Wine Quality Capstone Project

Nick Bova

26 September, 2020

1 Introduction

This project has been prepared as second part of the HarvardX Data Science Series[1] Capstone Course. For this "Choose Your Own" project, the Wine Quality Data Set from UCI (University of California, Irvine) Machine Learning Repository is being used.[2] The dataset is related to white vinho verde wine samples from the north of Portugal. The dataset contains physicochemical tests for an individual wine and an associated quality score.

Wine is an interesting beverage that is enjoyed in many parts of the world and can display a wide range of styles and flavors. Wine is produced from 2 basic ingredients, grapes and yeast, through the process of fermentation. There can be small additions of acids or sulfates used to control wild yeast or to balance and stabilize the product. "No matter what it costs, almost 98 per cent of most wines is made up of water and ethanol. The remaining 2 per cent is a combination of acids, sugars, volatile flavour and aroma compounds, pigment compounds and tannins. Amazingly, it's this 2 per cent that makes all the difference, giving a wine its unique flavour, colour, aroma and individuality. It's here, in this 2 per cent, that chemistry really comes into play."[3]

The goal of this project is to use the white wine dataset to explore the impact of 11 physicochemical features on wine quality. The 11 features contained in this dataset are only a small subset of the potential physicochemical features that impact wine quality. It is known that "The basic flavour of a wine is formed from the balance of sugars, acids, phenolics, and ethanol, but the character of the wine is provided by the volatile aroma compounds. Over 1000 of these volatiles have been identified in wines from around the world."[4] Can wine quality be predicted using just the 11 features contained in this dataset? How good are the predictions and where are the errors?

The project will include both supervised an unsupervised analysis. The goal of the supervised analysis will be to predict the quality score based on the physicochemical features. The goal of the unsupervised analysis will consist of reviewing the key components driving the model and discussing how winemakers might leverage the information to improve quality when making or blending wines.

The overall project method of approach was to:

1. Load and verify the data
2. Analyze the data
3. Standardize the data

[1] https://www.edx.org/professional-certificate/harvardx-data-science

[2] P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009. https://archive.ics.uci.edu/ml/datasets/Wine+Quality

[3] Australian Academy of Science. 2017. The chemistry of wine: Part 1. https://www.science.org.au/curious/earth-environment/chemistry-wine-part-1

[4] https://archive.ics.uci.edu/ml/datasets/Wine+Quality

4. Unsupervised model and principal component analysis
5. Evaluate prediction models
    - Numeric - regression models
    - Category - classification models
6. Estimate the impact of prevalence
7. Evaluate the models and develop a set of conclusions
8. Document the results

The modeling and R code are key components of this report. As such, the code will typically be shown in the modeling sections for clarity. In the data visualization sections the code will generally not be shown so that the analysis can flow smoother. When the code is not shown in the PDF document, it is still available in the Rmd document and in the code script.

2 Data Exploration and Analysis

2.1 Setup

This section sets the global options for code chunks, loads the libraries, and turns off scientific notation. All libraries used are loaded at this step. The setup code is shown below.

```
#set global options for code chunks to no messages or warnings
knitr::opts_chunk$set(
    message = FALSE,
    warning = FALSE,
    echo = TRUE
)

#Load the libraries
library(tidyverse)
library(caret)
library(knitr)
library(kableExtra)
library(rpart)
library(rpart.plot)
library(AppliedPredictiveModeling)
library(corrplot)
library(randomForest)
library(rpart)
library(FactoMineR)
library(factoextra)
library(broom)
library(corrplot)

#turn off scientific notation and set number of digits to print
options(scipen = 999, digits = 5)
```

2.2 Data Load

The wine quality data set was provided as a delimited data set with a semicolon used as the delimiter. There was also a red data set included, but the scope of this project will be limited to the analysis of the white wine data set only.

During the data load step, the column names were cleaned up to eliminate spaces and replace them with an underscore character.

```
# read the dataset for white wine
white <- read_delim("winequality-white.csv", delim = ";")
# get rid of the spaces in the column names
colnames(white)<- colnames(white) %>% str_replace_all(" ", "_")
```

2.3 Data Verification

During the data verification step, we will check the data for any apparent errors or problems. First, let's look at the structure of the data.

```
#display the data structure
str(white)
```

```
## tibble [4,898 x 12] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ fixed_acidity       : num [1:4898] 7 6.3 8.1 7.2 7.2 8.1 6.2 7 6.3 8.1 ...
##  $ volatile_acidity    : num [1:4898] 0.27 0.3 0.28 0.23 0.23 0.28 0.32 0.27 0.3 0.22 ...
##  $ citric_acid         : num [1:4898] 0.36 0.34 0.4 0.32 0.32 0.4 0.16 0.36 0.34 0.43 ...
##  $ residual_sugar      : num [1:4898] 20.7 1.6 6.9 8.5 8.5 6.9 7 20.7 1.6 1.5 ...
##  $ chlorides           : num [1:4898] 0.045 0.049 0.05 0.058 0.058 0.05 0.045 0.045 0.049 0.044 ...
##  $ free_sulfur_dioxide : num [1:4898] 45 14 30 47 47 30 30 45 14 28 ...
##  $ total_sulfur_dioxide: num [1:4898] 170 132 97 186 186 97 136 170 132 129 ...
##  $ density             : num [1:4898] 1.001 0.994 0.995 0.996 0.996 ...
##  $ pH                  : num [1:4898] 3 3.3 3.26 3.19 3.19 3.26 3.18 3 3.3 3.22 ...
##  $ sulphates           : num [1:4898] 0.45 0.49 0.44 0.4 0.4 0.44 0.47 0.45 0.49 0.45 ...
##  $ alcohol             : num [1:4898] 8.8 9.5 10.1 9.9 9.9 10.1 9.6 8.8 9.5 11 ...
##  $ quality             : num [1:4898] 6 6 6 6 6 6 6 6 6 6 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   `fixed acidity` = col_double(),
##   ..   `volatile acidity` = col_double(),
##   ..   `citric acid` = col_double(),
##   ..   `residual sugar` = col_double(),
##   ..   chlorides = col_double(),
##   ..   `free sulfur dioxide` = col_double(),
##   ..   `total sulfur dioxide` = col_double(),
##   ..   density = col_double(),
##   ..   pH = col_double(),
##   ..   sulphates = col_double(),
##   ..   alcohol = col_double(),
##   ..   quality = col_double()
##   .. )
```

A look for NA values shows that there are none. The code below is returning a zero value.

```r
#get index of any NA values
which(is.na(white))
```

```
## integer(0)
```

Let's look at the summary statistics for the white data set.

```r
# display summary statistics
summary(white)
```

```
##  fixed_acidity   volatile_acidity  citric_acid    residual_sugar
##  Min.   : 3.80   Min.   :0.080    Min.   :0.000   Min.   : 0.60
##  1st Qu.: 6.30   1st Qu.:0.210    1st Qu.:0.270   1st Qu.: 1.70
##  Median : 6.80   Median :0.260    Median :0.320   Median : 5.20
##  Mean   : 6.85   Mean   :0.278    Mean   :0.334   Mean   : 6.39
##  3rd Qu.: 7.30   3rd Qu.:0.320    3rd Qu.:0.390   3rd Qu.: 9.90
##  Max.   :14.20   Max.   :1.100    Max.   :1.660   Max.   :65.80
##    chlorides      free_sulfur_dioxide total_sulfur_dioxide    density
##  Min.   :0.0090   Min.   :  2.0       Min.   :  9          Min.   :0.987
##  1st Qu.:0.0360   1st Qu.: 23.0       1st Qu.:108          1st Qu.:0.992
##  Median :0.0430   Median : 34.0       Median :134          Median :0.994
##  Mean   :0.0458   Mean   : 35.3       Mean   :138          Mean   :0.994
##  3rd Qu.:0.0500   3rd Qu.: 46.0       3rd Qu.:167          3rd Qu.:0.996
##  Max.   :0.3460   Max.   :289.0       Max.   :440          Max.   :1.039
##        pH         sulphates       alcohol        quality
##  Min.   :2.72   Min.   :0.22   Min.   : 8.0   Min.   :3.00
##  1st Qu.:3.09   1st Qu.:0.41   1st Qu.: 9.5   1st Qu.:5.00
##  Median :3.18   Median :0.47   Median :10.4   Median :6.00
##  Mean   :3.19   Mean   :0.49   Mean   :10.5   Mean   :5.88
##  3rd Qu.:3.28   3rd Qu.:0.55   3rd Qu.:11.4   3rd Qu.:6.00
##  Max.   :3.82   Max.   :1.08   Max.   :14.2   Max.   :9.00
```

The white wine data set has 4898 obs of 12 variables. All variables are numeric. According to the documentation provided with the data set, these characteristics are correct.[5]

The quality ratings for white wine ranged from 3 to 9, with a mean of 5.88, a median of 6.

2.3.1 Verification Results

The data set appears to be clean and usable. The data is already in tidy format. No missing values or bad quality ratings were detected.

2.4 Separate Test and Training Data

Now the data will be separated into testing and training data sets. The test set will not be used for any further analysis or model training purposes.

In addition to this split, we will also be creating a test and train data set to use with our classification models. The original data is all numeric. We will also create a categorical version of the quality results to

[5] https://archive.ics.uci.edu/ml/datasets/Wine+Quality

use with classification models. In order to do this, only the quality variable, which is an integer from 3 to 9, will be converted to a factor.

The code below performs the initial split into the test and train sets. 20% of the observations are being reserved for testing.

```
# set the seed for repeatability
set.seed(831, sample.kind = "Rounding")

#create train and test sets for white wine
test_index <- createDataPartition(white$quality, times = 1, p = 0.2, list = FALSE)
train_rating <- white %>% slice(-test_index)
test_rating <- white %>% slice(test_index)
```

A second test and train data set with quality changed to a factor is created below. These data sets are identical to the original data sets with the exception of quality now being a factor instead of a numeric variable.

```
# Create a second train and test set with categories instead of ratings
# The categories are acceptable, good, premium
# These are the same data sets as the rating sets, but with the quality factor
# Changed to a category
test_category <- test_rating %>%
  mutate(quality = as.factor(
    ifelse(quality >=8, "premium",
           ifelse(quality <= 4, "acceptable", "good") )))

train_category <- train_rating %>%
  mutate(quality = as.factor(
    ifelse(quality >=8, "premium",
           ifelse(quality <= 4, "acceptable", "good") )))

test_category <- test_rating %>%
  mutate(quality = as.factor(quality))

train_category <- train_rating %>%
  mutate(quality = as.factor(quality))
```

2.5 Correlation matrix of variables

In order to further understand how the features and outcomes might be related, let's look at the correlation matrix. The correlation matrix between all variables are shown below along with a plot of the correlation matrix.

