	243.28222	219.775632	217.39983
free sulfur dioxide	Sulphates	fixed acidity	citric acid	pH
197.47739	129.54007	126.60292	120.30829	95.24039

2.2.3.2 XGBoost

The gradient boosting machine predicts using weak learners and add learners to minimize the loss function. We use xgboost package to apply.
We set the minimum log loss as the evaluation metric here. Since the problem is multi-classification, we set the objective to be 'multi: softprob' and num_class = 7. We first use grid search following the order of: 1) nrounds and eta (which are the most important parameters of tree and boosting respectively; 2) max_depth and min_child_weight; 3) subsample. After that, we subdivide the intervals and tune by hand in order to get better results.

With the tuned parameters in the Table 4, we obtained validation accuracy: 69.20%.
We show the feature importance ranking (using Gini gain) in Table 5.

Table 4：Parameter tuning result for XGBoost

Parameter	Range	Best	Explanation
nrounds	20 to 110	100	number of boosting stages
eta (learning rate)	0.01 to 0.5	0.5	shrink each tree's contribution
max_depth	6 to 12	9	max depth of regression estimators
min_child_weight	1 to 6	1	min sum of instance weight needed in a child
subsample*	0.5 to 1	1	fraction of samples fitting base learners

*Note that by setting subsample not equal to 1 may turn out to have better results, but the results are not stable because the sample size is relatively small. So we take 1 for default here.

Table 5：Varialbe Importance for XGBoost

alcohol	volatile acidity	free sulfur dioxide	residual sugar	sulphates
0.8156850	0.11243519	0.08568982	0.08508908	0.005713912
chlorides	density	pH	citric acid	fixed acidity
0.07832758	0.07702708	0.07318819	0.07088464	0.06773072

2.2.2.3 Random Forest

Random forest is an ensemble algorithm based on decision trees. In the random forest model, there are two hyperparameters: the number of trees (ntree) and the number of variables preselected by tree nodes (mtry). mtry generally works better when taking the square root of the number of all independent variables. Here the number of independent variables is 11, so choose mtry=3.

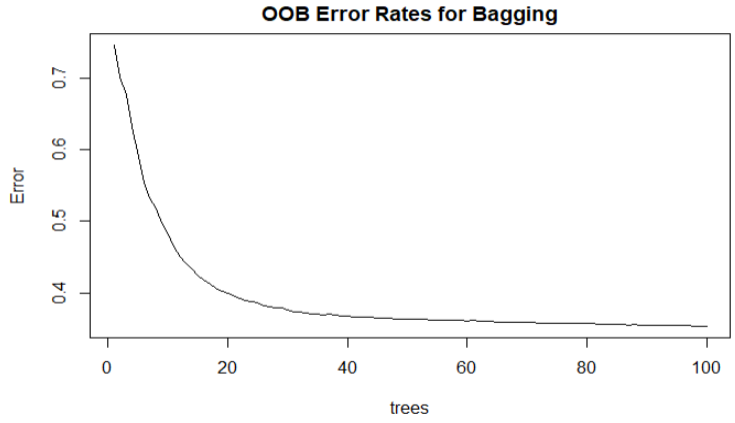


Figure 3： OOB Error Rates for Bagging of Random Forest

In order to find the suitable number of trees in a random forest, we draw the plot of OOB error rate. From Figure 3, we see that the error rate becomes stable after the number of trees reaches 60, so we choose 60 trees in the model.
The above hyper parameters lead to the cross validation accuracy of 68.83%.
We show the feature importance ranking (using Gini gain) below:

Table 6：Variable Importance for Random Forest

alcohol	density	volatile acidity	residual sugar	chlorides
0.009808338	0.011076923	0.005529667	0.007513285	0.005713912
free sulfur dioxide	sulphates	fixed acidity	citric acid	pH
0.004729214	0.003505321	0.003768605	0.004424125	0.003239989

3. Discussion

3.1 Model Comparison

So far, we utilized many tools to do the prediction. We then discuss the difference between them.

3.1.1 Accuracy

Table 7：Validation Accuray of Different Models

Linear Regression	Logistic Regression	SVM (Linear)	SVM (Radial)	Decision Tree	XGBoost	Random Forest
53.10%	53.83%	59.23%	66.66%	54.90%	69.20%	68.83%

As we can see from Table 7, XGBoost leads to the highest validation accuracy while linear regression has the lowest validation accuracy.

Regression v.s. Classification

We use linear regression to model this classification problem based on the idea that the quality label values have order meanings. However, we found that the linear regression performed the worst among all models. By taking a closer look at the linear regression's prediction, we notice that the 92.38% predicted data are of quality 5 and quality 6, which are the data's most frequent true labels (quality 5 and quality 6 are 76.55% in true data). This shows that Linear Regression performs weakly when the data is highly imbalanced. If we have a more balanced dataset, linear regression may lead to a better result. Furthermore, if we have a regression model that is not as sensitive as data imbalance as linear regression, we may have a better regression result.
However, as long as we can't solve the above two problems, our result shows that classification models lead to a higher accuracy in this data.

Linear v.s. Non-linear:

Linear regression, logistic regression and SVM with Linear Kernel have linear decision boundary, while all the other models are non-linear. As we can see from Table 7, except for Decision Tree, non-linear models' validation accuracy are all higher than linear models. This shows that the true relationship between predictors and the response variable is not linear.

