Predicting Wine Quality with ML

Omer Rubin

2023-09-12

Table of Contents

Introduction

In this project, my aim is to predict wine quality based on various aspects of its production process. I used the 'Red Wine Quality' dataset I found on Kaggle, which is available in the UCI Machine Learning Repository, comprising 1599 samples and 12 variables. A new categorical variable, 'QualityType', was introduced to categorize wine quality based on its numerical rating. Given the substantial number of observations, I opted for a 70/30 data split, considering that 70% should provide sufficient data for model training.

nitial data exploration techniques included generating a correlation matrix and histograms to flag any significant outliers or patterns among the variables. After identifying these, I scaled the relevant predictors to normalize their range and introduced an interaction term between two specific variables that appeared to have a synergistic effect on wine quality.

For the machine learning aspect, I tested a series of models—elastic net, SVM, basic decision trees, random forests, and boosted trees—to gauge their efficacy. The random forest model stood out for its superior predictive accuracy. When this model was applied to the test dataset, it yielded a roc_auc value closely aligned with the training set. This strong performance was further corroborated through heatmap matrices and ROC curve plots, effectively confirming the model's reliability in predicting wine quality.

.

.

.

.

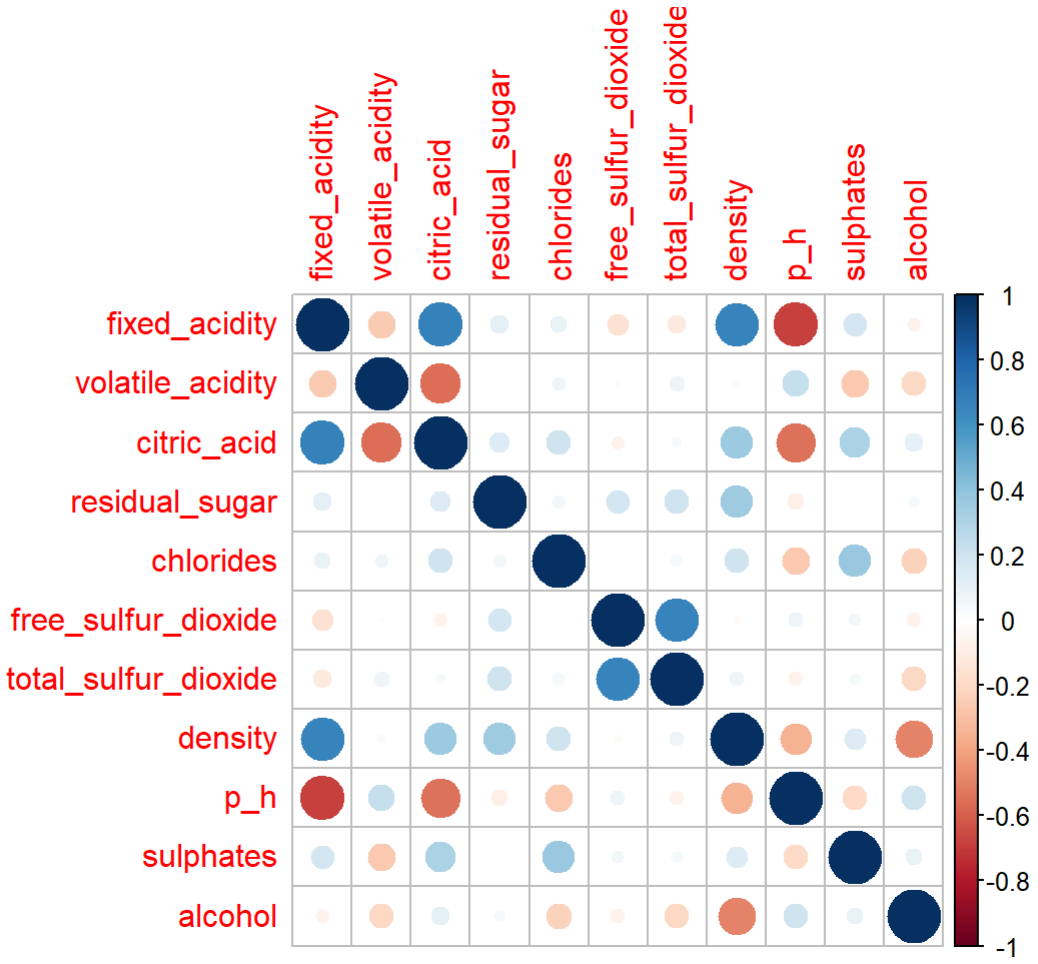
.

.

.

Exploratory Data Analysis

We will start off by checking for correlations among the predictors using a correlation matrix

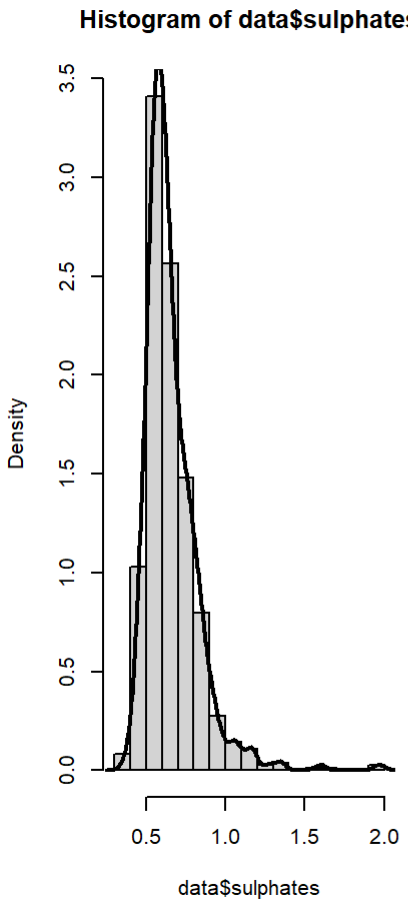


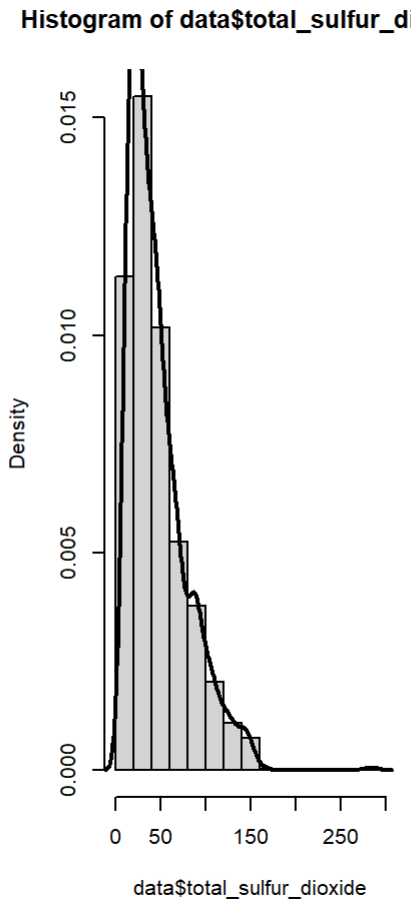
From the correlation matrix, it's evident that fixed acidity is strongly correlated with citric acid and density while also having a negative correlation with pH. Moreover, there is a strong correlation between total sulfur dioxide and free sulfur dioxide, which is logical since the latter is a component of the former. Lastly, pH has a significant negative correlation with both fixed acidity and citric acid. Likewise, volatile acidity shows a strong negative correlation with citric acid, as well as with alcohol and density