The outcome, quality, is most strongly correlated with alcohol and negatively correlated with density, showing correlations of 0.4536 and -0.3042. Density and alcohol are also strongly correlated to each other at -0.773. The fact that density is correlated to alcohol is not surprising. When fermenting beer or wine, density (or specific gravity) is used to measure the conversion of sugar to alcohol. Sugar is dense and raises the specific gravity of the wine. Alcohol is less dense than water and decreases the specific gravity of the wine as it is produced during the fermentation process. It is not surprising that density, alcohol and residual sugar are all highly correlated.

If the data set were extremely large, we might consider eliminating one or more of these features to achieve some dimension reduction. Due to the relatively limited number of features (11) contained in this dataset and the small number of observations, we will continue to retain all of the features.

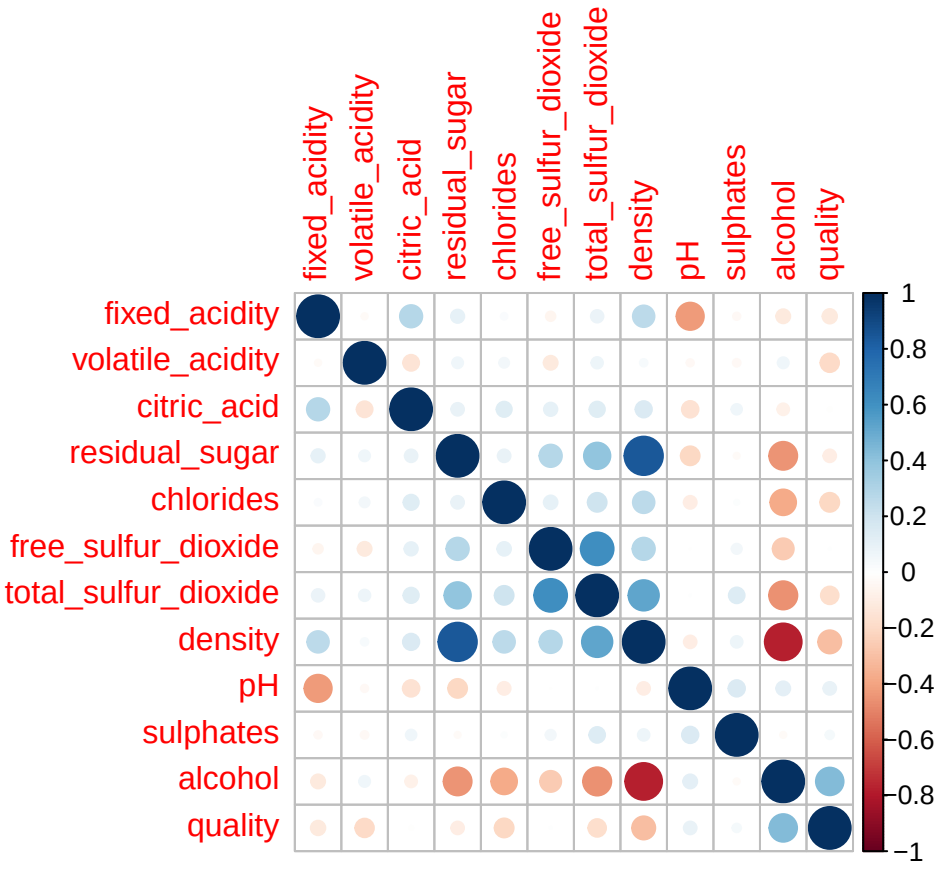

Figure 1: Correlation Plot

Table 1: Correlation Matrix

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality
fixed_acidity	1.00000	-0.02278	0.28460	0.10032	0.02592	-0.05139	0.08982	0.26551	-0.42349	-0.03077	-0.11163	-0.11959
volatile_acidity	-0.02278	1.00000	-0.14448	0.06819	0.05649	-0.11083	0.07520	0.03282	-0.03080	-0.03441	0.06953	-0.19589
citric_acid	0.28460	-0.14448	1.00000	0.09238	0.13209	0.10214	0.13441	0.15204	-0.15716	0.06396	-0.07952	-0.00884
residual_sugar	0.10032	0.06819	0.09238	1.00000	0.09321	0.28690	0.39886	0.84109	-0.20265	-0.02113	-0.44607	-0.09284
chlorides	0.02592	0.05649	0.13209	0.09321	1.00000	0.10706	0.20302	0.26644	-0.09517	0.01540	-0.37656	-0.20504
free_sulfur_dioxide	-0.05139	-0.11083	0.10214	0.28690	0.10706	1.00000	0.61405	0.28580	-0.00036	0.05936	-0.25248	0.00236
total_sulfur_dioxide	0.08982	0.07520	0.13441	0.39886	0.20302	0.61405	1.00000	0.52753	0.00124	0.14147	-0.45393	-0.17788
density	0.26551	0.03282	0.15204	0.84109	0.26644	0.28580	0.52753	1.00000	-0.09182	0.07756	-0.77323	-0.30418
pH	-0.42349	-0.03080	-0.15716	-0.20265	-0.09517	-0.00036	0.00124	-0.09182	1.00000	0.15408	0.11131	0.09314
sulphates	-0.03077	-0.03441	0.06396	-0.02113	0.01540	0.05936	0.14147	0.07756	0.15408	1.00000	-0.02212	0.04213
alcohol	-0.11163	0.06953	-0.07952	-0.44607	-0.37656	-0.25248	-0.45393	-0.77323	0.11131	-0.02212	1.00000	0.43556
quality	-0.11959	-0.19589	-0.00884	-0.09284	-0.20504	0.00236	-0.17788	-0.30418	0.09314	0.04213	0.43556	1.00000

2.6 Data Visualization

The quality variable is our outcome or dependent variable in this project. We are trying to predict quality based on the other 11 features. The features are based on lab analysis. The quality variable is based on scoring by human tasters. Each row in the dataset is for an individual wine. Each wine only has one entry and one quality rating.

Since quality is the outcome that we are trying to predict, let's take a look at that first. Figure 2 shows a histogram of the quality ratings in the training data set. The histogram shows a strong central tendency for the data. High and low ratings are relatively rare. This situation could create prevalence problems if there is overlap in the signal associated with the features. Machine learning models will typically error on the side of the more prevalent category in order to maximize accuracy.

2.6.1 Check for linear relationships

Linear relationships between the features and the predicted quality variable, might be useful in modeling. This section plots each feature vs quality in a facet grid (Figure 3). Most of the trend lines look fairly flat with a broad variation around them.

In order to see if there is any significance, the p value for each factor was calculated. Factors that are significant at a 95% confidence level will have a p value below 0.05. The factors showing significant p values are:

```
##        fixed_acidity     volatile_acidity       residual_sugar
##               0.0000               0.0000               0.0000
##            chlorides total_sulfur_dioxide              density
##               0.0000               0.0000               0.0000
##                   pH            sulphates              alcohol
##               0.0000               0.0084               0.0000
```

All values were significant except the 2 shown below. The lack of linear correlation does not mean that citric acid and free sulfur dioxide will not be valuable predictors. It simply implies that there is no linear trend.

```
##         citric_acid free_sulfur_dioxide
##              0.5800              0.8827
```

2.6.2 Boxplots of Features vs Quality

Boxplots provide another view of the relationship between features and quality. In order to better visualize this, the quality ratings were grouped into 3 categories as follows:

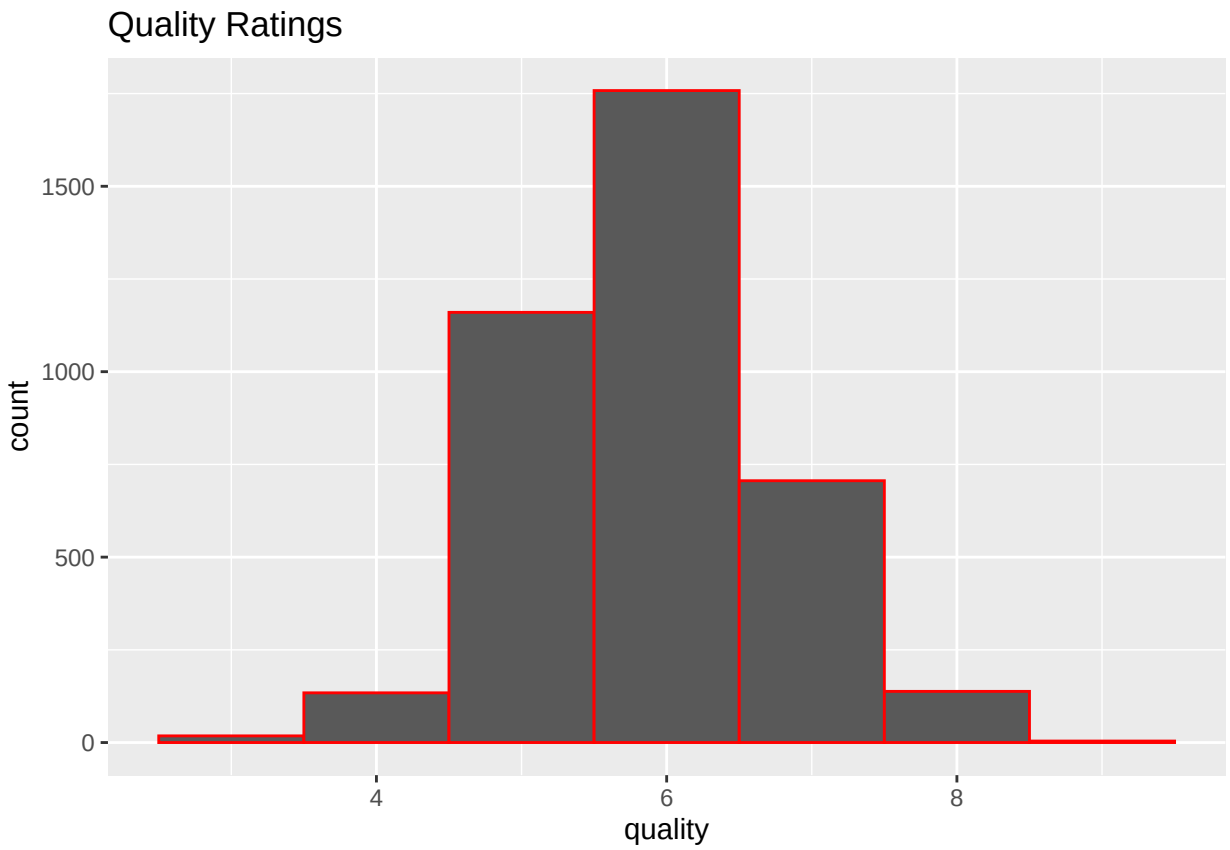

Figure 2: Quality Rating Histogram

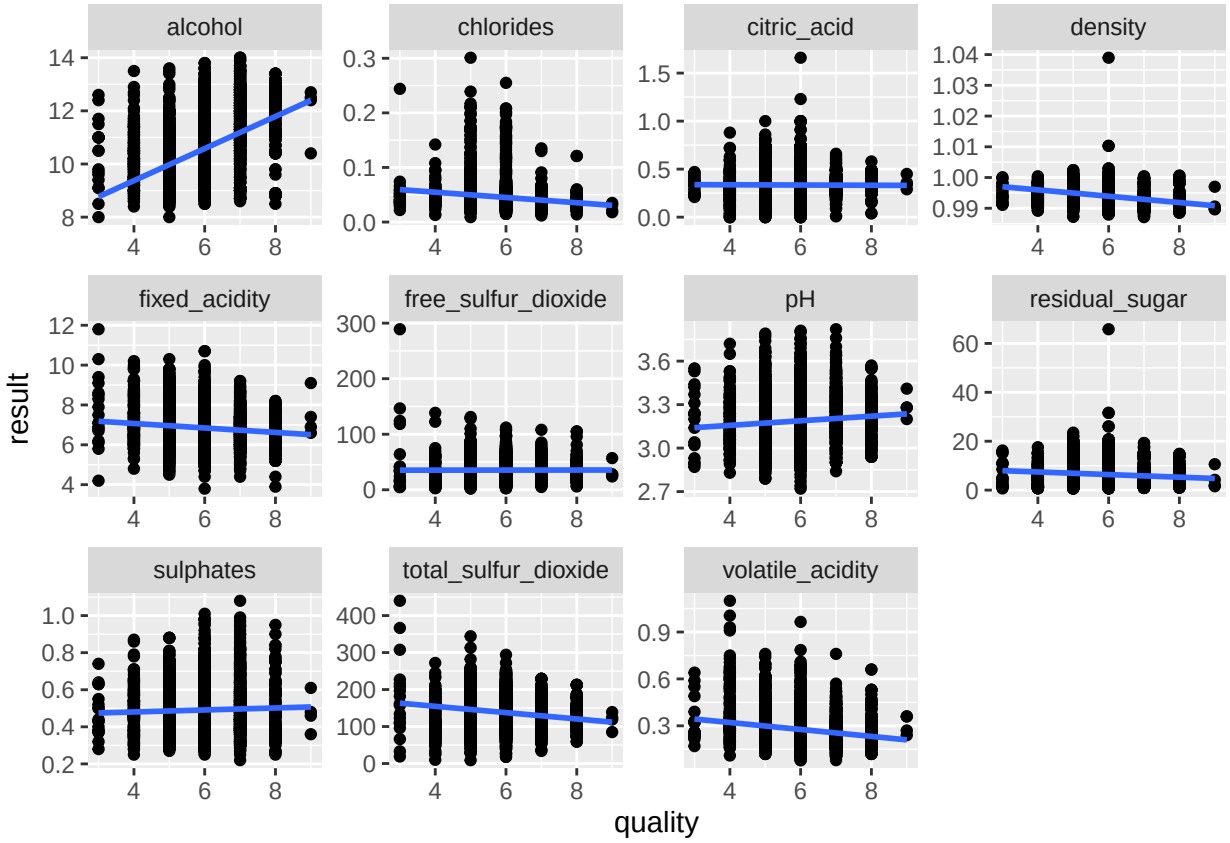

Figure 3: Features Test Results

- Acceptable - quality ratings 3 and 4
- Good - quality ratings 5, 6, and 7
- Premium - quality ratings 8 and 9

The reduction in categories allows us to see the information with less clutter. The boxplots are shown in Figure 4.

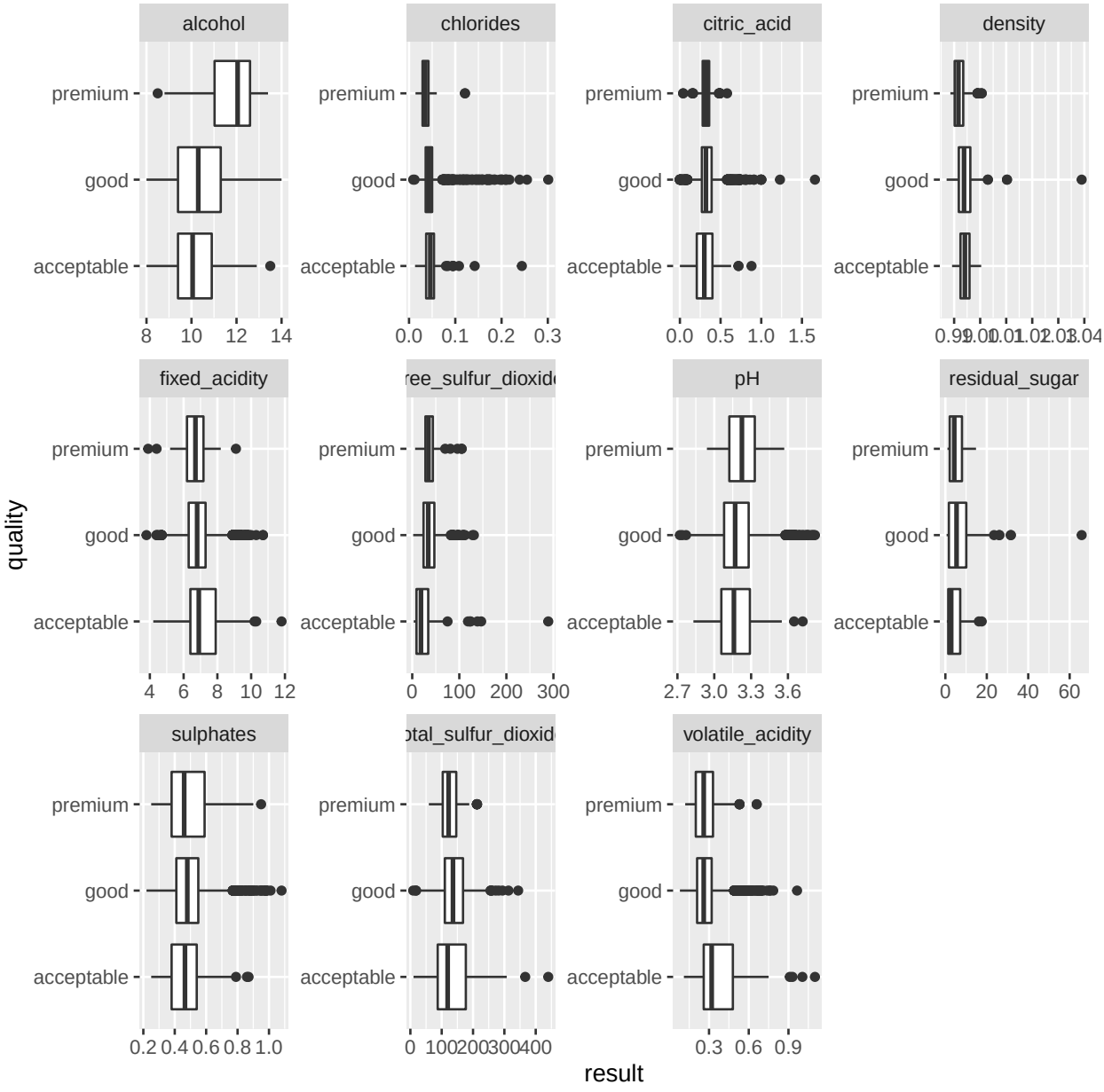

Figure 4: Test Result vs Quality

2.6.3 Scatterplot Matrix

A pairwise comparison of features vs quality shows some interesting results. We used the acceptable, good, premium ratings that were introduced above. A pairwise view of all 11 features cannot reasonably fit on a page, so the boxplots were split into two groups and are displayed in Figures 5 & 6. Although this does not provide a full feature to feature comparison, all plots show the same pattern. As can be seen in the 2 plots, the quality ratings seem to cluster around the premium ratings which are a target in the center. The acceptable ratings tend to be the outliers. This may make prediction difficult as most models seem to use separation between clusters. Finding a model that works well for concentric clusters may be difficult.