Different Tree Models:
We tried tree models including Decision Tree, Random Forest and XGBoost. A single decision tree is relatively weak, but through applying Ensemble Learning, Random Forest and XGBoost yield much higher accuracy. However, this comes with a trade of running time. Random forest gives a more robust model and prevents overfitting, but as the number of trees increases, the process can be really costly in terms of running time. Rather than bagging, XGBoost applies boosting and can result in better performance than random forest after fine tuning. However, the tuning process takes an even longer time to run.

3.1.2 Variable Contribution:
We want to interpret how each variable contributes to the prediction of quality. We conduct the analysis based on the regression coefficients of linear regression and variable importance result of tree models.

Regression Coefficients:
Considering the magnitude (absolute value) of coefficients, as we can see from Table 3, the largest one is density. The absolute value of the coefficient is 36.64, while all the other coefficients have an absolute value smaller than 2. This shows that a small change in density would lead to a large increase in the quality.
However, this doesn't necessarily mean that density is the most important predictors since the coefficients could vary with the magnitude of predictors. For example, density changes in a small range [0.98711, 1.03898], while free sulfur dioxide changes in a large range [1, 289], so it is not surprising that the coefficient for free sulfur dioxide is much smaller than density. In order to better compare, it would be better if we standardize all predictors and then run regression.

Variable Importance:
We print out the feature importance of different tree models in Table 3,5,6. As we can see, "alcohol" is the most important variable for all 3 models. "density" is the 2nd important variable for decision trees and random forest. "volatile acidity" is among the top3 most important variables for all 3 models. So we conclude that "alcohol", "density", "volatile acidity" are the most important variables in predicting quality. This agrees with our intuition, since these variables have the relative high absolute correlation with quality.

3.2 Improvements

3.2.1 White Red split

We remove the variable “typered” in the above analysis. However, based on our common sense, white and red wine may have different evaluation standards, so we divide the data according to their type, and run classification models on them separately.

We also use 10-fold cross validation to run each model on red wine and white wine separately. The results are shown below. In the table, we also included the result of 10-fold cross validation on the entire dataset as comparison. We tried Logistic regression and SVM, since Random Forest and XGBoost takes too long to run and tune the parameters.

Table 8：Validation Accuray of Red/White Wine

	Logistic regression	SVM with radial kernel
White accuracy	59.22%	67.21%
Red accuracy	53.77%	66.86%
Total accuracy	53.85%	66.66%

As we can see, when training the model on different wine-type separately, the validation accuracy for red type has little or no improvements, while for white wine it has significant improvements.

We guess that this is because the sample size for red wine is relatively small (white wine has 4898 observations while red wine only has 1599 observations). The improvement on the white wine’s validation accuracy compared with total accuracy shows that there’s some difference in the quality evaluation for different types of wine. In order to further improve the model, we may include more interaction terms related with wine type.

3.2.2 Imbalanced

By looking at the distribution of the original data, we will find that the class of data is normally distributed, which means that for some certain classes, we may not have enough sample size for the model to train well.

As we can see from Table 1, the data set is not balanced: most data are of quality 5-6, while there’s little cases for quality 3 and 9. Since the data is not balanced, then by assigning the quality to more frequent class rather than less frequent class, would definitely lead to a higher accuracy.

In order to address the issue, we use an over-sampling method by sampling data from the less frequent class to make all classes have the same sample size. We use SVM with a radial kernel to run the oversampled data. Also using 10-fold cross validation, we get an validation accuracy of 67.03%, which is a small improvement of 66.66% when directly training using the original data.