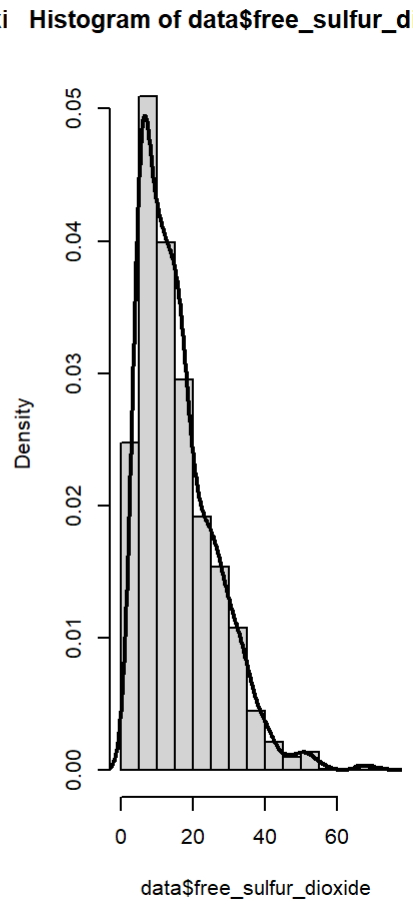
Now we examine each variable for outliers that may require scaling in future steps

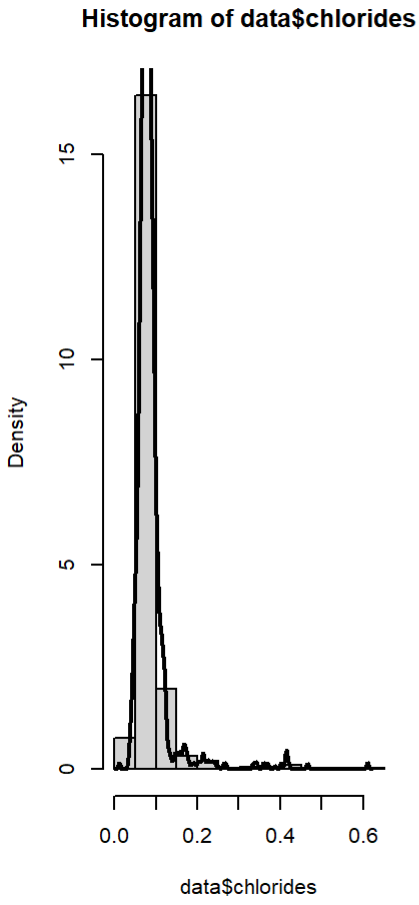
```
##  fixed_acidity   volatile_acidity  citric_acid    residual_sugar
##  Min.   : 4.60   Min.   :0.1200   Min.   :0.000   Min.   : 0.900
##  1st Qu.: 7.10   1st Qu.:0.3900   1st Qu.:0.090   1st Qu.: 1.900
##  Median : 7.90   Median :0.5200   Median :0.260   Median : 2.200
##  Mean   : 8.32   Mean   :0.5278   Mean   :0.271   Mean   : 2.539
##  3rd Qu.: 9.20   3rd Qu.:0.6400   3rd Qu.:0.420   3rd Qu.: 2.600
##  Max.   :15.90   Max.   :1.5800   Max.   :1.000   Max.   :15.500
##    chlorides       free_sulfur_dioxide total_sulfur_dioxide    density
##  Min.   :0.01200   Min.   : 1.00       Min.   :  6.00       Min.   :0.9901
##  1st Qu.:0.07000   1st Qu.: 7.00       1st Qu.: 22.00       1st Qu.:0.9956
##  Median :0.07900   Median :14.00       Median : 38.00       Median :0.9968
##  Mean   :0.08747   Mean   :15.87       Mean   : 46.47       Mean   :0.9967
##  3rd Qu.:0.09000   3rd Qu.:21.00       3rd Qu.: 62.00       3rd Qu.:0.9978
##  Max.   :0.61100   Max.   :72.00       Max.   :289.00       Max.   :1.0037
##       p_h          sulphates         alcohol
##  Min.   :2.740   Min.   :0.3300   Min.   : 8.40
##  1st Qu.:3.210   1st Qu.:0.5500   1st Qu.: 9.50
##  Median :3.310   Median :0.6200   Median :10.20
##  Mean   :3.311   Mean   :0.6581   Mean   :10.42
##  3rd Qu.:3.400   3rd Qu.:0.7300   3rd Qu.:11.10
##  Max.   :4.010   Max.   :2.0000   Max.   :14.90
```

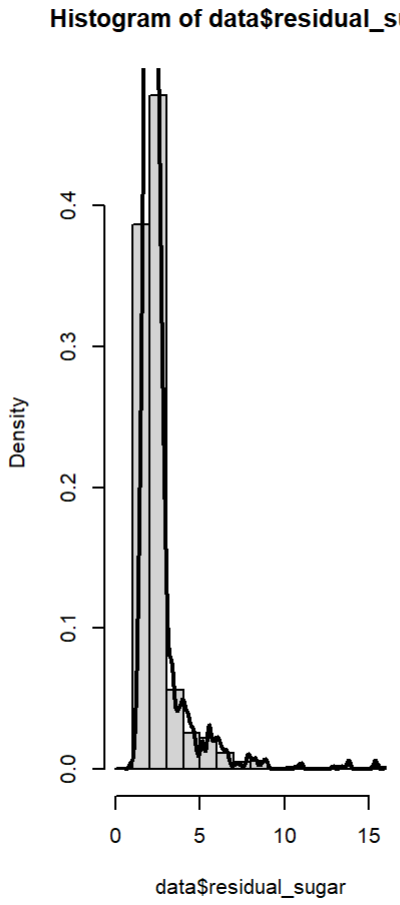
The summary suggests that the distributions of sulphates, total sulfur dioxide, free sulfur dioxide, chlorides, and residual sugar are significantly skewed. This observation is further confirmed by examining their respective histograms.


Histogram of data$sulphates
Density
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0.5
1.0
1.5
2.0
data$sulphates


Histogram of data$total_sulfur_dioxi
Density
0.000
0.005
0.010
0.015
0
50
150
250
data$total_sulfur_dioxide


Histogram of data$free_sulfur_dioxi
Density
0.00
0.01
0.02
0.03
0.04
0.05
0
20
40
60
data$free_sulfur_dioxide


Histogram of data$chlorides
Density
0
5
10
15
0.0
0.2
0.4
0.6
data$chlorides


Histogram of data$residual_sugar
Density
0.0
0.1
0.2
0.3
0.4
0
5
10
15
data$residual_sugar

Data Splitting And Cross-Validation

I partitioned the dataset into a 70/30 split, allocating 70% of the data for training and the remaining 30% for testing. This division was made to ensure a sufficient amount of data for robust model training prior to evaluation. I used stratified sampling based on the "QualityType" variable, as it is our primary variable of interest. Following this, I divided the data into 10 stratified folds, again based on "QualityType," to ensure each fold contained a balanced representation of different wine quality ratings.