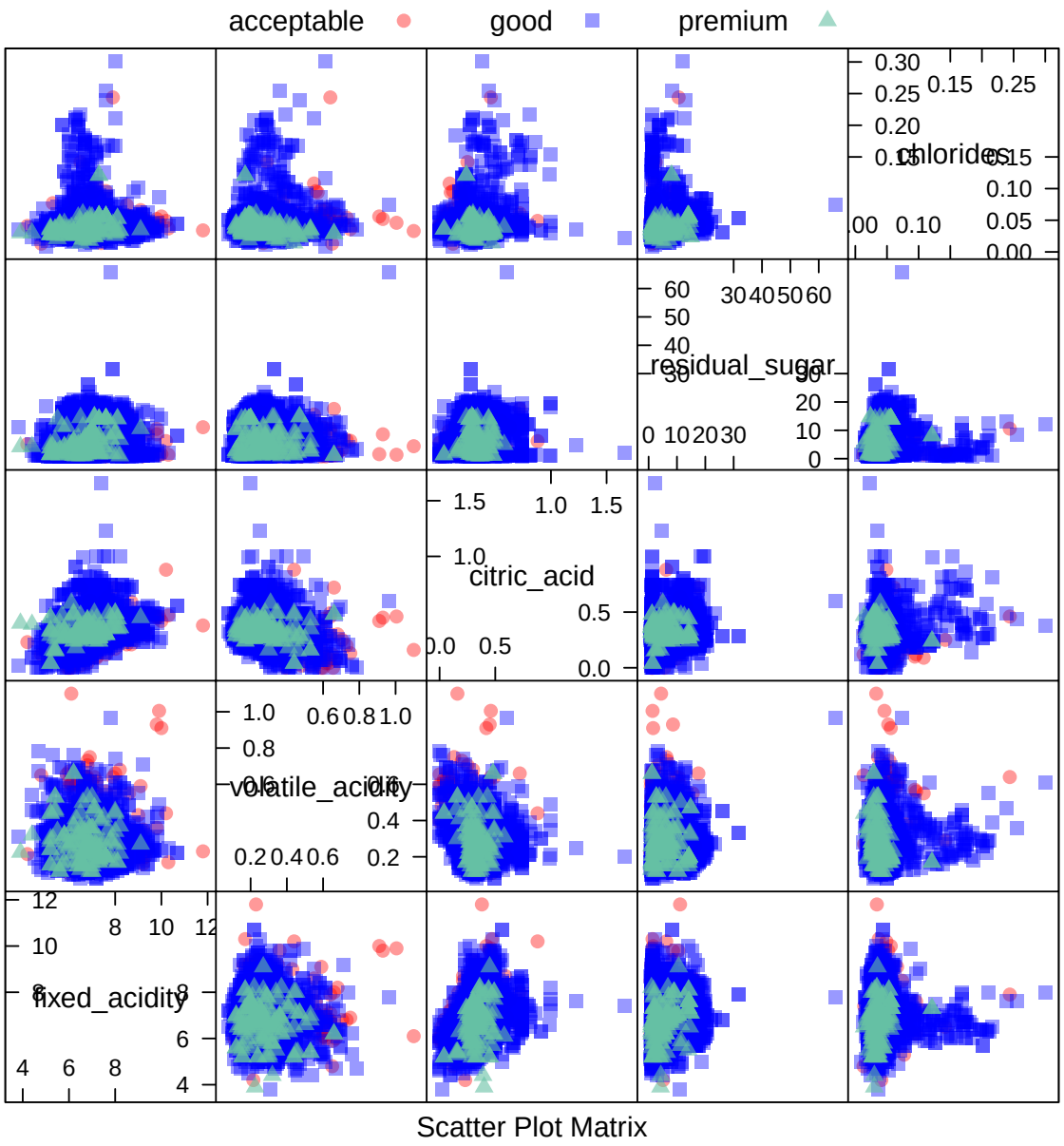

Figure 5: Features Pairwise Comparison 1

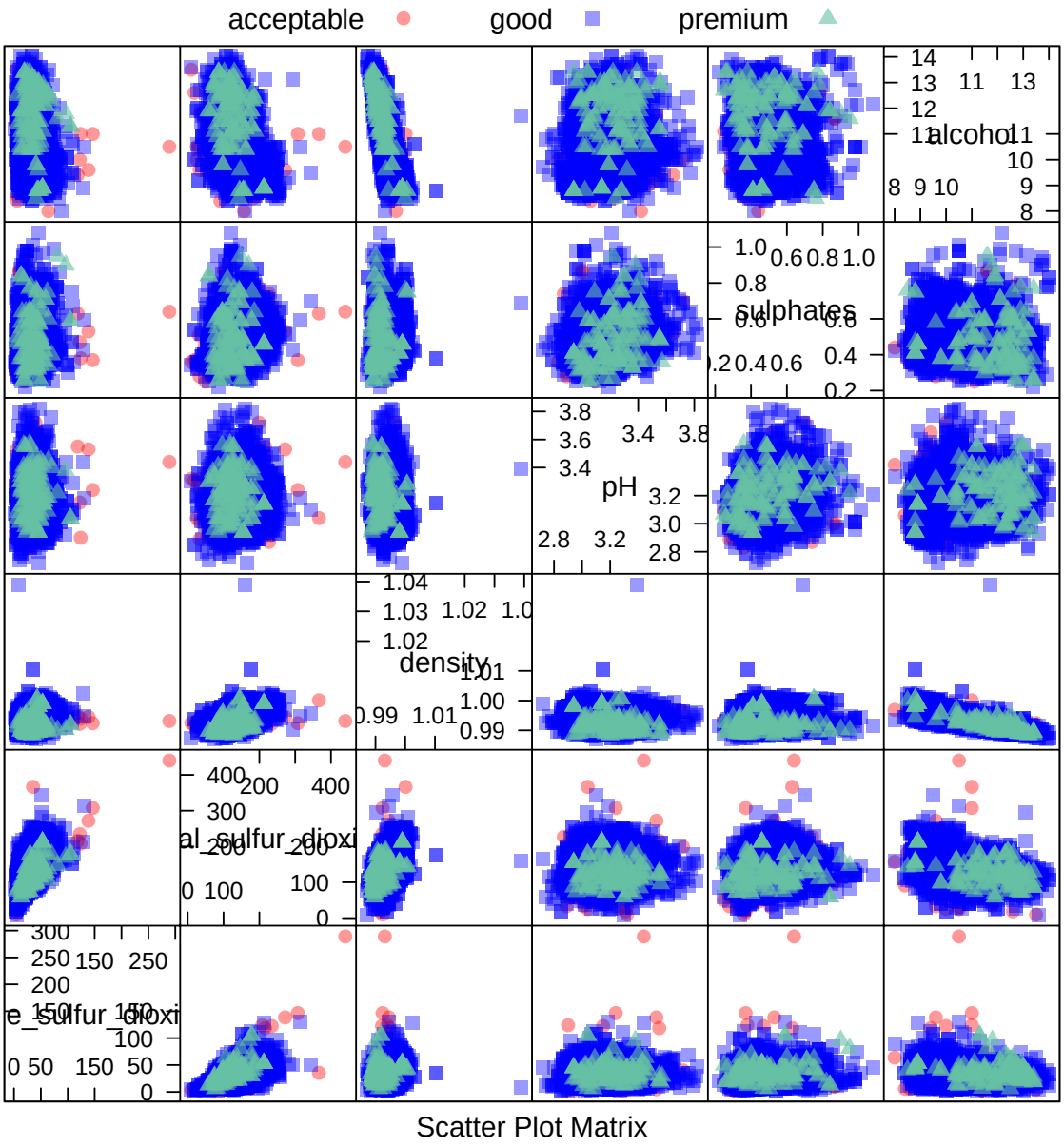

Figure 6: Features Pairwise Comparison 2

2.7 Standardize the features

Up to this point, the analysis has been conducted on the raw data as provided. The features have widely varying measurements and scales. A few test runs with a subset of models showed that many performed significantly better on centered and standardized data, and the other models showed no impact. With this in mind, we are going to use a standardized features data set from this point forward.

In order to standardize the data the preProcess function from the caret package will be used. We are also going to create a single set of features for both numeric and categorical analysis. The quality predictor, y_num or y_cat, will be separate for numeric (regression) and categorical (classification) models.

```
# using caret
# Create the subset of features
# features <- train_rating[ , -12]
y_num <- train_rating[ , 12]
y_cat <- train_category[ , 12]
y_test_num <- test_rating[ , 12]
y_test_cat <- test_category[ , 12]

#obtain the preProcessValues
# there is only one set of features, so only 1 preProcValues object is needed.
preProcValues <- preProcess(train_rating[,-12], method = c("center", "scale"))

# calculate the features for the training set
train_feat <- predict(preProcValues, train_rating[ , -12])
# the test set is transformed using the mean and sd from the train set
# that are stored in the preProcValues List
test_feat <- predict(preProcValues, test_rating[ , -12])
```

As shown below, all features in the transformed train set have a mean of zero and a standard deviation of 1. As expected the mean and standard deviation for the test set are slightly different. This is because the preProcessValues were built using only the train set.

```
## Mean and Standard Deviations for the training set:

##      fixed_acidity volatile_acidity citric_acid residual_sugar chlorides
## [1,]             0                0           0              0         0
## [2,]             1                1           1              1         1
##      free_sulfur_dioxide total_sulfur_dioxide density pH sulphates alcohol
## [1,]                   0                    0       0  0         0       0
## [2,]                   1                    1       1  1         1       1

##
##
## Mean and Standard Deviations for the test set:

##      fixed_acidity volatile_acidity citric_acid residual_sugar chlorides
## [1,]        -0.036           -0.056      -0.016         -0.038     0.036
## [2,]         1.070            1.003       0.993          0.974     1.217
##      free_sulfur_dioxide total_sulfur_dioxide density    pH sulphates alcohol
## [1,]              -0.036               -0.033  -0.033 0.060    -0.018   0.033
## [2,]               0.917                0.973   0.966 1.002     1.015   0.992
```

2.8 Decision Trees for Data Mining

One of the stated goals for this project is to determine which features might be useful to wine makers during the wine making or blending processes. Decision trees provide an excellent tool for this purpose.

> Of all the well-known learning methods, decision trees come closest to meeting the requirements for serving as an off-the-shelf procedure for data mining. They are relatively fast to construct and they produce interpretable models (if the trees are small). [@Hastie, Trevor, Robert Tibshirani, Jerome Friedman. 2017. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Second Edition. pg 352]

First we will take a look at the decision tree based on predicting the numerical rating of a wine (Figure 7). Please note that this tree is based on standardized data. If we were using this for decision making, we would want to reverse the standardization process.

```
#create the training and test sets from the standardized features
train <- cbind(train_feat, y_num)
test <- cbind(test_feat, y_test_num )
# train the rpart model
train_rpart2 <- train(quality ~ .,
                      method = "rpart",
                      tuneGrid = data.frame(cp = seq(0.0,0.1, len=25)),
                      data = train)
# check the rmse of the model
y_hat <- predict(train_rpart2, test)
rmse_rpart_tuned <- RMSE(test$quality, y_hat)
#display the rmse results
cat("The RMSE for the model produced by the decision tree is", rmse_rpart_tuned )
```

```
## The RMSE for the model produced by the decision tree is 0.72519
```

```
# predictors from the rpart model
rpart.plot(train_rpart2$finalModel,
           main = "Decision Tree\nStandardized Features",
           type = 5,
           yesno = 2,
           cex = .7)
```

The tree is helpful to visualize how values are assigned by the model. There are a number of branches with same feature used on multiple branches as the decision criterion, such as alcohol and volatile_acidity. As shown below, there are 7 unique features that appear on the decision tree.

```
## [1] "alcohol"             "volatile_acidity"     "free_sulfur_dioxide"
## [4] "pH"                  "total_sulfur_dioxide" "fixed_acidity"
## [7] "residual_sugar"
```

The rpart model also produces a variable importance metric. This metric is "An overall measure of variable importance is the sum of the goodness of split measures for each split for which it was the primary variable, plus goodness * (adjusted agreement) for all splits in which it was a surrogate."[6]

It is interesting to note that both chlorides and citric acid have a high variable importance, but are not criteria that actually appear on the decision tree. The variable importance is shown below.