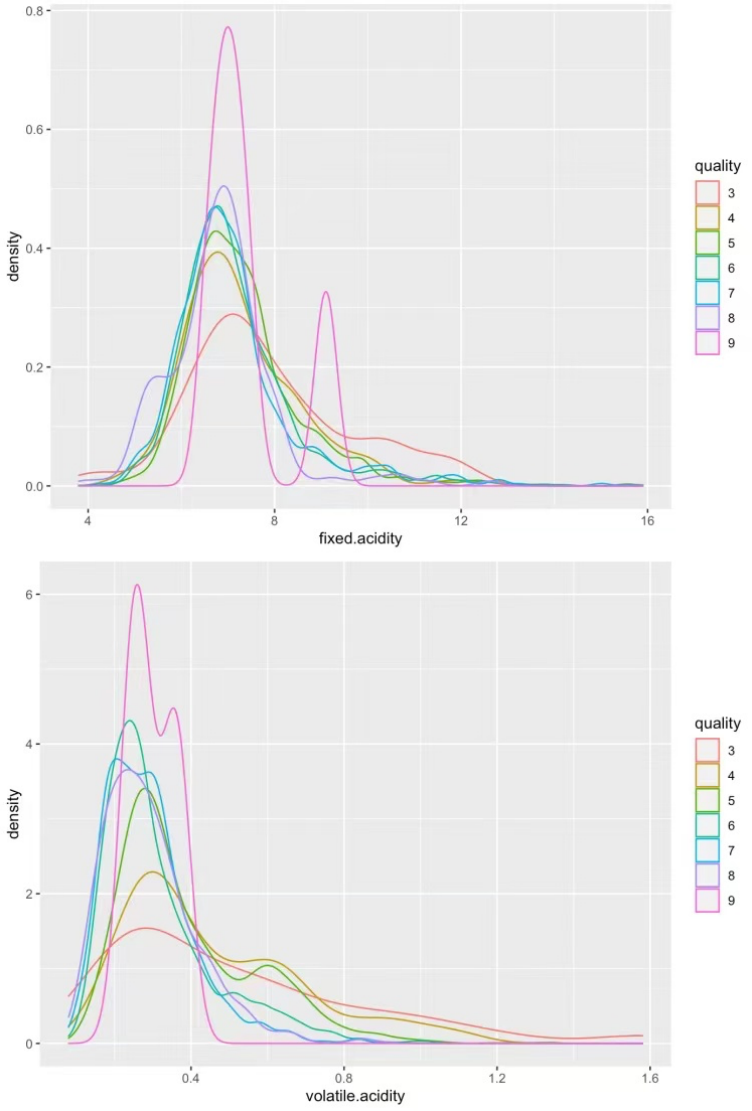
References

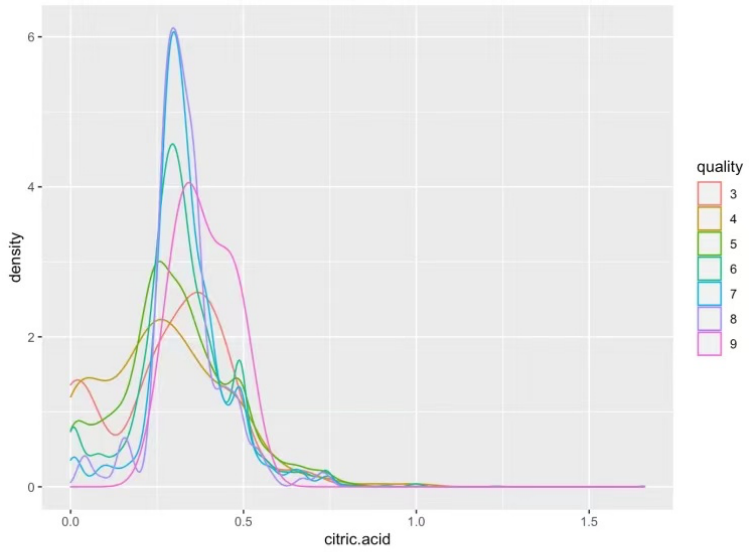
[1] Wine Quality Data Set: https://archive.ics.uci.edu/ml/datasets/wine+quality
[2] XGBoost Parameters: https://xgboost.readthedocs.io/en/latest/parameter.html