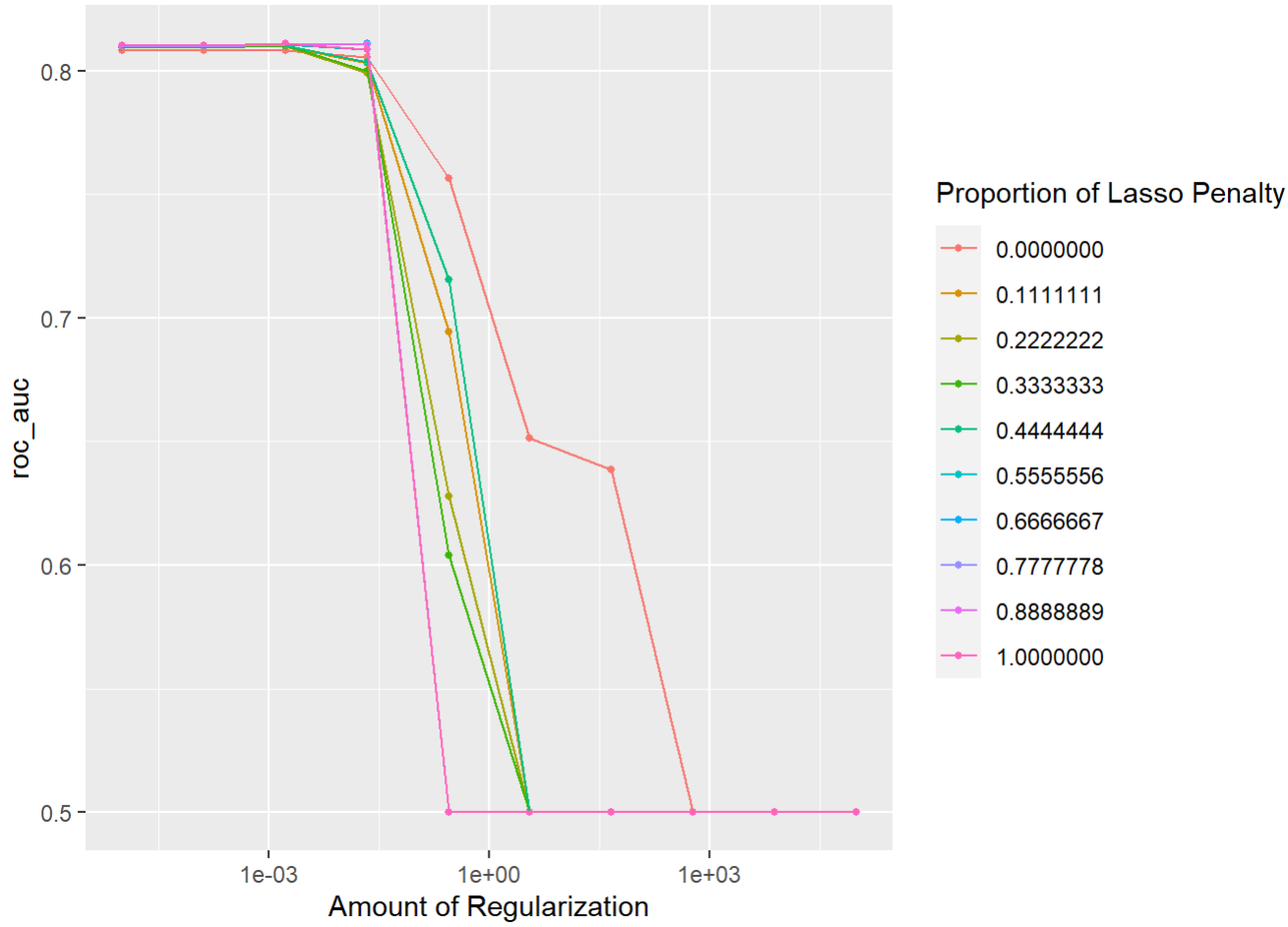
Then, a recipe was created for the training data, which included scaling the predictors we talked about with high outliers and introducing interaction terms between residual sugar and chlorides. This was motivated by the roles these variables play in wine-making: residual sugar influences the wine's sweetness, while chlorides contribute to its bitterness, and their interaction affects the overall flavor profile of the wine.

```
set.seed(111)


split <- initial_split(data, strata = QualityType, prop = 0.7)
training <- training(split)
testing <- testing(split)

#folding
training_fold <- vfold_cv(training, v = 10, strata = QualityType)

#recipe
recipe <- recipe(QualityType ~ fixed_acidity + volatile_acidity + citric_acid + residual_sugar +
chlorides + free_sulfur_dioxide + total_sulfur_dioxide + density + p_h + sulphates + alcohol, da
ta = training) %>%
step_dummy(all_nominal_predictors()) %>%
step_scale(residual_sugar, chlorides, free_sulfur_dioxide, total_sulfur_dioxide, sulphates) %>%
step_interact(residual_sugar ~ sulphates)
```

Model Fitting/Training

Elastic Net Model

We will start off with an Elastic Net model, which uses the penalties from both Lasso and Ridge regression. Elastic net should be used as we have groups of highly correlated independent variables. For example, total_sulfur_dioxide and free_sulfur_dioxide form a highly correlated group that is independent from the highly correlated group of fixed_acidity and citric_acid. We will be tuning the hyperparameters mixture and penalty to capture a wide variety of models. Below is a graph of all the models and a table of summary statistics of the best performing models.