[6]Therneau, Terry, Elizabeth Atkinson. 2019. An Introduction to Recursive Partitioning Using the RPART Routines. p12. https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf

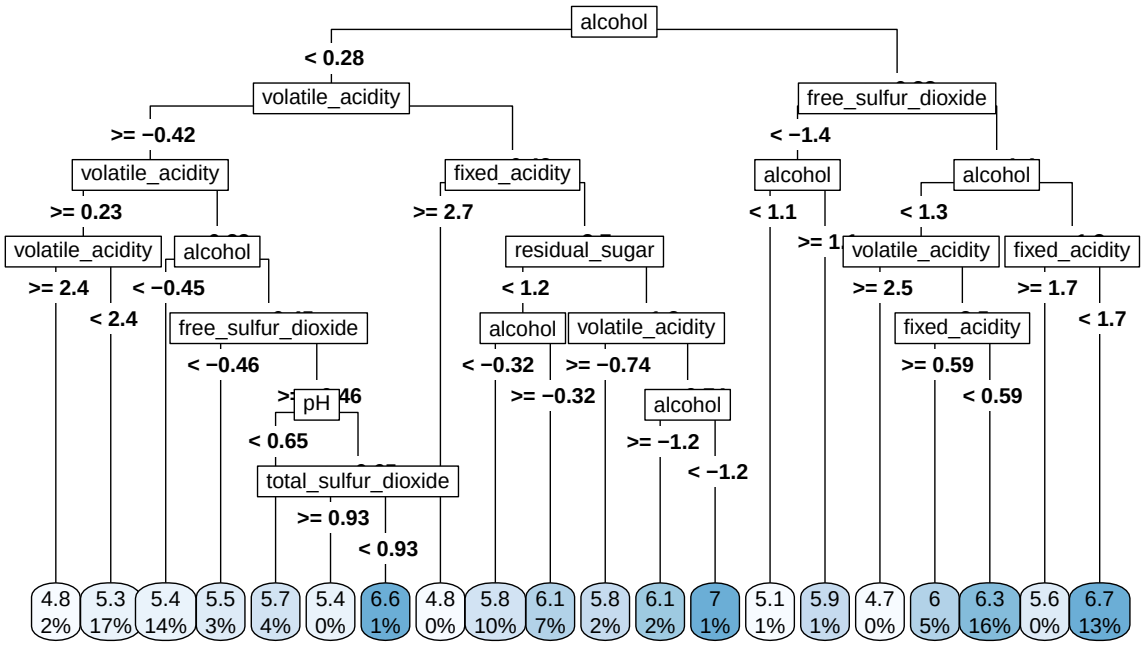

Figure 7: Decision Tree for Numeric Ratings

```r
#variable importance
vi <- varImp(train_rpart2)$importance %>% arrange(desc(Overall))
#save the variable importance for future comparison to other unsupervised models
vi_table <- as_tibble_col(rownames(vi)[1:4], column_name = "rating")
#display the variable importance
vi
```

```
##                       Overall
## alcohol              100.0000
## chlorides             65.4288
## citric_acid           51.7975
## total_sulfur_dioxide  46.7874
## volatile_acidity      40.6349
## pH                    39.5504
## free_sulfur_dioxide   38.9010
## residual_sugar        32.9472
## fixed_acidity         11.1699
## density                2.8704
## sulphates              0.0000
```

2.8.1 Decision Tree for Premium Category

Next we will look at the decision tree for only two categories; our premium category and everything else (Figure 8). If a winemaker is targeting top quality, this should provide some insight. When focusing on premium quality, the tree produced is quite different than the one shown above, where the focus was correctly picking ratings across the full spectrum of ratings.

```r
#set up the categories as premium or other
y_prem <- y_num %>% mutate(quality = ifelse(quality < 7, "other", "premium"))
#create the training data set
train <- cbind(train_feat, y_prem)
#train the model for the category analysis
train_rpart2 <- train(quality ~ .,
                      method = "rpart",
                      tuneGrid = data.frame(cp = seq(0.0,0.1, len=25)),
                      data = train)

# prune the tree to reduce the number of branches
pruned <- prune(train_rpart2$finalModel, cp = 0.01)

#plot the tree
rpart.plot(pruned,
           main = "Decision Tree - Premium Category\nStandardized Features",
           type = 5,
           yesno = 2,
           cex = .7)
```

This decision tree looks quite different from the tree for the numeric ratings. In fact, it had to be pruned to reduce the number of branches and make it useable. As would be expected from the change in the tree, the features used on the tree changed significantly from 7 features to using all 11 features. The variable importance rankings also showed significant shifts.

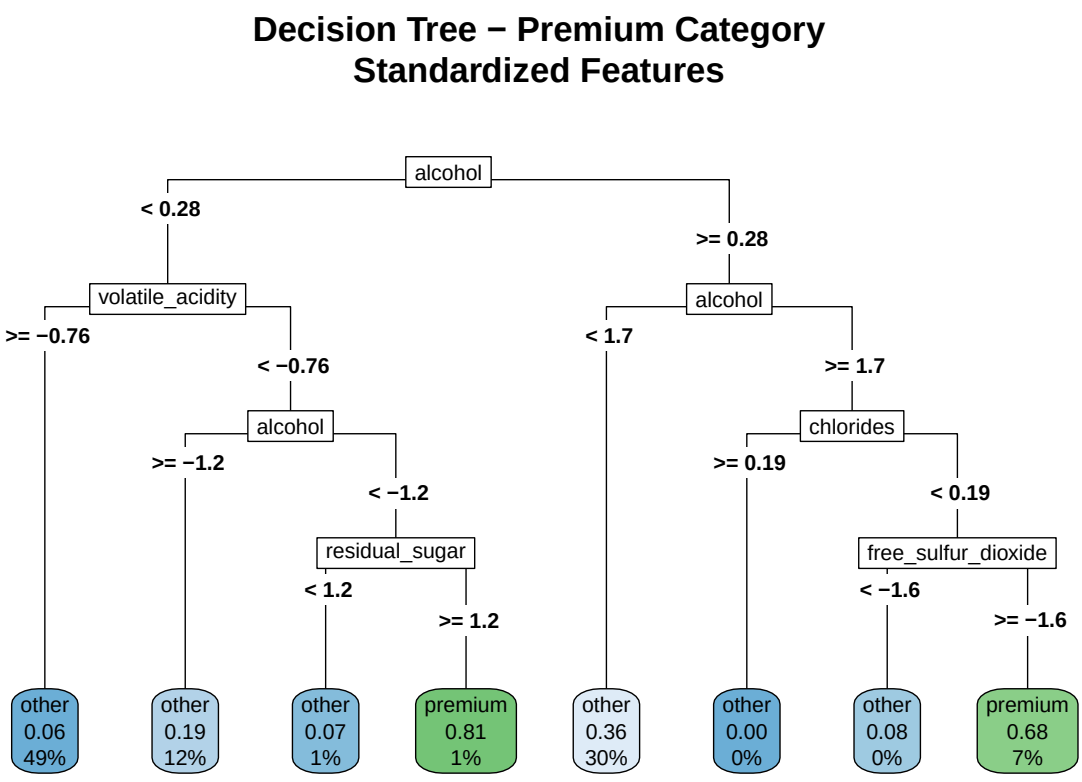

Figure 8: Decision Tree for Premium Category

```
#this chunk of code returns the tree terms that are used in the model
ind <- !(train_rpart2$finalModel$frame$var == "<leaf>")
tree_terms <-
  train_rpart2$finalModel$frame$var[ind] %>%
  unique() %>%
  as.character()
#display the tree terms
tree_terms
```

```
##  [1] "alcohol"              "volatile_acidity"     "pH"
##  [4] "density"              "citric_acid"          "free_sulfur_dioxide"
##  [7] "residual_sugar"       "total_sulfur_dioxide" "sulphates"
## [10] "fixed_acidity"        "chlorides"
```

```
#variable importance
vi <- varImp(train_rpart2)$importance %>% arrange(desc(Overall))
#save the variable importance for future comparison to other unsupervised models
vi_table <- add_column(vi_table, premium = rownames(vi)[1:4])
#display the variable importance
vi
```

```
##                      Overall
## alcohol              100.000
## density               60.061
## chlorides             35.204
## residual_sugar        34.639
## pH                    29.872
## total_sulfur_dioxide  29.718
## free_sulfur_dioxide   14.661
## volatile_acidity      14.350
## fixed_acidity         13.154
## citric_acid           11.558
## sulphates              0.000
```

2.9 Principal Component Analysis

Principal Component Analysis (PCA) was also used to analyze the features. The quality rating was not included in this analysis. We are looking at the relationships between features. Two graphs from the PCA are shown below. "PCA (Principal Component Analysis) is a multivariate data analysis method that allows us to summarize and visualize the information contained in a large data sets of quantitative variables."[7]

Figure 9 displays the variable contribution to the first two dimensions of the PCA. Features with arrows pointing in the same direction are positively correlated (such as density and residual sugar; or citric acid and fixed acidity). Arrows that point in the opposite direction are negatively correlated (as acidity goes up, pH goes down). The color scale displays the contribution of the variables to the first two dimensions.

Figure 10 presents a correlation plot of the variables across the first 5 dimensions of the PCA. The size and the color of the dots represent the amount of contribution to the dimension, which is shown in the scale on the right side of the plot.

The plots indicate that the key components are density, alcohol, residual sugar, total sulfur dioxide, pH and fixed acidity. The numeric results for Dimension 1 of the PCA analysis confirm the graphical conclusions and are shown below.