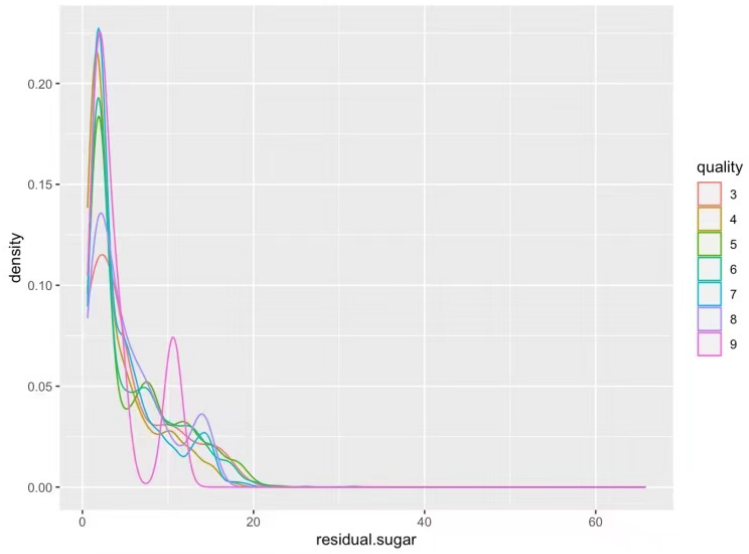
Appendix

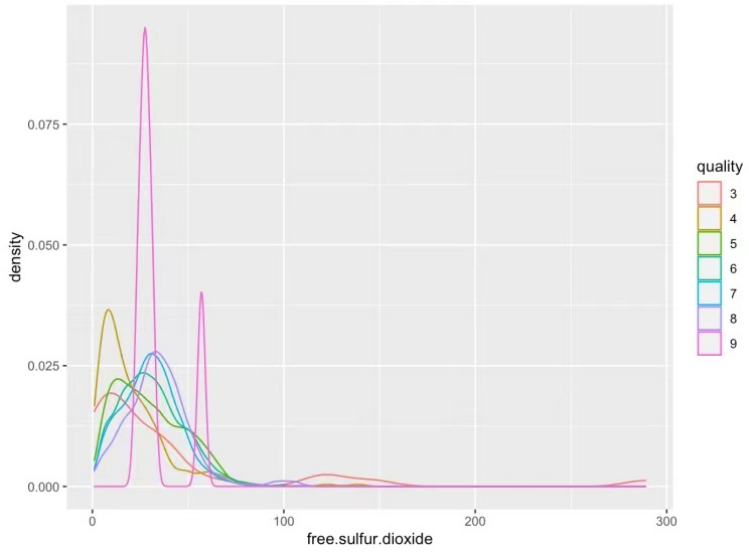
A1. Density Plots

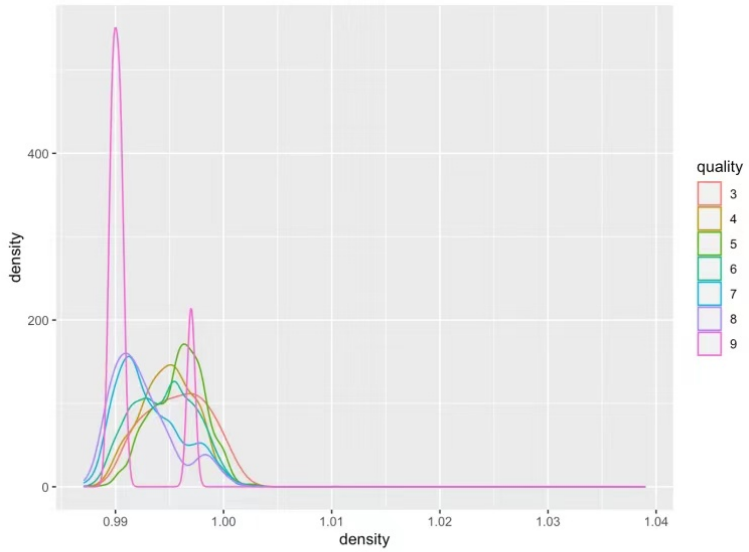
Also, to see the distribution of each variable, we created a density plot for each of the independent variables relative to the dependent variable quality.