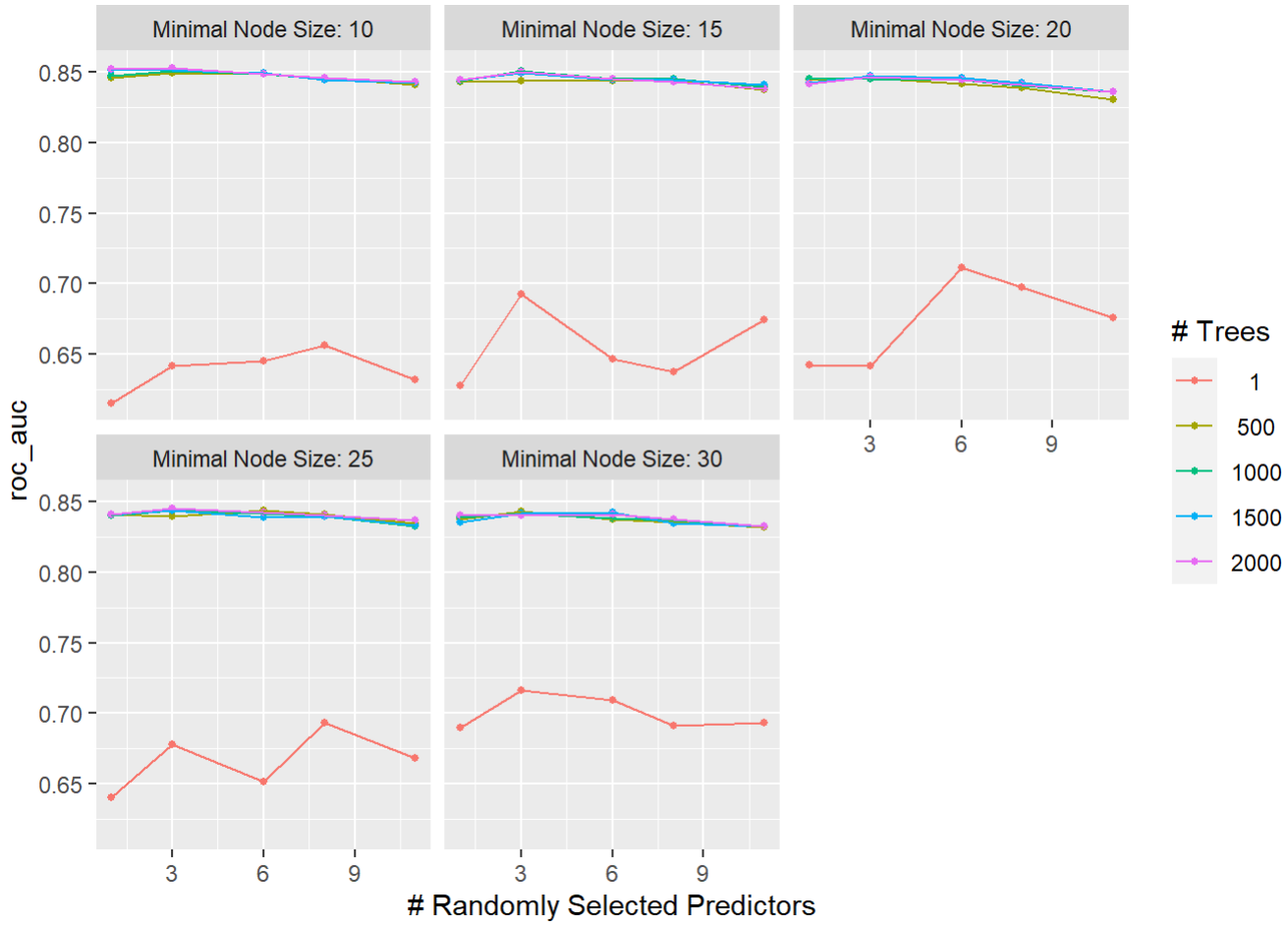


Metric	Value	Standard_Error	Penalty_Value	Mixture_Value
roc_auc	0.8108850	0.0157270	0.0016681	1.0000000
roc_auc	0.8108604	0.0160230	0.0215443	0.6666667
roc_auc	0.8106027	0.0156360	0.0016681	0.8888889
roc_auc	0.8105880	0.0157872	0.0016681	0.7777778
roc_auc	0.8104760	0.0159211	0.0016681	0.5555556
roc_auc	0.8104316	0.0162720	0.0215443	0.7777778

From the table, it's evident that the best Elastic Net model achieved a roc_auc of 0.81088 and had a standard error of 0.0157. Using a penalty value of 0.0016681 and a mixture value set at 1.00.

Random Forest Model

Next, we'll employ a random forest model, which is particularly suitable since we have a large numer of observations. This approach can also function as a bagging model; in our context, a random forest model with n = 11 serves that purpose. We'll tune mtry, trees, and min_n. The value of min_n will be constrained to the range (1,11). Any value outside of this interval would render the model pretty useless, as it's impractical to use either more than 11 or fewer than 1 predictor variables. Additionally, we'll use 5 levels instead of 10 to avoid generating unnecessary models that might not improve performance.

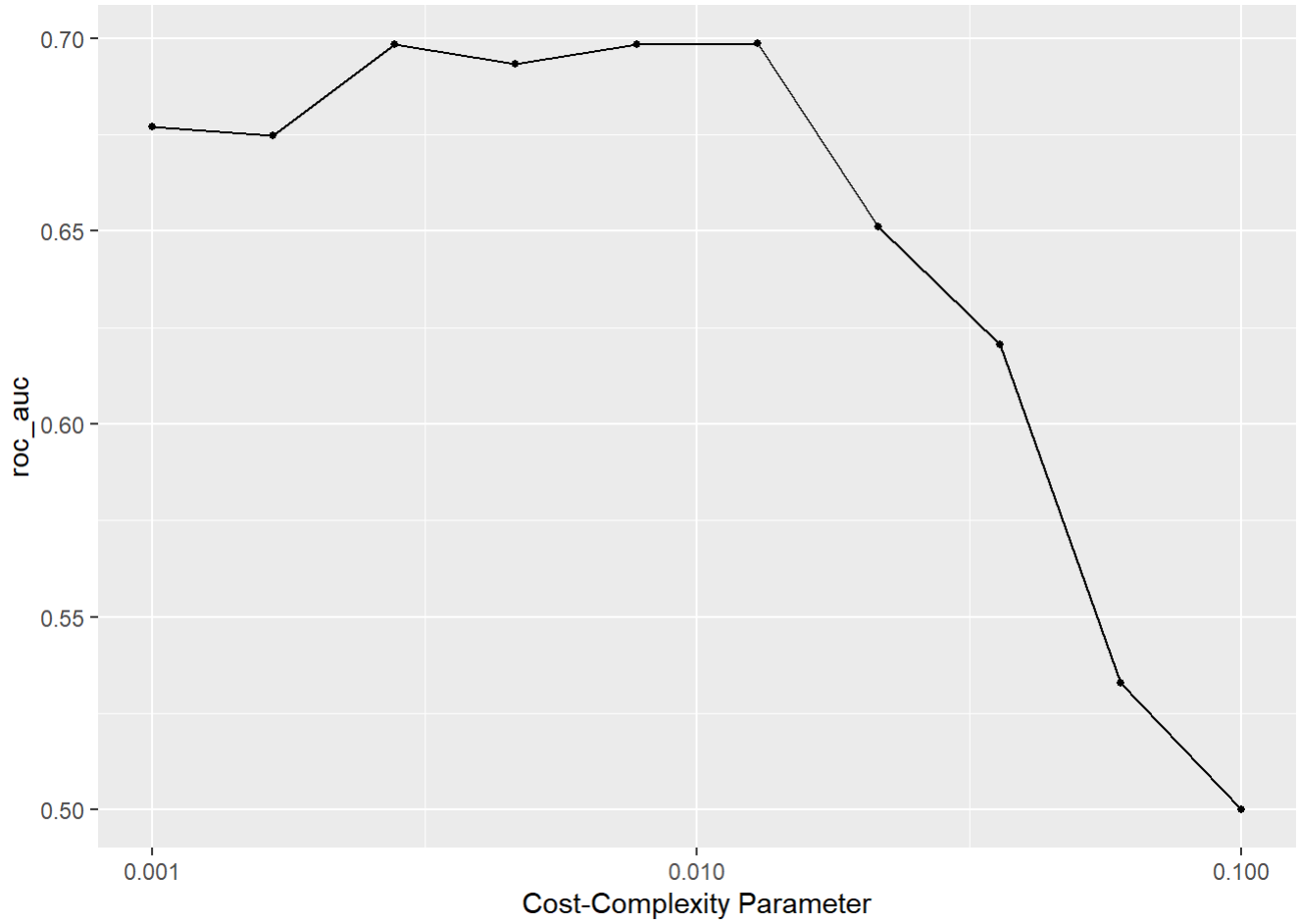


Metric	Value	Standard_Error	mtry_Value	trees_Value	min_n_Value
roc_auc	0.8531677	0.0148602	3	2000	10
roc_auc	0.8520025	0.0140932	1	2000	10
roc_auc	0.8516857	0.0154222	3	1500	10
roc_auc	0.8513134	0.0146157	1	1500	10
roc_auc	0.8510966	0.0155769	3	1000	10
roc_auc	0.8510811	0.0154217	3	1000	15

Based on the results, our random forest model performed better than the elastic net model. It had a ROC_AUC score of 0.853 and had a standard error of 0.048. The model had an mtry value of 3, was built with 2000 trees, and had a min_n value of 10. As the mtry value doesn't equal 11, we can deduce that this isn't a bagging model.

Basic Tree with Tuned Hyper-parameters

Next, we'll display a basic tree graph. With 11 predictors at our disposal, there are multiple methods for incorporating them. As such, we're interested in examining how varying inputs might yield diverse outcomes or predictions.

