Wine Quality Prediction from Physiochemical Properties

Rebecca L.E. Miller

21 November, 2020

Summary

This analysis works with a wine quality dataset that contains numeric physiochemical measurements and a subjective `quality` score for multiple Portuguese "vinho verde" wines. Separate files are provided for red and white wines, and we combined the two into a single dataset. We explored the data and discovered that `alcohol` content and `volatile.acidity` were the most strongly correlated with `quality` without being intercorrelated. We experimented with regression to predict the actual `quality` score, but ended up using a classification model where we assigned a `quality` score cutoff for "Good" and "NotGood" wines based on a modified Tukey outlier procedure. The model that consistently performed the best among the models we explored was the Random Forest model. Our prediction for the final hold-out set, which we selected as 10% of the total data, was able to achieve 89% overall accuracy with an F-score around 0.68. There are several other algorithms that we could explore, and there is still a lot of room for improvement, particularly in the lower-prevalence very high and very low `quality` scores.

Introduction

The data used in this analysis is from the University of California Irvine Machine Learning Repository, data set number 1341136, or the Wine Quality data set from "Modeling Wine Preferences by Data Mining from Physiochemical Properties." [1] The data consists of eleven physiochemical properties and a quality score between 0 and 10 for each wine in the set. There are two subsets, both for Portuguese "vinho verde" wines, one for red wine and one for white wine. The most obvious machine learning task for this dataset is to use the physiochemical data as predictors and the quality score as the response variable, or in more direct language, to use the physiochemical properties to predict quality. More succinctly

Task: Predict good, bad and average wine based on physiochemical properties.

Download Data

The data set is available in two *.csv files found at red and white. The files can be downloaded with the following code.

```
###------------------------------------------------------------
###     Download And Parse Files From Remote Repository
###-----------------------------------------------------------
# create a temporary file in the local filesystem for download of raw csv
dl <- tempfile()
```

[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.

```
# download the file
download.file("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red

# read the red wine quality file into a data frame
red <- read.csv(file = dl, header = TRUE, sep = ";")

# remove the dl file handle
rm(dl)

#--------------------- White Wine File ---------------------
# make a new temp file
dl1 <- tempfile()

# Download raw csv
download.file("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-whi

# read the white wine quality file into a data frame
white <- read.csv(file = dl1, header = TRUE, sep = ";")

rm(dl1)
```

Data Exploration

The physiochemical properties, or columns of the data sets, for the wines in the two files are

```
union(colnames(red), colnames(white))
```

```
##  [1] "fixed.acidity"        "volatile.acidity"     "citric.acid"
##  [4] "residual.sugar"       "chlorides"            "free.sulfur.dioxide"
##  [7] "total.sulfur.dioxide" "density"              "pH"
## [10] "sulphates"            "alcohol"              "quality"
```

Both files have the same physiochemical property columns

```
setdiff(colnames(red),colnames(white))
```

```
## character(0)
```

Neither data set contains `NA` values.

```
sum(is.na(red))
```

```
## [1] 0
```

```
sum(is.na(white))
```

```
## [1] 0
```