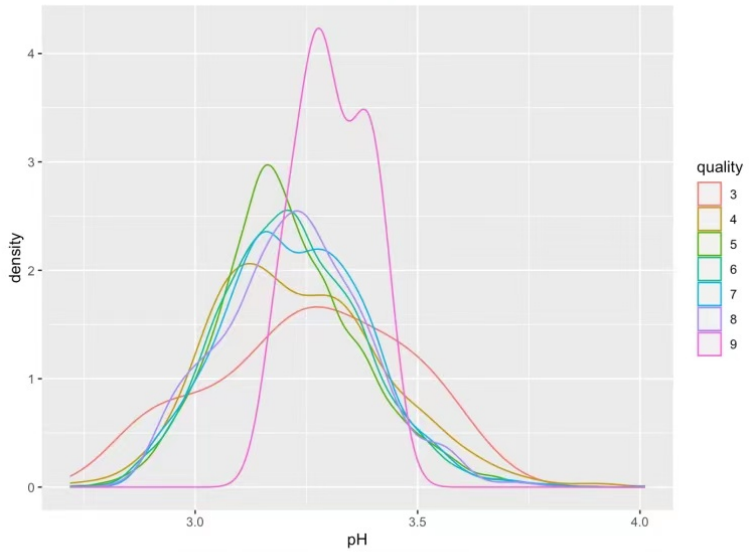


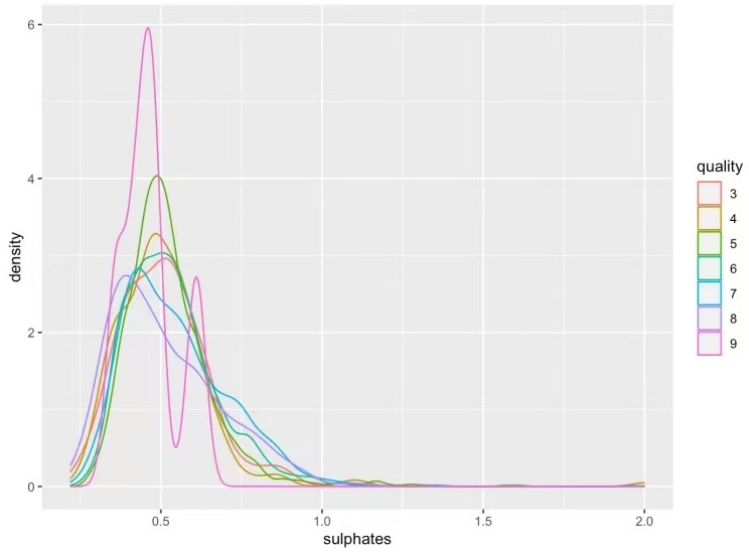

density
6
4
2
0
0.0
0.5
1.0
1.5
citric.acid
quality
3
4
5
6
7
8
9

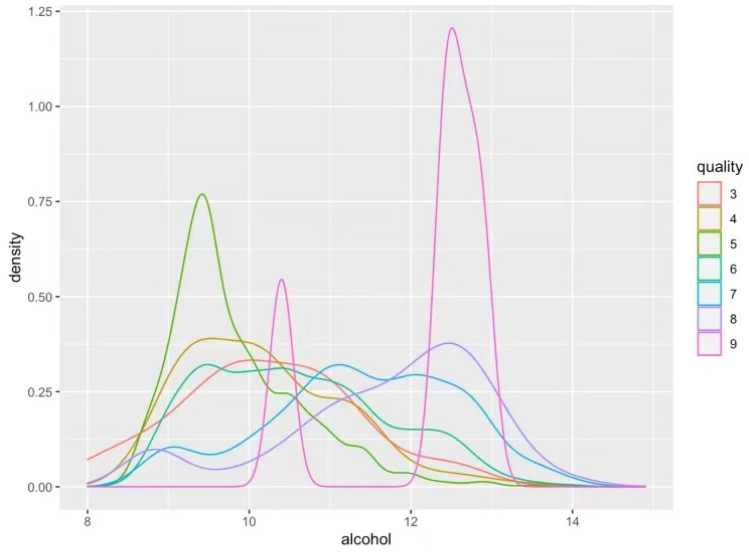

density
0.20
0.15
0.10
0.05
0.00
0
20
40
60
residual.sugar
quality
3
4
5
6
7
8
9

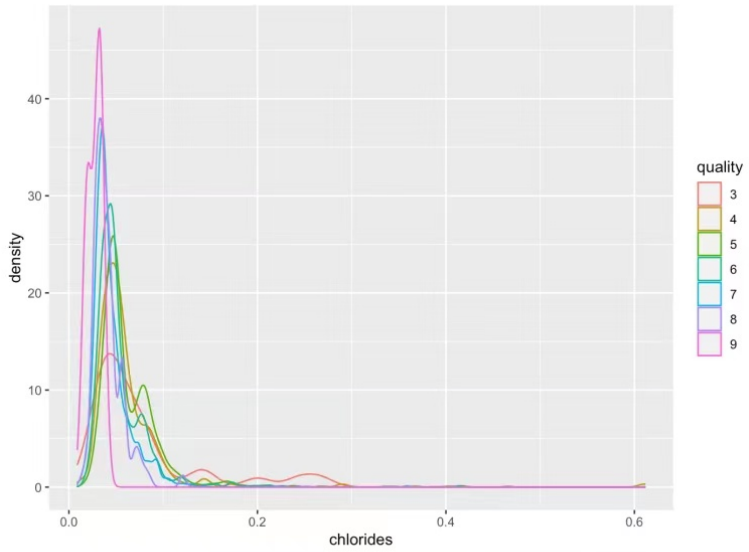

density
0.075
0.050
0.025
0.000
0
100
200
300
free.sulfur.dioxide
quality
3
4
5
6
7
8
9

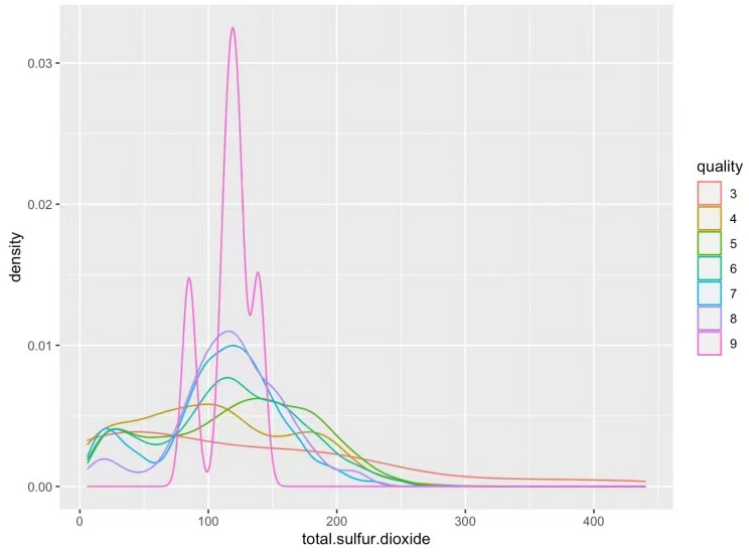

density
400
200
0
0.99
1.00
1.01
1.02
1.03
1.04
density
quality
3
4
5
6
7
8
9

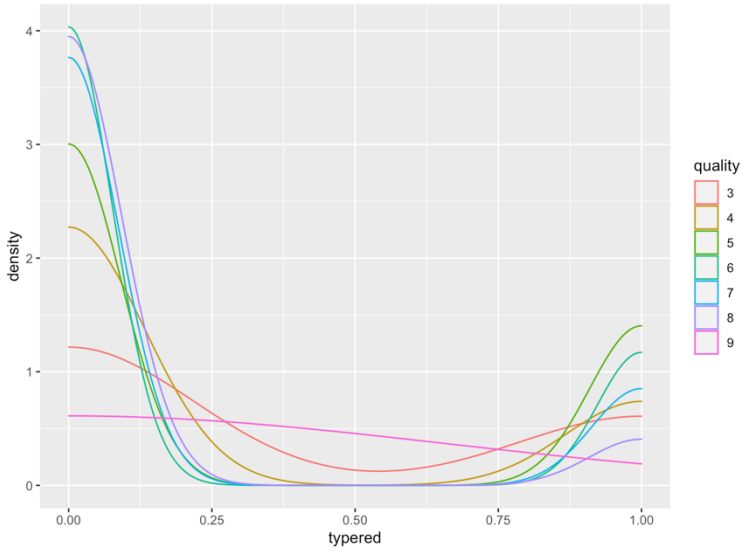

density
4
3
2
1
0
3.0
3.5
4.0
pH
quality
3
4
5
6
7
8
9


density
6
4
2
0
0.5
1.0
1.5
2.0
sulphates
quality
3
4
5
6
7
8
9


density
1.25
1.00
0.75
0.50
0.25
0.00
8
10
12
14
alcohol
quality
3
4
5
6
7
8
9


density
40
30
20
10
0
0.0
0.2
0.4
0.6
chlorides
quality
3
4
5
6
7
8
9


density
0.03
0.02
0.01
0.00
0
100
200
300
400
total.sulfur.dioxide
quality
3
4
5
6
7
8
9