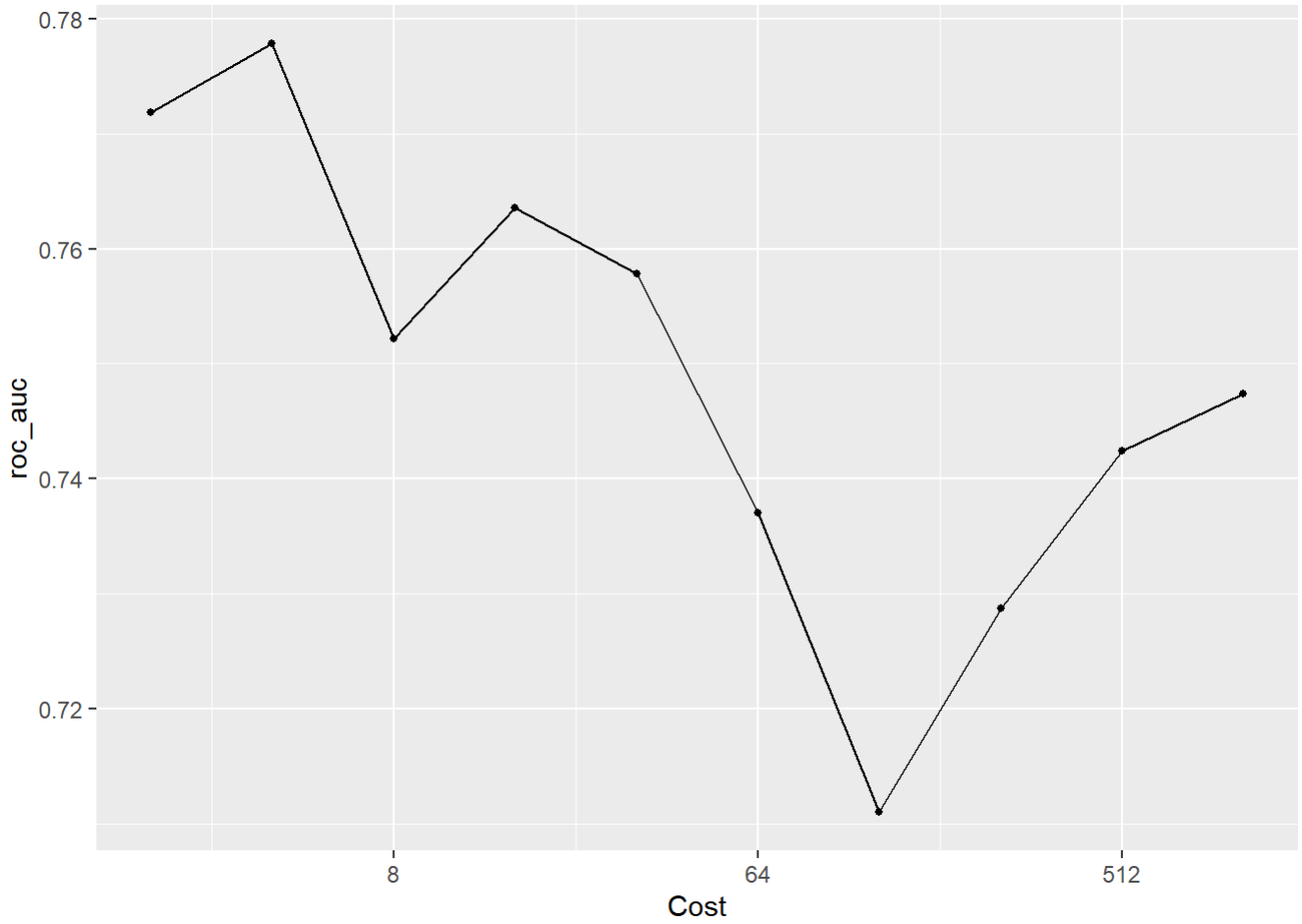


Metric	Value	Standard_Error	Cost_Complexity_Value
roc_auc	0.6986038	0.0171716	0.0129155
roc_auc	0.6984402	0.0188064	0.0027826
roc_auc	0.6983184	0.0167570	0.0077426
roc_auc	0.6932617	0.0167327	0.0046416
roc_auc	0.6771286	0.0276894	0.0010000
roc_auc	0.6748261	0.0272061	0.0016681

From our observing the table, we can see that the top-performing tree model achieved a ROC_AUC score of 0.6986, along with a standard error of 0.01717 and a cost complexity of 0.00129. Clearly, this is our worst performing model yet.

SVM Model

Next we will use a SVM model. The goal is to categorize wine quality, which can fall into one of three of our distinct classes. Therefore, The SVM model will aid in classifying wines into these categories using our predictors' values. We will tune the hyperparameter of cost. The hyperparameter 'cost' will be tuned for optimization.

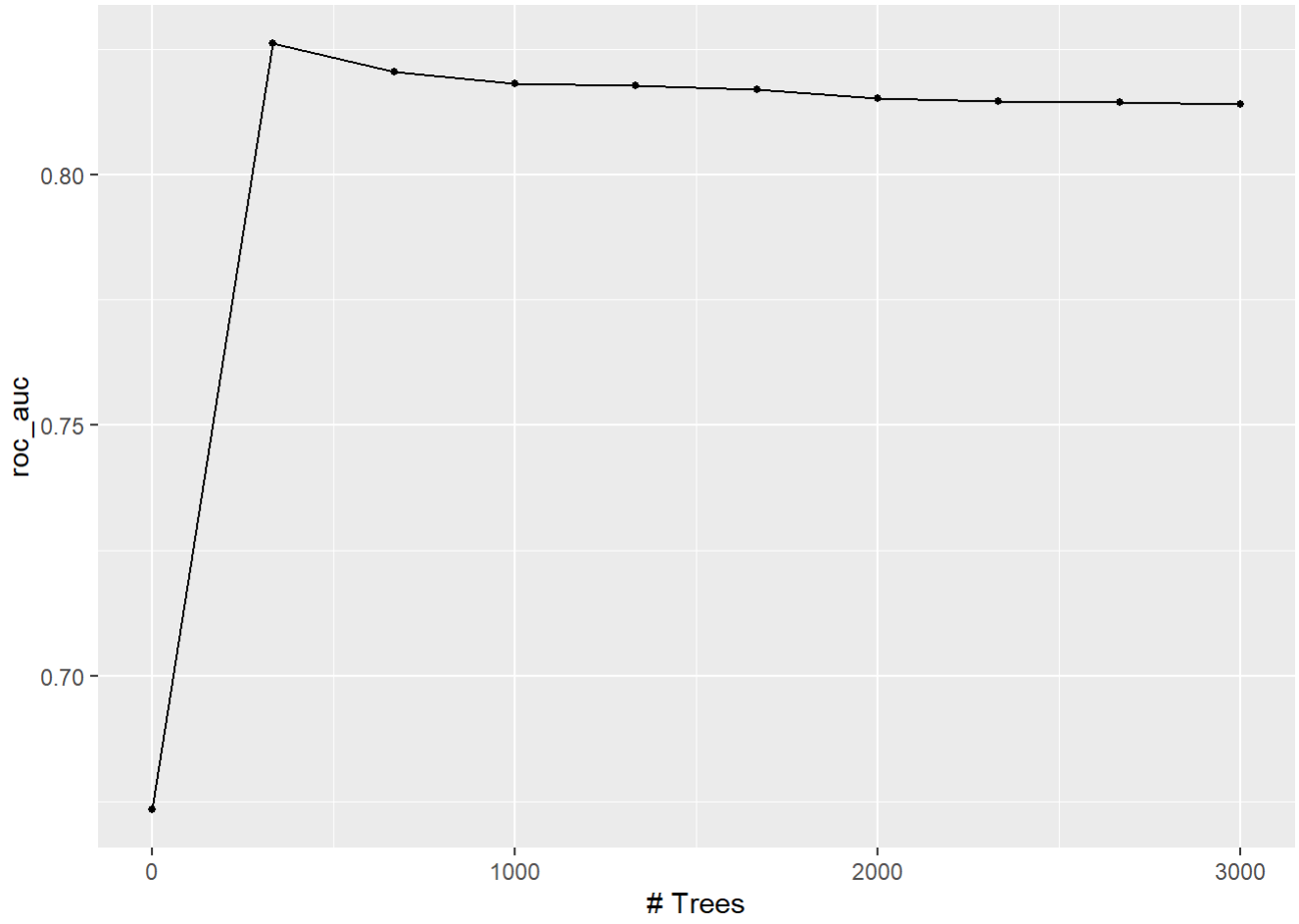


Metric	ROC_AUC	Standard_Error	Cost_Value
roc_auc	0.7778489	0.0165040	4
roc_auc	0.7719104	0.0214722	2
roc_auc	0.7635217	0.0180825	16
roc_auc	0.7577979	0.0155998	32
roc_auc	0.7521751	0.0224258	8
roc_auc	0.7474070	0.0149513	1024

Based on the table, the top-performing SVM model achieved an accuracy of 0.777, with a standard error of 0.0165. Although it performed well, it didn't surpass our best Random Forest model in terms of accuracy. As a result, I will not proceed with this model

Boosted Tree Model

Lastly, we'll employ a Boosted Tree Model. Given the substantial number of observations we have, utilizing this model could offer advantages.