Now that we have a basic understanding of the structures of the two data sets, we can combine them into a single data set. We can try to predict wine quality based on the physiochemical properties and the wine color category that we create when we join the two data sets. We also will designate a hold-out subset and a test subset that we will use to develop the model.

```r
# make new columns for the wine color or type
red <- red %>% mutate(red = 1)
white <- white %>% mutate(red = 0)

# combine the two data frames into one
wines <- red %>% full_join(., white)

# Show a glimpse of the data
head(wines)
```

```
##   fixed.acidity volatile.acidity citric.acid residual.sugar chlorides
## 1           7.4             0.70        0.00            1.9     0.076
## 2           7.8             0.88        0.00            2.6     0.098
## 3           7.8             0.76        0.04            2.3     0.092
## 4          11.2             0.28        0.56            1.9     0.075
## 5           7.4             0.70        0.00            1.9     0.076
## 6           7.4             0.66        0.00            1.8     0.075
##   free.sulfur.dioxide total.sulfur.dioxide density   pH sulphates alcohol
## 1                  11                   34  0.9978 3.51      0.56     9.4
## 2                  25                   67  0.9968 3.20      0.68     9.8
## 3                  15                   54  0.9970 3.26      0.65     9.8
## 4                  17                   60  0.9980 3.16      0.58     9.8
## 5                  11                   34  0.9978 3.51      0.56     9.4
## 6                  13                   40  0.9978 3.51      0.56     9.4
##   quality red
## 1       5   1
## 2       5   1
## 3       5   1
## 4       6   1
## 5       5   1
## 6       5   1
```

```r
summary(wines)
```

```
##  fixed.acidity    volatile.acidity  citric.acid     residual.sugar
##  Min.   : 3.800   Min.   :0.0800   Min.   :0.0000   Min.   : 0.600
##  1st Qu.: 6.400   1st Qu.:0.2300   1st Qu.:0.2500   1st Qu.: 1.800
##  Median : 7.000   Median :0.2900   Median :0.3100   Median : 3.000
##  Mean   : 7.215   Mean   :0.3397   Mean   :0.3186   Mean   : 5.443
##  3rd Qu.: 7.700   3rd Qu.:0.4000   3rd Qu.:0.3900   3rd Qu.: 8.100
##  Max.   :15.900   Max.   :1.5800   Max.   :1.6600   Max.   :65.800
##    chlorides       free.sulfur.dioxide total.sulfur.dioxide    density
##  Min.   :0.00900   Min.   :  1.00      Min.   :  6.0        Min.   :0.9871
##  1st Qu.:0.03800   1st Qu.: 17.00      1st Qu.: 77.0        1st Qu.:0.9923
##  Median :0.04700   Median : 29.00      Median :118.0        Median :0.9949
##  Mean   :0.05603   Mean   : 30.53      Mean   :115.7        Mean   :0.9947
##  3rd Qu.:0.06500   3rd Qu.: 41.00      3rd Qu.:156.0        3rd Qu.:0.9970
##  Max.   :0.61100   Max.   :289.00      Max.   :440.0        Max.   :1.0390
##        pH          sulphates         alcohol          quality     
##  Min.   :2.720   Min.   :0.2200   Min.   : 8.00   Min.   :3.000  
##  1st Qu.:3.110   1st Qu.:0.4300   1st Qu.: 9.50   1st Qu.:5.000  
##  Median :3.210   Median :0.5100   Median :10.30   Median :6.000  
##  Mean   :3.219   Mean   :0.5313   Mean   :10.49   Mean   :5.818  
##  3rd Qu.:3.320   3rd Qu.:0.6000   3rd Qu.:11.30   3rd Qu.:6.000  
##  Max.   :4.010   Max.   :2.0000   Max.   :14.90   Max.   :9.000  
##       red        
##  Min.   :0.0000  
##  1st Qu.:0.0000  
##  Median :0.0000  
##  Mean   :0.2461  
##  3rd Qu.:0.0000  
##  Max.   :1.0000
```