A2. R Codes for Modeling

```
##############################################
##                                          ##
##              Preprocessing               ##
##                                          ##
##############################################

data = read.csv("wine-quality-white-and-red.csv")

library(dplyr)
data_encode = cbind(select_if(data,is.numeric),as.data.frame(model.matrix(~type-1,data)))
data_encoded1 = data_encode[,-14]

## White-red Index
index_red = which(data_encoded1$typered==1)

white_wine <- data_encoded[-index_red,]
red_wine <- data_encoded[index_red,]

drop <- c("total.sulfur.dioxide","typered")
data_encoded = data_encoded1[,!(names(data_encoded1) %in% drop)]

##############################################
##                                          ##
##                Modeling                  ##
##                                          ##
```

```
##########################################

## 2.1 Regression
## Linear Regression

folds = createFolds(data_encoded$quality,k=10)
cv_linear = lapply(folds, function(x){
  train_fold = data_encoded[-x,]
  test_fold = data_encoded[x,]
  linear_model <- lm(quality ~ fixed.acidity + volatile.acidity + citric.acid +
                     residual.sugar + chlorides + free.sulfur.dioxide + density+
                     pH + sulphates + alcohol,data=train_fold)
  return(linear_model)
})
summary(linear_model)

## chlorides has a higher p value
## so we fit the linear model again without this variable

## Comparison:

## Include chlorides:
folds = createFolds(data_encoded$quality,k=10)
cv_linear = lapply(folds, function(x){
  train_fold = data_encoded[-x,]
  test_fold = data_encoded[x,]
  linear_model <- lm(quality ~ fixed.acidity + volatile.acidity + citric.acid +
                     residual.sugar + chlorides + free.sulfur.dioxide + density+
                     pH + sulphates + alcohol,data=train_fold)
  linear_pred = predict(linear_model,newdata = test_fold)
  linear_pred= round(linear_pred)
  accuracy =
length(linear_pred[which(linear_pred==test_fold$quality)])/length(test_fold$quality)
  return(accuracy)
})
(accuracy = mean(as.numeric(cv_linear)))


## Without chlorides:
folds = createFolds(data_encoded$quality,k=10)
```

```
cv_linear = lapply(folds, function(x){
  train_fold = data_encoded[-x,]
  test_fold = data_encoded[x,]
  linear_model2 <- lm(quality ~ fixed.acidity + volatile.acidity + citric.acid +
                        residual.sugar + free.sulfur.dioxide + density+
                        pH + sulphates + alcohol,data=train_fold)
  linear_pred = predict(linear_model2,newdata = test_fold)
  linear_pred= round(linear_pred)
  accuracy =
length(linear_pred[which(linear_pred==test_fold$quality)])/length(test_fold$quality)
  return(accuracy)
})
(accuracy = mean(as.numeric(cv_linear)))


## 2.2 Classification
## 2.2.1 Multiple Logistic Regression

folds = createFolds(data_encoded$quality,k=10)
cv_multi_logistic = lapply(folds, function(x){
  train_fold = data_encoded[-x,]
  test_fold = data_encoded[x,]
  multi_logistic_model <- nnet::multinom(quality ~ fixed.acidity + volatile.acidity + citric.acid+
residual.sugar + chlorides + free.sulfur.dioxide + density+
                                         pH + sulphates + alcohol,data = train_fold)
  predicted.classes <- multi_logistic_model %>% predict(test_fold)
  accuracy =
length(multi_logistic_model[which(predicted.classes==test_fold$quality)])/length(test_fold$qual
ity)
  return(accuracy)
}
(accuracy = mean(as.numeric(cv_multi_logistic)))


## Discussion: red and white

## red-wine multi logistic regression
folds = createFolds(red_wine$quality,k=10)
cv_multi_logistic_red = lapply(folds, function(x){
  train_fold = red_wine[-x,]
```

```
  test_fold = red_wine[x,]
  multi_logistic_model <- nnet::multinom(quality ~ fixed.acidity + volatile.acidity + citric.acid+
residual.sugar + chlorides + free.sulfur.dioxide + density+ pH + sulphates + alcohol,data =
train_fold)
  predicted.classes <- multi_logistic_model %>% predict(test_fold)
  accuracy =
length(multi_logistic_model[which(predicted.classes==test_fold$quality)])/length(test_fold$qual
ity)
  return(accuracy)
})
(accuracy = mean(as.numeric(cv_multi_logistic_red)))

## white wine multi-logistic regression
folds = createFolds(white_wine$quality,k=10)
cv_multi_logistic_white = lapply(folds, function(x){
  train_fold = white_wine[-x,]
  test_fold = white_wine[x,]
  multi_logistic_model2 <- nnet::multinom(quality ~ fixed.acidity + volatile.acidity + citric.acid+
residual.sugar + chlorides + free.sulfur.dioxide + density+
                                          pH + sulphates + alcohol,data = train_fold)
  predicted.classes <- multi_logistic_model2 %>% predict(test_fold)