Metric	Value	Standard_Error	trees_Value
roc_auc	0.8260870	0.0229597	334
roc_auc	0.8204441	0.0238726	667
roc_auc	0.8180463	0.0238068	1000
roc_auc	0.8176342	0.0241684	1333
roc_auc	0.8168812	0.0241379	1667
roc_auc	0.8151618	0.0239705	2000

As you can see, our boosted tree's best performing model achieved a ROC_AUC score of 0.826, accompanied by a standard error of 0.0229 and utilizing 334 trees. This is our 2nd best model yet, however it did not surpass our Random Forest model.

Model Selection and Performance

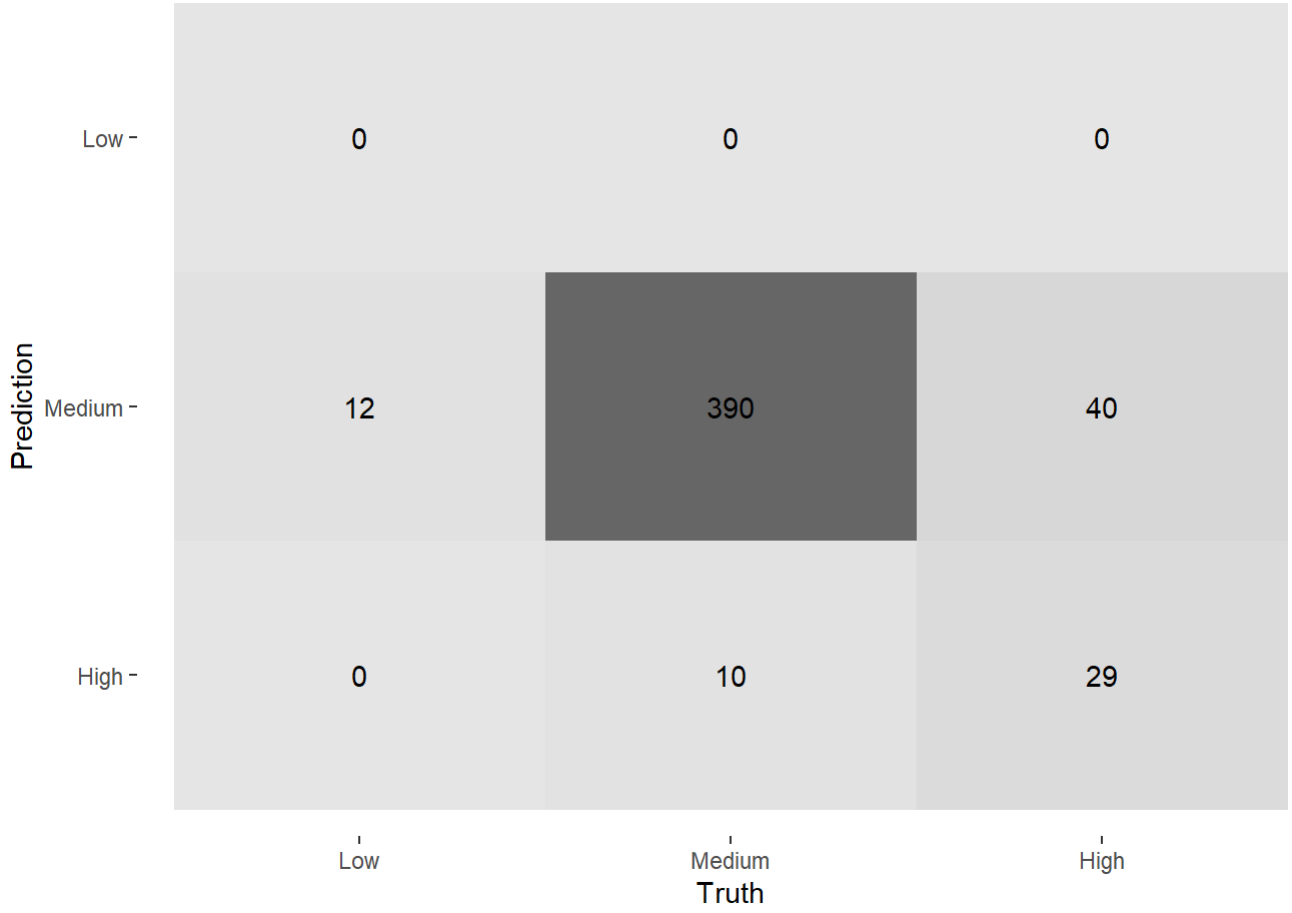
Let us recap the roc_auc of all our models.

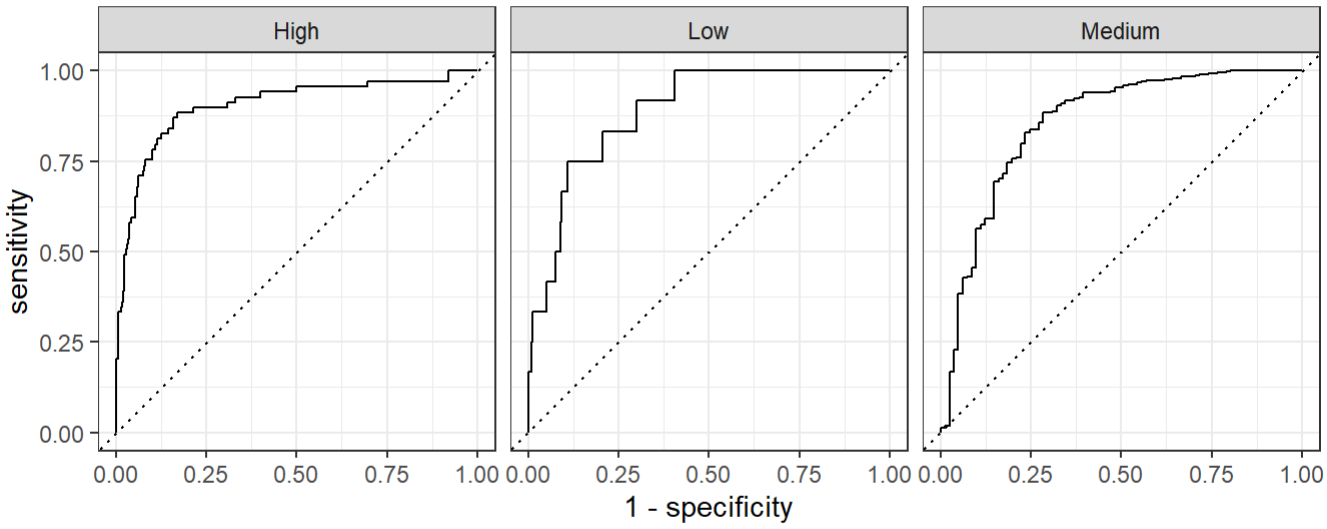
Models	ROC_AUC_Values	Standard_Error
Elastic Net	0.8531677	0.0148602
Random Forest	0.8108850	0.0229597