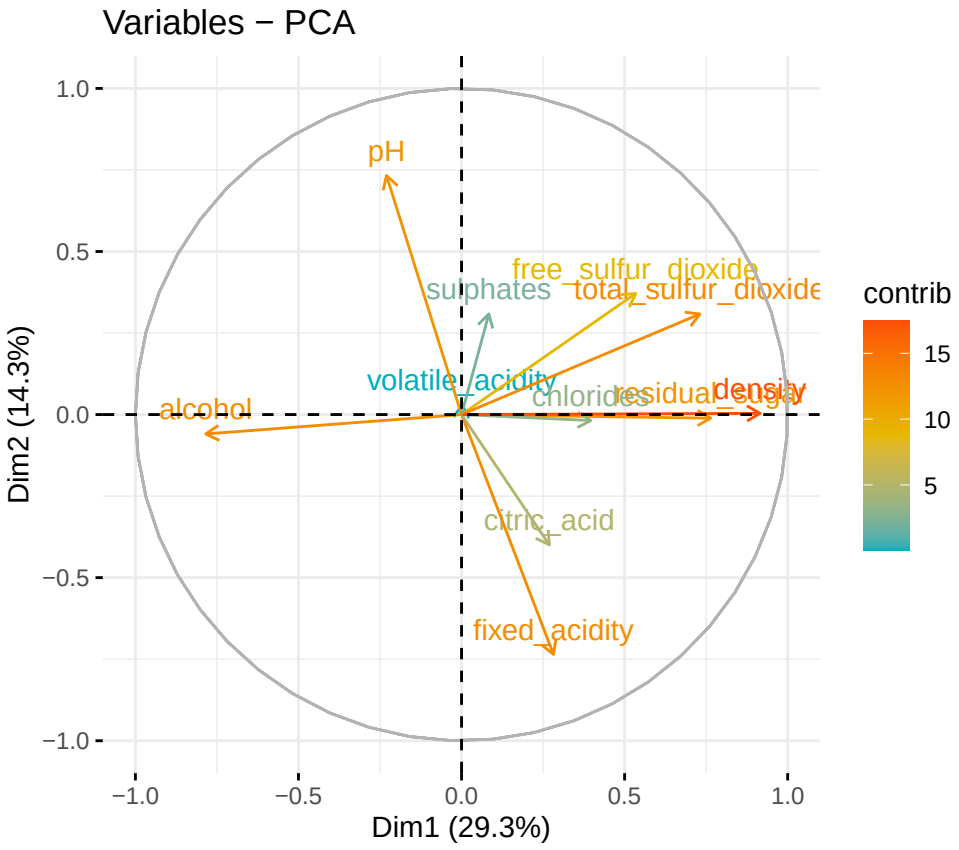

Figure 9: PCA Variables Plot

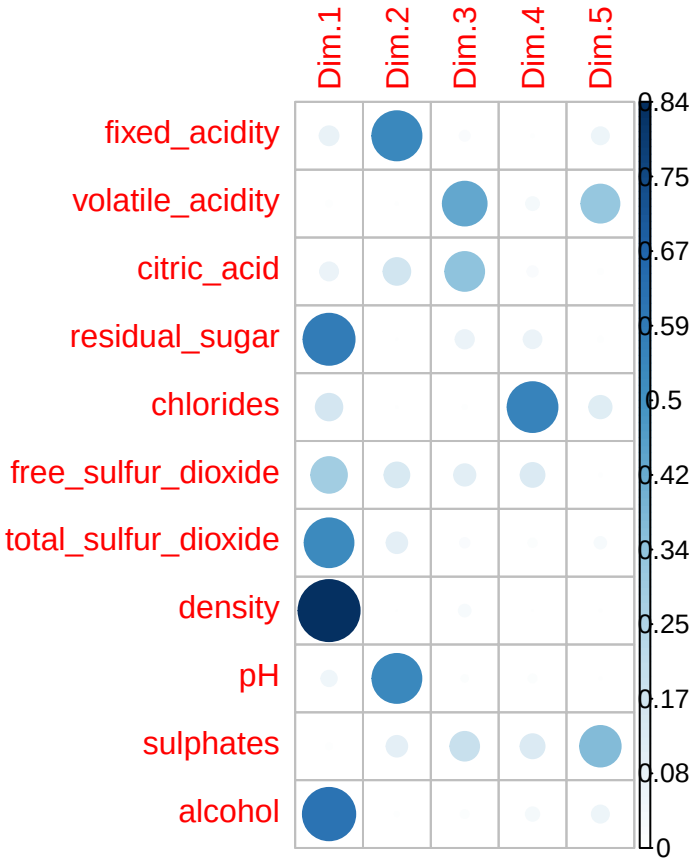

Figure 10: Correlation Plot

```
## # A tibble: 11 x 2
##    feature                  Dim.1
##    <chr>                    <dbl>
##  1 density                26.0
##  2 alcohol                19.1
##  3 residual_sugar         18.1
##  4 total_sulfur_dioxide   16.6
##  5 free_sulfur_dioxide     8.84
##  6 chlorides               4.88
##  7 fixed_acidity           2.46
##  8 citric_acid             2.25
##  9 pH                      1.64
## 10 sulphates               0.218
## 11 volatile_acidity        0.000137
```

2.10 Scatterplot Matrix Based on PCA

Based on the PCA analysis, the most important features are alcohol, fixed acidity, density, pH , residual sugar, and total sulfur dioxide. The scatter plot shown in Figure 11, shows these features in their standardized format. The "bulls eye" pattern still exists after standardization for these features.

2.11 Unsupervised Analysis Results

The 3 unsupervised models provide some commonality in terms of variable importance, but the results also contain some variability. As shown in Table 2 below, alcohol and density are the top two variables in all 3 models. Chlorides would be the next most important variable, followed by residual sugar and total sulfur dioxide in that order. Citric Acid only appears in one model.

Alcohol, density, and residual sugar are key factors, but they are also strongly correlated. The results indicate that increasing alcohol content is key, but the results have some variability, and none of the models were showing high levels of accuracy. For example, Dimension 1 of the PCA analysis only explained 25% of the variation and the RMSE for the first decision tree was 0.75. There could be two causes for this, the first is that we are missing key features, such as the volatile components. The second reason could be that the features we are looking at are correlated but not causative factors. For example, alcohol, density and residual sugar are important and highly correlated, but winemakers deal with and control ripeness of the grapes (which impacts sugar content or brix), fermentation time, fermentation temperature, and yeast. The features that winemakers directly control are not in our dataset, but they may provide much better predictors.

```
#variable importance
#save the variable importance for future comparison to other unsupervised models
vi_table <- add_column(vi_table, PCA = vc$feature[1:4])
#display the variable importance table
kable(vi_table, caption = "Variable Importance", booktabs=T)%>%
  kable_styling(latex_options = "hold_position")
```

[7]Kassambra, Alboukadel. 2017. PCA in R Using FactoMineR: Quick Scripts and Videos. http://www.sthda.com/english/articles/22-principal-component-methods-videos/65-pca-in-r-using-factominer-quick-scripts-and-videos/

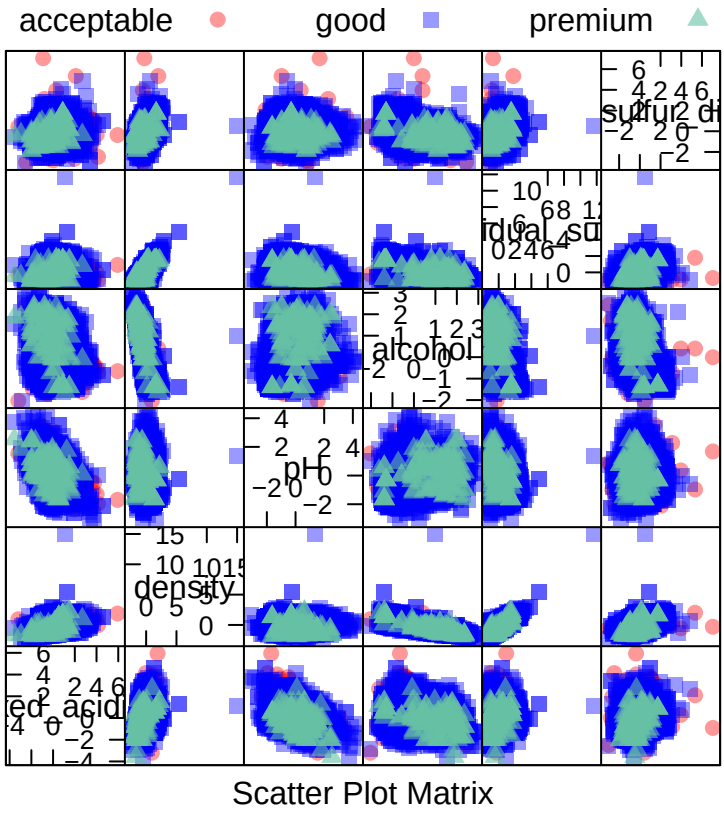

Figure 11: Important Features from PCA

Table 2: Variable Importance

rating	premium	PCA
alcohol	alcohol	density
chlorides	density	alcohol
citric_acid	chlorides	residual_sugar
total_sulfur_dioxide	residual_sugar	total_sulfur_dioxide

3 Modeling

Modeling was conducted for both numeric and categorical outcomes. The modeling was accomplished by leveraging the caret package. As can be seen from the code shown in this section, a "group_train" function was employed which allowed for the testing of multiple models in one run. Many more models were tested than the ones included in this report. Only a few example model outcomes are included as part of this report.

3.1 Regression Models

Regression Models provide numeric predictions of results. In this section, we are going to evaluate multiple models of varying types to determine which models perform best. Since this is a regression, we will be looking for models that minimize RMSE (Root Mean Square Error).

Table 3 shows the summary of the Regression model runs. Given the previous analysis of the data, it was not surprising to see that the linear models did not perform well. On the other hand, the random forest based models all produced the lowest RMSE results at 0.58.

```
#create the test and training data sets from the standardized features
train <- train_feat %>% cbind(y_num)
test <- test_feat %>% cbind(y_test_num)


# create a list of machine learning models to evaluate
models <- c("glm", "svmLinear", "gamboost","gamLoess", "knn",
            "kknn", "gam","ranger", "rf", "Rborist", "mlp",
            "svmRadial", "svmRadialCost", "svmRadialSigma")

#this function will evaluate each function in the supplied list
#it returns a list of training models
group_train <- function(model_list, seed = 831){
  # set the seed for the model run or use the user override
  set.seed(seed, sample.kind = "Rounding")
  # get the length of the model list
  l <- length(model_list)
  # train the first model in the input list and initialize the train_list
  train_result <- train(quality ~ ., method = model_list[1], data = train)
  train_list <- list(train_result)
  # train the remaining models
  for (i in 2:l){
    train_result <- train(quality ~ ., method = model_list[i], data = train)
    train_list[[i]] <- train_result
  }
```

```
  # name the models in the list
  names(train_list) <- model_list
  # return the list of training models
  return(train_list)
}
# train the models
model_list <- group_train(models)

# Create a prediction matrix with a column for each ML Model
p <- sapply(model_list,predict, test)

# Calculate the rmse for each model
a <- apply(p, 2, RMSE, test$quality) %>% enframe(name="Model", value = "RMSE")
# display the rmse table
kable(a, caption = "Regression Models", booktabs=T)%>%
  kable_styling(latex_options = "hold_position")
```

Table 3: Regression Models

Model	RMSE
glm	0.74343
svmLinear	0.74211
gamboost	0.72163
gamLoess	0.70576
knn	0.70293
kknn	0.68195
gam	0.71133
ranger	0.58361
rf	0.58374
Rborist	0.57969
mlp	0.77867
svmRadial	0.67307
svmRadialCost	0.67249
svmRadialSigma	0.66556

3.1.1 Random Forest Rborist

Rborist produced some of the best results for regression models. In this section, we are going to tune the Rborist model to see if we can improve on the results.

```
#reset the train and test sets for evaluation as a numeric
train <- train_feat %>% cbind(y_num)
test <- test_feat %>% cbind(y_test_num)

#train the model using a tuning grid
train_rborist <- train(quality ~ .,
             method = "Rborist",
             tuneGrid = data.frame(predFixed = 2,
                                   minNode = c(3, 50)),
             data = train)

#calculate the predicted values for the test set
y_hat <- predict(train_rborist, test)
#calculate the model RMSE
rmse_rborist <- RMSE(test$quality, y_hat)
#display the results
cat("The RMSE for the tuned Rborist model is", rmse_rborist)
```

```
## The RMSE for the tuned Rborist model is 0.58
```

The RMSE for the tuned model is the same as for the base caret call. The default tuning parameters in the caret package performed very well.