  accuracy =
length(multi_logistic_model2[which(predicted.classes==test_fold$quality)])/length(test_fold$qu
ality)
  return(accuracy)
})
(accuracy <- mean(as.numeric(cv_multi_logistic_white)))

## Discussion: binary logistic regression

data_encoded$quality_dummy <- ifelse(data_encoded$quality >= 6,1,0)
data_encoded = data_encoded[,-12]
data_encoded

folds = createFolds(data_encoded$quality_dummy,k=10)
cv_binary_logistic = lapply(folds, function(x){
  train_fold = data_encoded[-x,]
  test_fold = data_encoded[x,]
```

```
  binary_logistic <- glm(quality_dummy ~ fixed.acidity + volatile.acidity + citric.acid+
residual.sugar + chlorides + free.sulfur.dioxide + density +
                         pH + sulphates + alcohol, family=binomial, data=train_fold)
  predicted.classes <- binary_logistic %>% predict(test_fold)
  predictions = (predict(binary_logistic,   test_fold
[,c("fixed.acidity","volatile.acidity","citric.acid","residual.sugar","chlorides",
                                              "free.sulfur.dioxide","density","pH","sulphates","alcohol")],
                       type='response') > 0.5)
  valid_true_labels = (test_fold [,c("quality_dummy")] == 1)
  accuracy = mean(predictions == valid_true_labels)
  return(accuracy)
})

table_binary = mean(as.numeric(cv_binary_logistic))

## 2.2.2 SVM: linear kernel & radial kernel

## SVM linear kernel
folds = createFolds(data_encoded$quality,k=10)
cv_SVM_linear = lapply(folds, function(x){
  train_fold = data_encoded[-x,]
  test_fold = data_encoded[x,]
  svm_model = svm(quality~.,data=train_fold, kernal="linear", cost = 15)
  svm_pred = predict(svm_model,newdata = as.matrix(test_fold[-11]))
  svm_pred= round(svm_pred)
  accuracy = length(svm_pred[which(svm_pred==test_fold$quality)])/length(test_fold$quality)
  return(accuracy)
})
(accuracy = mean(as.numeric(cv_SVM_linear)))

## SVM radial kernel
folds = createFolds(data_encoded$quality,k=10)
cv_SVM_radial = lapply(folds, function(x){
  train_fold = dat3[-x,]
  test_fold = dat3[x,]
  svm_model = svm(quality~.,data=train_fold, kernal="radial",gamma=1.5,cost=15)
  svm_pred = predict(svm_model,newdata = as.matrix(test_fold[-11]))
  svm_pred= round(svm_pred)
  accuracy = length(svm_pred[which(svm_pred==test_fold$quality)])/length(test_fold$quality)
  return(accuracy)
```

```
})
(accuracy = mean(as.numeric(cv_SVM_radial)))

## Discussion: red and white

## red-wine SVM
cv_SVM_red = lapply(folds, function(x){
  train_fold = red_wine[-x,]
  test_fold = red_wine[x,]
  svm_model = svm(quality~.,data=train_fold, kernal="radial",gamma=1.5,cost=5)
  svm_pred = predict(svm_model,newdata = as.matrix(test_fold[-11]))
  svm_pred= round(svm_pred)
  accuracy = length(svm_pred[which(svm_pred==test_fold$quality)])/length(test_fold$quality)
  return(accuracy)
})
(accuracy = mean(as.numeric(cv_SVM_red)))

## white-wine SVM
cv_SVM_white = lapply(folds, function(x){
  train_fold = white_wine[-x,]
  test_fold = white_wine[x,]
  svm_model = svm(quality~.,data=train_fold, kernal="radial",gamma=1.5,cost=5)
  svm_pred = predict(svm_model,newdata = as.matrix(test_fold[-11]))
  svm_pred= round(svm_pred)
  accuracy = length(svm_pred[which(svm_pred==test_fold$quality)])/length(test_fold$quality)
  return(accuracy)
})
(accuracy = mean(as.numeric(cv_SVM_white)))

## Discussion: imbalanced sampling

## oversampling SVM
folds = createFolds(data_encoded$quality,k=10)
cv_SVM_oversampling = lapply(folds, function(x){
  train_fold = data_encoded[-x,]
  test_fold = data_encoded[x,]
  balanced_train_fold <- upSample(x = train_fold[,-11],
                                  y = factor(train_fold$quality))
  balanced_train_fold$Class = as.numeric(paste(balanced_train_fold$Class))
  svm_model = svm(Class~.,data=balanced_train_fold, kernal="radial",gamma=1.5,cost=13)
```

```
  svm_pred = predict(svm_model,newdata = as.matrix(test_fold[-11]))
  svm_pred= round(svm_pred)
  accuracy = length(svm_pred[which(svm_pred==test_fold$quality)])/length(test_fold$quality)
  return(accuracy)
})
(accuracy = mean(as.numeric(cv_SVM_oversampling)))

## 2.2.3 Trees
## 2.2.3.1 Decision Tree

library(rpart)

folds = createFolds(data_encoded$quality,k=10)
cv_decision_tree = lapply(folds, function(x){
  train_fold = data_encoded[-x,]
  test_fold = data_encoded[x,]
  ct <- rpart.control(xval=10, minsplit=5, cp=.0018)
  model_tree<-rpart(quality~.,data=train_fold, method="class",control = ct,
parms=list(split="gini"))
  tree_pred<-predict(model_tree,newdata=test_fold[-11],type="class")
  accuracy = length(tree_pred[which(tree_pred==test_fold$quality)])/length(test_fold$quality)
  return(accuracy)
})
(accuracy = mean(as.numeric(cv_decision_tree)))


## 2.2.3.2 Random Forest

library(randomForest)
library(caret)