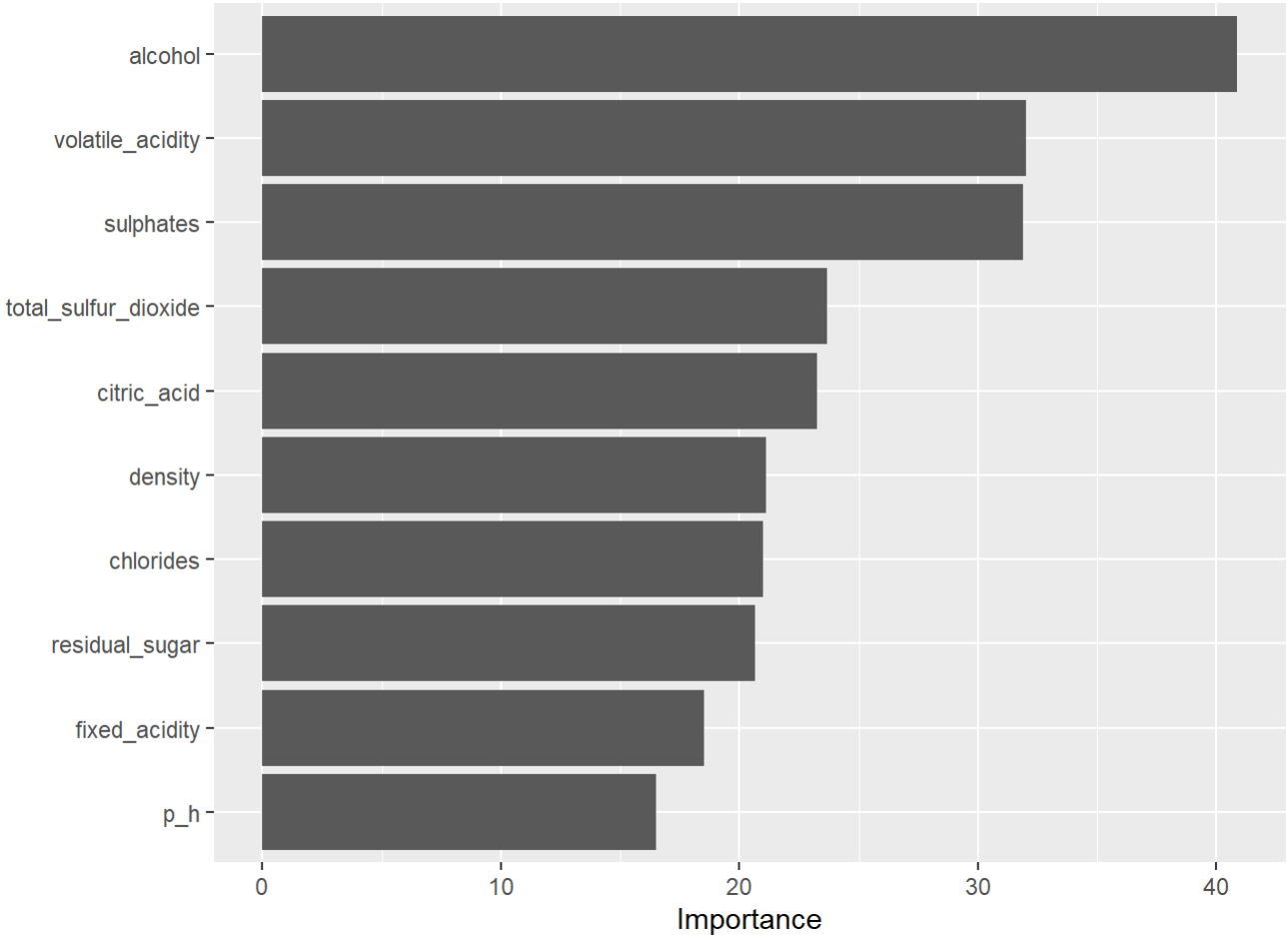
Models	ROC_AUC_Values	Standard_Error
Basic Tree	0.6986038	0.0157270
SVM	0.7778489	0.0165040
Boosted Tree	0.8260870	0.0171716

Evidently, the random forest model the highest roc_auc score and the lowest standard error. Now we will fit this model, then evaluate its performance on the test data..

Metric	Testing_AUC_ROC	Training_ROC_AUC
roc_auc	0.8713527	0.8531677




High
Low
Medium
sensitivity
1 - specificity
0.00
0.25
0.50
0.75
1.00


alcohol
volatile_acidity
sulphates
total_sulfur_dioxide
citric_acid
density
chlorides
residual_sugar
fixed_acidity
p_h
0
10
20
30
40
Importance

The final model exhibited a marginally lower AUC_ROC, but still performed commendably. The confusion matrix indicates a high rate of accurate predictions, particularly for Medium and Low-quality wines. High-quality wines were not as accurately predicted. The shape of all three ROC curves, leaning upwards and to the left, confirms the strong performance of the models. The VIP plot shows that alcohol is among the best predictors to use.

Since the final model achieved an ROC_AUC of 0.867, indicating that it ranks a randomly chosen positive instance higher than a random negative instance 86.7% of the time

Conclusion

All in all, I'm confident that my model did a good job. I evaluated multiple models and ultimately settled on a random forest model, which yielded a roc_auc score of 0.853. This model was configured with an mtry value of 3, 2000 trees, and a min_n value of 10