3.2 Classification Models

Classification Models provide categorical predictions of results. In this section, we are going to evaluate multiple models of varying types to determine which models perform best. Since this is a classification modeling approach, we will be looking for models that maximize accuracy. Accuracy is the proportion of correct (true positive) predictions. The R confusionMatrix functionality will be used to calculate accuracy.

We will also convert the quality rating factors back to numbers and assess them via RMSE. This is not being used to optimize the model, but it might be interesting to obtain and compare the RMSE calculated this way to the values produced during the regression model analysis.

```
#reset the train and test sets for modeling as categories
train <- train_feat %>% cbind(y_cat)
test <- test_feat %>% cbind(y_test_cat)

# create a list of machine learning models to evaluate
models <- c("knn", "kknn", "Rborist", "ranger", "mlp","svmRadial")

#this function will evaluate each function in the supplied list
#it returns a list of training models
group_train <- function(model_list, seed = 831){
  # set the seed for the model run or use the user override
  set.seed(seed, sample.kind = "Rounding")
  # get the length of the model list
  l <- length(model_list)
```

```
  # train the first model in the input list and initialize the train_list
  train_result <- train(quality ~ ., method = model_list[1], data = train)
  train_list <- list(train_result)
  # train the remaining models
  for (i in 2:l){
    train_result <- train(quality ~ ., method = model_list[i], data = train)
    train_list[[i]] <- train_result
  }
  # name the models in the list
  names(train_list) <- model_list
  # return the list of training models
  return(train_list)
}
# train the models
model_list <- group_train(models)

# calculate the rmse
RMSE2 <- function(predicted_ratings, true_ratings){
  #the min factor is 3, so add 2 to the conversion to get the rating correct
  tr <- as.numeric(true_ratings) +2
  pr <- as.numeric(predicted_ratings)
  sqrt(mean((tr - pr)^2))
}

# Create a prediction matrix with a column for each ML Model
p <- sapply(model_list, predict, test)

# Calculate the rmse for each model
a <- apply(p, 2, RMSE2, test$quality) %>% enframe(name="Model", value = "RMSE")
kable(a, caption = "Classification Models", booktabs=T) %>%
  kable_styling(latex_options = "hold_position")
```

Table 4: Classification Models

Model	RMSE
knn	0.82251
kknn	0.78311
Rborist	0.62922
ranger	0.61445
mlp	0.79475
svmRadial	0.75795

```
#initialize an empty tibble to hold the accuracy results
accuracy_df <- tibble(Model = character(),
                      Accuracy = numeric())
# calculate and display the accuracy for all of the models run
for(i in 1:ncol(p)){
  y_hat <- factor(p[,i], levels(test$quality))
  cm <- confusionMatrix(y_hat, test$quality)
  accuracy_df[i,1] <- models[i]
  accuracy_df[i,2] <- cm$overall["Accuracy"]
```

```
}
kable(accuracy_df, caption = "Classification Model Accuracy", booktabs=T )%>%
  kable_styling(latex_options = "hold_position")
```

Table 5: Classification Model Accuracy

Model	Accuracy
knn	0.53980
kknn	0.62551
Rborist	0.70612
ranger	0.70714
mlp	0.54388
svmRadial	0.55918

```
# Get the predictors for the best model, which is "ranger"
y_hat <- factor(p[,"ranger"], levels(test$quality))

# print out the specificity and sensitivity
cm <- confusionMatrix(y_hat, test$quality)
cm$overall["Accuracy"]
```

```
## Accuracy
##  0.70714
```

```
#cm$byClass[,1:2]

#use the broom function to get the confusion matrix output in tidy format
tcm <- tidy(cm)
#drop the first two rows of the data and then create a wide version
tcm <- tcm[3:nrow(tcm),] %>%
  select(term, class, estimate) %>%
  pivot_wider(names_from = term, values_from = estimate) %>%
  select(class, sensitivity, specificity, prevalence, detection_rate)
#display the results
tcm
```

```
## # A tibble: 7 x 5
##   class sensitivity specificity prevalence detection_rate
##   <chr>       <dbl>       <dbl>      <dbl>          <dbl>
## 1 3           0           1        0.00204        0
## 2 4           0.207       1        0.0296         0.00612
## 3 5           0.667       0.915    0.303          0.202
## 4 6           0.868       0.630    0.449          0.390
## 5 7           0.529       0.965    0.178          0.0939
## 6 8           0.405       0.999    0.0378         0.0153
## 7 9           0           1        0.00102        0
```

```
#save the predictions from the ranger model to use in an ensemble
y_hat_ranger <- y_hat
```

Ranger produced the best results for accuracy and RSME. The results for specificity and sensitivity are shown above. Because of prevalence issues, our sensitivity for predicting premium (8 & 9 ratings) or acceptable(3 & 4 ratings) is low. The model errors towards the high prevalence predictions from 5 to 7.

From a practical standpoint, if we were buying or blending wine, we would want to avoid the low ratings or target the high ratings. Our specificity is perfect for class 8 and 9 wines. If we predict a premium wine, there is nearly a 100% chance that it is a premium wine.Our sensitivity though could be improved. It is currently 0 for class 9 and 0.41 for class 8.

If we could find a way to maintain a good specificity, but improve our sensitivity, it would make our model more useful. If another model offered significantly better specificity, an ensemble might work to improve the predictions.

A review of sensitivity and specificity of the other models did not show any models with significantly better sensitivity. The knn model results are shown below as an example.

```
# Get the predictors for the knn model as an example
y_hat <- factor(p[,1], levels(test$quality))
# print out the specificity and sensitivity
cm <- confusionMatrix(y_hat, test$quality)
cm$overall["Accuracy"]
```

```
## Accuracy
##     0.54
```

```
#cm$byClass[,1:2]

#use the broom function to get the confusion matrix output in tidy format
tcm2 <- tidy(cm)
#drop the first two rows of the data and then create a wide version
tcm2 <- tcm2[3:nrow(tcm2),] %>%
  select(term, class, estimate) %>%
  pivot_wider(names_from = term, values_from = estimate) %>%
  select(class, sensitivity, specificity, prevalence, detection_rate)
#display the results
tcm2
```

```
## # A tibble: 7 x 5
##   class sensitivity specificity prevalence detection_rate
##   <chr>       <dbl>       <dbl>      <dbl>          <dbl>
## 1 3           0           1        0.00204        0
## 2 4           0.0345      0.997    0.0296         0.00102
## 3 5           0.562       0.824    0.303          0.170
## 4 6           0.655       0.580    0.449          0.294
## 5 7           0.408       0.885    0.178          0.0724
## 6 8           0.0541      0.992    0.0378         0.00204
## 7 9           0           1        0.00102        0
```

As the results above show, not only is the knn model worse on accuracy, but sensitivity at the tails is much worse than the Ranger model.

3.3 Leveling to Compensate for Prevalence

We will attempt to remove prevalence by "leveling" the observations of all categories. The leveling process will create a dataset with 2000 observations of every category. The process will start with the base observations for that category and then sample with replacement to create the additional observations necessary to level the observations for all categories. The sampling with replacement approach is similar to bootstrapping.

I believe that this approach is unique. My searches did not come up with any similar methodologies to address the prevalence issue. In our machine learning class, a method to control prevalence using a Naive Bayes approach was presented.[8] This approach was used to compensate for prevalence of a binary outcome (gender) with a singular feature (height), where there was potential overlap in the choice of outcomes based on the feature and where we had insight on a bias in the dataset that under represented females. Extending it to handle an outcome with multiple categories and where the actual prevalence was correctly reflected by the dataset was beyond the scope of this project.

Using the leveled dataset, if prevalence contributes to the low sensitivity in the extremely high and low ratings for our model, the leveling process should significantly improve the sensitivity for our models. If the 11 features that we are using simply do not contain a sufficient signal to make a good prediction, leveling will not impact the results significantly

'pub.va/leanpub.com/d

```
# create clean train and test sets
train <- train_feat %>% cbind(y_cat)
test <- test_feat %>% cbind(y_test_cat)

#determine the number of observations by rating
obs_by_category <- train %>% group_by(quality) %>% summarize(obs = n())

#create a leveled dataset with 2000 observations for each rating
train_level_prev <- train
for(i in 1:7){
  obs <- obs_by_category[i,2] %>% .$obs
  qlty <- obs_by_category[i, 1] %>% .$quality
  sample_obs <- train %>% filter(quality == qlty)
  ind <- sample(1:nrow(sample_obs), 2000 - obs, replace = TRUE)
  new_obs <- sample_obs[ind,]
  train_level_prev <- rbind(train_level_prev, new_obs)
}
```

3.4 Leveled Models

This section repeats the multiple model run from above, but this time using the leveled features. For efficiency purposes, the number of models was also reduced. Table 6 summarizes the Accuracy results and Table 7 summarizes the RMSE results.

```
#set the training set for the leveled model run
train <- train_level_prev
#remove the redundant object
rm(train_level_prev)

#reset the test set for modeling as categories
```