## data_encoded1 includes 'typered' and 'total.sulfur.dioxide'
folds = createFolds(data_encoded1$quality,k=10)

cv_random_forest_not_rm = lapply(folds, function(x){
  train_fold = data_encoded1[-x,]
  test_fold = data_encoded1[x,]
  begclassifier = randomForest(y = train_fold$quality,x = train_fold[-12],
                               ntree =60,mtry=3,importance=TRUE, proximity=TRUE)
  beg.pred = predict(begclassifier,newdata = as.matrix(test_fold[-12]))
```

```
  beg.class = round(beg.pred)
  accuracy = length(beg.class[which(beg.class==test_fold$quality)])/length(test_fold$quality)
  return(accuracy)
})

(accuracy = mean(as.numeric(cv_random_forest_not_rm)))

## feature importance
begclassifier = randomForest(y = data_encoded1$quality,x = data_encoded1[c(-12,-13,-7)],
                             ntree =60,mtry=3,importance=TRUE, proximity=TRUE)

sort(begclassifier$importanceSD)

## find number of trees
begclassifier = randomForest(y = data_encoded1$quality,x = data_encoded1[c(-12,-13,-7)],
                             ntree =100,mtry=3,importance=TRUE, proximity=TRUE)
plot(begclassifier,main="OOB Error Rates for Bagging")
MDSplot(begclassifier)

## remove 'typered' and 'total.sulfur.dioxide'
folds = createFolds(data_encoded$quality,k=10)

cv_random_forest = lapply(folds, function(x){
  train_fold = data_encoded[-x,]
  test_fold = data_encoded[x,]
  begclassifier = randomForest(y = train_fold$quality,x = train_fold[-11],
                               ntree =100,importance=TRUE, proximity=TRUE)
  beg.pred = predict(begclassifier,newdata = as.matrix(test_fold[-11]))
  beg.class = round(beg.pred)
  accuracy = length(beg.class[which(beg.class==test_fold$quality)])/length(test_fold$quality)
  return(accuracy)
})

(accuracy = mean(as.numeric(cv_random_forest)))

## 2.2.3 XGBoost

library(xgboost)

## number of class
```

```
(numofclass = unique(data_encoded$quality))

## change quality class to (0,1,2,3,4,5,6) in order to fit xgb
data_encoded$quality = data_encoded$quality - 3

xgb_params<-list("objective" = "multi:softprob",
                 "eval_metric" = "mlogloss",
                 "num_class" = 7)

folds = createFolds(data_encoded$quality,k=10)

cv_xgb = lapply(folds, function(x){
  train_fold = data_encoded[-x,]
  test_fold = data_encoded[x,]
  train_data = as.matrix(train_fold[,-11])
  test_data = as.matrix(test_fold[,-11])
  train_label = train_fold[,11]
  test_label = test_fold[,11]
  train_matrix = xgb.DMatrix(data = train_data, label = train_label)
  test_matrix = xgb.DMatrix(data = test_data, label = test_label)
  xgb.model <-xgb.train(params = xgb_params,
                        data = train_matrix,
                        nrounds = 100,
                        eta = 0.5,
                        max_depth = 9,
                        min_child_weight = 1,
                        subsample = 1)
  xgb.pred = predict(xgb.model,newdata = test_matrix)
  xgb.prediction = matrix(xgb.pred,nrow = 7, ncol = length(xgb.pred)/7)%>%
    t()%>%
    data.frame()%>%
    mutate(max_prob = max.col(.,"last"))
  xgb.class = xgb.prediction$max_prob
  accuracy = length(xgb.class[which(xgb.class==(test_label)+1)])/length(test_label)
  return(accuracy)
})
(accuracy = mean(as.numeric(cv_xgb)))

## Discussion: imbalanced sampling
```

```
## oversampling XGBoost
xgb_params<-list("objective" = "multi:softprob",
                 "eval_metric" = "mlogloss",
                 "num_class" = 7)

cv_xgb_oversampling = lapply(folds, function(x){
  train_fold = data_encoded[-x,]
  test_fold = data_encoded[x,]
  balanced_train_fold <- upSample(x = train_fold[,-11],
                                  y = factor(train_fold$quality))
  balanced_train_fold$Class = as.numeric(paste(balanced_train_fold$Class))
  train_data = as.matrix(balanced_train_fold[,-11])
  test_data = as.matrix(test_fold[,-11])
  train_label = balanced_train_fold[,11]
  test_label = test_fold[,11]
  train_matrix = xgb.DMatrix(data = train_data, label = train_label)
  test_matrix = xgb.DMatrix(data = test_data, label = test_label)
  xgb.model <-xgb.train(params = xgb_params,
                        data = train_matrix,
                        nrounds = 115,
                        eta = 0.5,
                        max_depth = 10,
                        min_child_weight = 1,
                        subsample = 1)
  xgb.pred = predict(xgb.model,newdata = test_matrix)
  xgb.prediction = matrix(xgb.pred,nrow = 7, ncol = length(xgb.pred)/7)%>%
    t()%>%
    data.frame()%>%
    mutate(max_prob = max.col(.,"last"))
  xgb.class = xgb.prediction$max_prob
  accuracy = length(xgb.class[which(xgb.class==(test_label)+1)])/length(test_label)
  return(accuracy)
})
(accuracy = mean(as.numeric(cv_xgb_oversampling)))
```