Appendix

```
library(tidymodels)
library(rpart.plot)
library(dplyr)
library(ggplot2)
library(kernlab)
library(tidyverse)
library(corrplot)
library(corrr)
library(glmnet)
library(xgboost)
library(dials)
library(ranger)
library(janitor)
library(MASS)
library(knitr)

# === Data Preprocessing ===

# Load the dataset from a CSV file and clean the column names
data <- read.csv('C:/Users/omerr/OneDrive/Desktop/winequality-red.csv', sep = ";")
data <- clean_names(data)

# Create a new categorical variable based on wine quality ratings
data$QualityType <- with(data, ifelse(data$quality > 6, 'High',
                               ifelse(data$quality > 4, 'Medium',
                               ifelse(data$quality > 2, 'Low'))))
data$QualityType <- ordered(as.factor(data$QualityType), levels = c('Low','Medium','High'))

# Remove the original 'quality' column
data <- dplyr::select(data, -quality)


# === Exploratory Data Analysis ===

# Compute the correlation matrix
cor_matrix <- cor(data %>% select_if(is.numeric), use ="pairwise.complete.obs")

# Visualize the correlation matrix
corrplot(cor_matrix)

# Generate summary statistics for numerical variables
summary(data %>% dplyr::select(-QualityType))

# Create histograms
par(mfrow = c(1,3))

hist(data$sulphates, prob = TRUE)
lines(density(data$sulphates), lwd = 2)
```

```
hist(data$total_sulfur_dioxide, prob = TRUE)
lines(density(data$total_sulfur_dioxide), lwd = 2)

hist(data$free_sulfur_dioxide, prob = TRUE)
lines(density(data$free_sulfur_dioxide), lwd = 2)

hist(data$chlorides, prob = TRUE)
lines(density(data$chlorides), lwd = 2)

hist(data$residual_sugar, prob = TRUE)
lines(density(data$residual_sugar), lwd = 2)


# === Data Splitting and Cross-Validation ===

# Initialize the random seed for reproducibility
set.seed(111)

# Split the data into training and testing sets
split <- initial_split(data, strata = QualityType, prop = 0.7)
training <- training(split)
testing <- testing(split)

# Set up 10-fold cross-validation on the training data
training_fold <- vfold_cv(training, v = 10, strata = QualityType)

# Define a recipe for preprocessing
recipe <- recipe(QualityType ~ ., data = training) %>%
step_dummy(all_nominal_predictors()) %>%
step_scale(residual_sugar, chlorides, free_sulfur_dioxide, total_sulfur_dioxide, sulphates) %>%
step_interact(residual_sugar ~ sulphates)


# === Model Building ===

# ---- Elastic Net Model ----
netmodel <- multinom_reg(penalty = tune(), mixture = tune()) %>%
set_engine('glmnet') %>%
set_mode('classification')

net_wflow <- workflow() %>%
add_recipe(recipe) %>%
add_model(netmodel)

net_grid <- grid_regular(penalty(range = c(-5,5)),mixture(range = c(0,1)), levels = 10)
load(file = 'C:/Users/omerr/OneDrive/Desktop/PSTAT131FINAL/Tuneresults/netmodelresults.rda')
autoplot(net_tune)

#Getting roc_auc metrics
net_metrics <- collect_metrics(net_tune) %>% dplyr::arrange(desc(mean))

 # Create Table
```

```
net_table <- data.frame(
  Metric = net_metrics$.metric,
  Value = net_metrics$mean,
  Standard_Error = net_metrics$std_err,
  Penalty_Value = net_metrics$penalty,
  Mixture_Value = net_metrics$mixture)

kable(head(net_table, caption = 'First Few Rows Of Best Performing Elastic Net Models And Their
Statistics'))

# ---- Random Forest Model ----
rf_model <- rand_forest() %>%
set_engine("ranger", importance = 'impurity') %>%
set_mode("classification")

rf_wflow <- workflow() %>%
add_recipe(recipe) %>%
add_model(rf_model)

rf_grid <- grid_regular(
mtry(range = c(1, 11)),
trees(range = c(1, 2000)),
min_n(range = c(10, 30)),
levels = 5)

load(file = 'C:/Users/omerr/OneDrive/Desktop/PSTAT131FINAL/Tuneresults/rfresults.rda')
autoplot(rf_tune)

#Getting roc_auc metrics
rf_metrics <- collect_metrics(rf_tune) %>% dplyr::arrange(desc(mean))

# Create Table
rf_table <- data.frame(Metric = rf_metrics$.metric, Value = rf_metrics$mean, Standard_Error = rf
_metrics$std_err, mtry_Value = rf_metrics$mtry, trees_Value = rf_metrics$trees, min_n_Value = rf
_metrics$min_n)

kable(head(rf_table, caption = 'First Few Rows Of Best Performing Random Forest Models And Their
Statistics')).

# ---- Basic Decision Tree Model ----
basictree <- decision_tree() %>%
set_engine('rpart') %>%
set_mode('classification')

tree_wflow <- workflow() %>%
add_recipe(recipe) %>%
add_model(basictree)

tree_grid <- grid_regular(cost_complexity(range = c(-3,-1)), levels = 10)
load(file = 'C:/Users/omerr/OneDrive/Desktop/PSTAT131FINAL/Tuneresults/basictreeresults.rda')
autoplot(tree_tune)
```

```
# Metrics
tree_metrics <- collect_metrics(tree_tune) %>% dplyr::arrange(desc(mean))

tree_table <- data.frame(Metric = tree_metrics$.metric, Value = tree_metrics$mean, Standard_Erro
r = tree_metrics$std_err, Cost_Complexity_Value = tree_metrics$cost_complexity )

kable(head(tree_table, caption = 'First Few Rows of Best Performing Tree Models and Their Statis
tics'))

# ---- Support Vector Machine (SVM) Model ----
svm_model <- svm_poly(degree = 1) %>%
set_engine("kernlab", scaled = FALSE) %>%
set_mode("classification")

svm_wflow <- workflow() %>%
add_recipe(recipe) %>%
add_model(svm_model)

svm_grid <- grid_regular(cost(range = c(1, 10)), levels = 10)
load(file = 'C:/Users/omerr/OneDrive/Desktop/PSTAT131FINAL/Tuneresults/svmresults.rda')
autoplot(svm_tune)

# Metrics
svm_metrics <- collect_metrics(svm_tune) %>% dplyr::arrange(desc(mean))

svm_table <- data.frame(Metric = svm_metrics$.metric, ROC_AUC = svm_metrics$mean, Standard_Error
= svm_metrics$std_err, Cost_Value = svm_metrics$cost )

kable(head(svm_table, caption = 'First Few Rows Of Best Performing SVM Models And Their Statisti
cs'))

# ---- Boosted Tree Model ----
boost <- boost_tree(trees = tune()) %>%
set_engine('xgboost') %>%
set_mode('classification')


boost_wflow <- workflow() %>%
add_model(boost) %>%
add_recipe(recipe)

boost_grid <- grid_regular(trees(range = c(1,3000)), levels = 10)

load(file = 'C:/Users/omerr/OneDrive/Desktop/PSTAT131FINAL/Tuneresults/boosttreeresults.rda')

autoplot(boost_tune)

 #Metrics
boost_metrics <- collect_metrics(boost_tune) %>% dplyr::arrange(-mean)

boost_table <- data.frame(Metric = boost_metrics$.metric, Value = boost_metrics$mean, Standard_E
rror = boost_metrics$std_err, trees_Value = boost_metrics$trees )
```

```
kable(head(boost_table, caption = 'First Few Rows of Best Performing Boosted Tree Models and The
ir Statistics'))

# === Model Selection and Performance ===

# Getting the best ROC AUC values and standard errors from previously created tables
net_best <- net_table[1,2]
rf_best <- rf_table[1,2]
tree_best <- tree_table[1,2]
svm_best <- svm_table[1,2]
boost_best <- boost_table[1,2]

# Getting the standard errors
net_se <- net_table[1,3]
rf_se <- rf_table[1,3]
tree_se <- tree_table[1,3]
svm_se <- svm_table[1,3]
boost_se <- boost_table[1,3]

# Combine the best parameters into a single table
models <- c('Elastic Net', 'Random Forest', 'Basic Tree', 'SVM', 'Boosted Tree')
rocaucs <- c(rf_best, net_best, tree_best, svm_best, boost_best)
stderr <- c(rf_se, boost_se, net_se, svm_se, tree_se)
best_table <- data.frame(Models = models, ROC_AUC_Values = rocaucs, Standard_Error = stderr)
kable(best_table)


# Finalizing the best Random Forest model
best_rocaucs <- select_best(rf_tune)
rf_final <- finalize_workflow(rf_wflow, best_rocaucs)

# Fitting the model to the training data
rf_fit <- fit(rf_final, data = training)

# Evaluating the model performance on the test data
results <- augment(rf_fit, testing) %>%
  roc_auc(truth = QualityType, .pred_Low, .pred_Medium, .pred_High)
final_table <- data.frame(Metric = results$.metric,
                          Testing_AUC_ROC = results$.estimate,
                          Training_ROC_AUC = rf_best)
kable(final_table)


# Generating confusion matrix heatmap
conf_mat <- augment(rf_fit, testing)
conf_mat(conf_mat, QualityType, .pred_class) %>% autoplot(type = 'heatmap')

# Plotting the ROC curve
roc_data <- augment(rf_fit, testing)
roc_curve(roc_data, QualityType,.pred_Low, .pred_Medium, .pred_High) %>%
  autoplot()
```

```
# Variable importance plot
vip::vip(rf_fit %>% extract_fit_parsnip())
```