[8] Irizzary, Rafael, 2019. Introduction to Data Science.<[[https:/com/datascom/datasciencebook\\](https:/com/datascom/datasciencebook){.uri]
/

```r
# train <- train_feat %>% cbind(y_cat)
test <- test_feat %>% cbind(y_test_cat)

# create a list of machine learning models to evaluate
models <- c("knn", "kknn", "ranger", "rf", "Rborist", "mlp")

#this function will evaluate each function in the supplied list
#it returns a list of training models
group_train <- function(model_list, seed = 831){
  # set the seed for the model run or use the user override
  set.seed(seed, sample.kind = "Rounding")
  # get the length of the model list
  l <- length(model_list)
  # train the first model in the input list and initialize the train_list
  train_result <- train(quality ~ ., method = model_list[1], data = train)
  train_list <- list(train_result)
  # train the remaining models
  for (i in 2:l){
    # print(model_list[i])
    train_result <- train(quality ~ ., method = model_list[i], data = train)
    train_list[[i]] <- train_result
  }
  # name the models in the list
  names(train_list) <- model_list
  # return the list of training models
  return(train_list)
}
# train the models
model_list <- group_train(models)

# Create a prediction matrix with a column for each ML Model
# using the caret predict function
p <- sapply(model_list, predict, test)

# Get the predictors for the ranger model results
y_hat <- factor(p[,"ranger"], levels(test$quality))

#calculate and display the confusion matrix
cm <- confusionMatrix(y_hat, test$quality)
cm$overall["Accuracy"]
```

```
## Accuracy
##  0.69898
```

```r
#use the broom function to get the confusion matrix output in tidy format
tcm_r <- tidy(cm)
#drop the first two rows of the data and then create a wide version
tcm_r <- tcm_r[3:nrow(tcm_r),] %>%
  select(term, class, estimate) %>%
  pivot_wider(names_from = term, values_from = estimate) %>%
  select(class, sensitivity, specificity, prevalence, detection_rate)
#display the results
cat("\n\n\nThe confusion matrix results for the ranger model after leveling are:\n")
```

```
##
##
##
## The confusion matrix results for the ranger model after leveling are:
```

```
tcm_r
```

```
## # A tibble: 7 x 5
##   class sensitivity specificity prevalence detection_rate
##   <chr>       <dbl>       <dbl>      <dbl>          <dbl>
## 1 3           0           1        0.00204        0
## 2 4           0.241       0.998    0.0296         0.00714
## 3 5           0.724       0.900    0.303          0.219
## 4 6           0.780       0.7      0.449          0.35
## 5 7           0.603       0.923    0.178          0.107
## 6 8           0.405       0.999    0.0378         0.0153
## 7 9           0           1        0.00102        0
```

```
# create a data frame to store the accuracy results
accuracy_df <- tibble(Model = character(),
                      Accuracy = numeric())
# print the accuracy for all of the models run
for(i in 1:ncol(p)){
  y_hat <- factor(p[,i], levels(test$quality))
  cm <- confusionMatrix(y_hat, test$quality)
  accuracy_df[i,1] <- models[i]
  accuracy_df[i,2] <- cm$overall["Accuracy"]
}
kable(accuracy_df, caption = "Leveled Model Accuracy", booktabs=T )%>%
  kable_styling(latex_options = "hold_position")
```

Table 6: Leveled Model Accuracy

Model	Accuracy
knn	0.48061
kknn	0.64286
ranger	0.69898
rf	0.70102
Rborist	0.70306
mlp	0.28980

```
#create a function to calculate the RMSE values
RMSE2 <- function(predicted_ratings, true_ratings){
  #the min factor is 3, so add 2 to the conversion to get the rating correct
  tr <- as.numeric(true_ratings) +2
  pr <- as.numeric(predicted_ratings)
  sqrt(mean((tr - pr)^2))
}

# Calculate the rmse for each model
```

```
a <- apply(p, 2, RMSE2, test$quality) %>% enframe(name="Model", value = "RMSE")
# display the rmse results
kable(a, caption = "Leveled Model RMSE", booktabs=T)%>%
  kable_styling(latex_options = "hold_position")
```

Table 7: Leveled Model RMSE

Model	RMSE
knn	1.00509
kknn	0.76265
ranger	0.61611
rf	0.61694
Rborist	0.62024
mlp	1.30227

As expected, there was a decrease in accuracy for the model conducted on the leveled data, but it was slight decreasing from the original accuracy of 0.73 to 0.70 for the random forest models. Since ranger offered the best RMSE results out of this group, let's use that as a tie-breaker and take a look at the sensitivity and specificity of the ranger model.

The knn model showed a nice jump in sensitivity

```
# show the pre-leveled results
cat("The confusion matrix results for the knn model prior to leveling were:\n")
```

```
## The confusion matrix results for the knn model prior to leveling were:
```

```
tcm2
```

```
## # A tibble: 7 x 5
##   class sensitivity specificity prevalence detection_rate
##   <chr>       <dbl>       <dbl>      <dbl>          <dbl>
## 1 3           0           1        0.00204        0
## 2 4           0.0345      0.997    0.0296         0.00102
## 3 5           0.562       0.824    0.303          0.170
## 4 6           0.655       0.580    0.449          0.294
## 5 7           0.408       0.885    0.178          0.0724
## 6 8           0.0541      0.992    0.0378         0.00204
## 7 9           0           1        0.00102        0
```

```
# Get the predictors for the knn model results
y_hat <- factor(p[,1], levels(test$quality))

#calculate and display the confusion matrix
cm <- confusionMatrix(y_hat, test$quality)
cm$overall["Accuracy"]
```

```
## Accuracy
##  0.48061
```

```
#use the broom function to get the confusion matrix output in tidy format
tcm <- tidy(cm)
#drop the first two rows of the data and then create a wide version
tcm <- tcm[3:nrow(tcm),] %>%
  select(term, class, estimate) %>%
  pivot_wider(names_from = term, values_from = estimate) %>%
  select(class, sensitivity, specificity, prevalence, detection_rate)
#display the results
cat("\n\n\nThe confusion matrix results for the knn model after leveling are:\n")
```

```
##
##
##
## The confusion matrix results for the knn model after leveling are:
```

```
tcm
```

```
## # A tibble: 7 x 5
##   class sensitivity specificity prevalence detection_rate
##   <chr>       <dbl>       <dbl>      <dbl>          <dbl>
## 1 3           0           0.994    0.00204         0
## 2 4           0.483       0.941    0.0296          0.0143
## 3 5           0.542       0.835    0.303           0.164
## 4 6           0.409       0.798    0.449           0.184
## 5 7           0.552       0.798    0.178           0.0980
## 6 8           0.541       0.935    0.0378          0.0204
## 7 9           0           0.999    0.00102         0
```

As shown above, the sensitivity when predicting category 8 increased from 0.0541 in the original model to 0.541 in the leveled model. Although the overall model accuracy is lacking, this increase in sensitivity while still maintaining very good specificity, could be useful in some applications such as wine purchasing.

4 Summary of Results

4.0.1 Unsupervised Models

Three unsupervised models were explored; a decision tree for numeric ratings, a decision tree for premium vs other categories, and a PCA analysis. The 3 models had some similarities in important variables, but the accuracy of the models as measured by RMSE, Accuracy or percent of variability explained was only mediocre.

In addition, several of the features that had the highest importance are also highly correlated. Alcohol, density and residual sugar are key features, but the underlying factors used by winemakers would typically be original gravity of the juice, brix, yeast variety, fermentation temperature, fermentation time, etc.

4.0.2 Supervised models

A variety of models were tested using the caret package. The best results for both predicting categories and accuracy were the random forest family of models. The RMSE of the Rborist model was 0.58 which is not bad, but not precise.

Errors in the Rborist model primarily existed in the high and low quality ratings. It was suspected that prevalence might be impacting the results because of the low prevalence of both high and low quality ratings.

Leveling was introduced to test the impact of prevalence. Leveling had a significant impact on the knn model, but had no impact on the random forest models. The random forest models still performed significantly better than any other family of models. It appears that there is not a clear signal present in the features contained in the dataset to accurately predict highly and poorly rated wines.

A look at the confusion matrix when predicting categories showed that although the accuracy was poor, the specificity of the predictions was actually very good and nearly 100% accurate. The sensitivity was poor only picking up a small percentage of the premium quality ratings.

5 Conclusions

Several conclusions can be drawn from these results.

1. The data set contains a very limited set of features.
    - These products are based on physicochemical characteristics of the finished wine. Volatile compounds, such as esters, thiols, fatty acids, etc., were not measured and are key aroma components in finished wine.
    - Features associated with the wine making ingredients and process were not included in the dataset. These features might provide much more insight for the unsupervised models and increase the accuracy of the supervised models.
    - Quality ratings were expressed as a single final statistic. Typically these ratings have a number of subcategories and are complied by multiple individual tasters. Access to the quality scoring process data might be very useful
2. Accuracy and RMSE of models could be better
    - The results are marginally useable. They do not predict high and low quality ratings well, and that is what would be most useful.
    - Specificity is good, which would make the models useful for wine purchasing if the ratings were still unknown.
    - Wether the models could be useful for blending or wine making would need further testing.
3. Leveling did not improve the accuracy of the models but did improve the sensitivity of the knn model for predicting the premium category. If this were being used by a wine buyer, the increased sensitivity might be valuable.

Overall, the models produced have some limited usefulness. I suspect that expanding the features available in the data set, as described above, would produce improved results.

6 Further Study

The primary area for increased study would be to broaden the features data set and include key features that address the volatile aroma components, the wine making process, and the quality scoring process.

In addition, it would be interesting to see if establishing blending targets using the models could be leveraged to improve quality scores. This might be very useful to winemakers.

Extensibility would be another area of interest. This dataset was for white vihno verde produced in Northern Portugal. I suspect that the characteristics for red wines from the same region would be quite different. Also, I would suspect that region and grape varieties have a major impact on the model. Studying the impact and variability over a variety of regions, grapes, and wine styles might lead to some interesting insights.

7 References

Australian Academy of Science. 2017. The chemistry of wine: Part 1. https://www.science.org.au/curious/earth-environment/chemistry-wine-part-1

P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009. https://archive.ics.uci.edu/ml/datasets/Wine+Quality

Hastie, Trevor, Robert Tibshirani, Jerome Friedman. 2017. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Second Edition

Irizzary, Rafael, 2019. Introduction to Data Science. https://leanpub.com/datasciencebook

Kassambra, Alboukadel. 2017. PCA in R Using FactoMineR: Quick Scripts and Videos. http://www.sthda.com/english/articles/22-principal-component-methods-videos/65-pca-in-r-using-factominer-quick-scripts-and-videos/

Kuhn, Max. 2019. The caret Package. https://topepo.github.io/caret/

Lê, S., Josse, J. & Husson, F. (2008). FactoMineR: An R Package for Multivariate Analysis. Journal of Statistical Software. 25(1). pp. 1-18.

Therneau, Terry, Elizabeth Atkinson. 2019. An Introduction to Recursive Partitioning Using the RPART Routines. https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf

Wansbrough, Heather, Robert Sherlock, Maurice Barnes, Malcolm Reeves. 2017. Chemistry in Winemaking. pg 5. https://nzic.org.nz/app/uploads/2017/10/6B.pdf