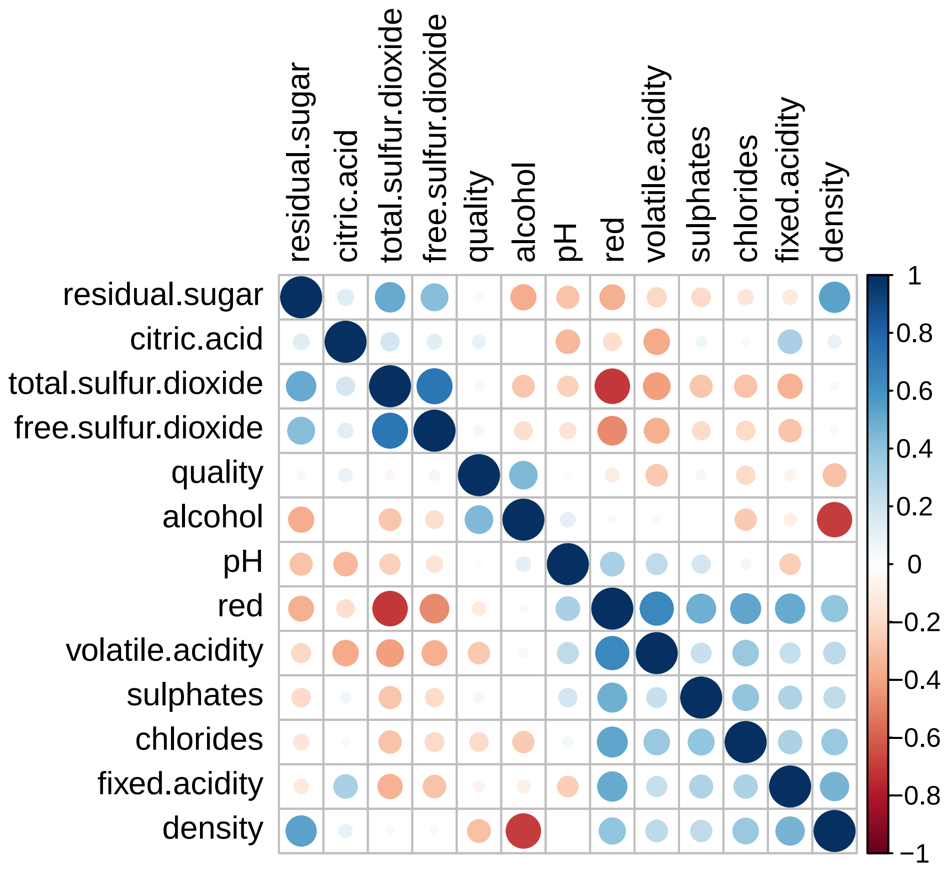
```r
# set the seed for reproducibility, user set.seed(1234) for R 3.5 or earlier
set.seed(1234, sample.kind="Rounding")

# use createDataPartition as in the Data Science course material
# set aside 10% of the data as a final test set, which we call hold_out
test_index <- createDataPartition(y = wines$quality, times = 1,
                                  p = 0.1, list = FALSE)
full_train <- wines[-test_index, ]
hold_out <- wines[test_index, ]

# partition the data a second time to create test and train sets from the
# full_train set
set.seed(1234, sample.kind="Rounding")
test_index2 <- createDataPartition(y = full_train$quality, times = 1,
                                  p = 0.1, list = FALSE)
train <- full_train[-test_index2, ]
test <- full_train[test_index2, ]
```

We can check to see if predictors are correlated with each other, as well.

```r
# Make a correlation plot of the train set
corrplot(cor(train), order = "AOE", tl.col = "black")
```



We see that quality is positively correlated with alcohol content, while negatively correlated with volatile acidity, chloride content, and density. We should note that there are multiple intercorrelations just between these variables that are strongly correlated with quality. For example, density and alcohol content are strongly negatively correlated. We will probably need to deal with the lack of independence among predictors in some way.

We can also look at histograms of our variables in the `train` set to get some idea of the character of the data.

We see that quality is approximately normally distributed, with few values in the set of {4,8} and most values in the set {5,6,7}. That is the expected result, with just a few very good wines and a few very bad ones.

Other approximately normally distributed predictors include `chlorides`, `citric.acid`, `fixed.acidity`, `pH`, `sulphates`, and `volatile.acidity`. The remaining predictors are either bimodal, like `total.sulfur.dioxide` or are significantly skewed, as with `alcohol` content.

With a basic grasp of the shape and size of our data, we can begin to construct models. Because `quality` is a continuous set of natural numbers, we can allow our prediction to be continuous, as well. Our predictions will be the continuous set for the range of the training set,

```
range(train$quality)
```

```
## [1] 3 9
```

We can therefore choose our loss function as something appropriate for continuous data. We will use Root Mean Squared Error, or

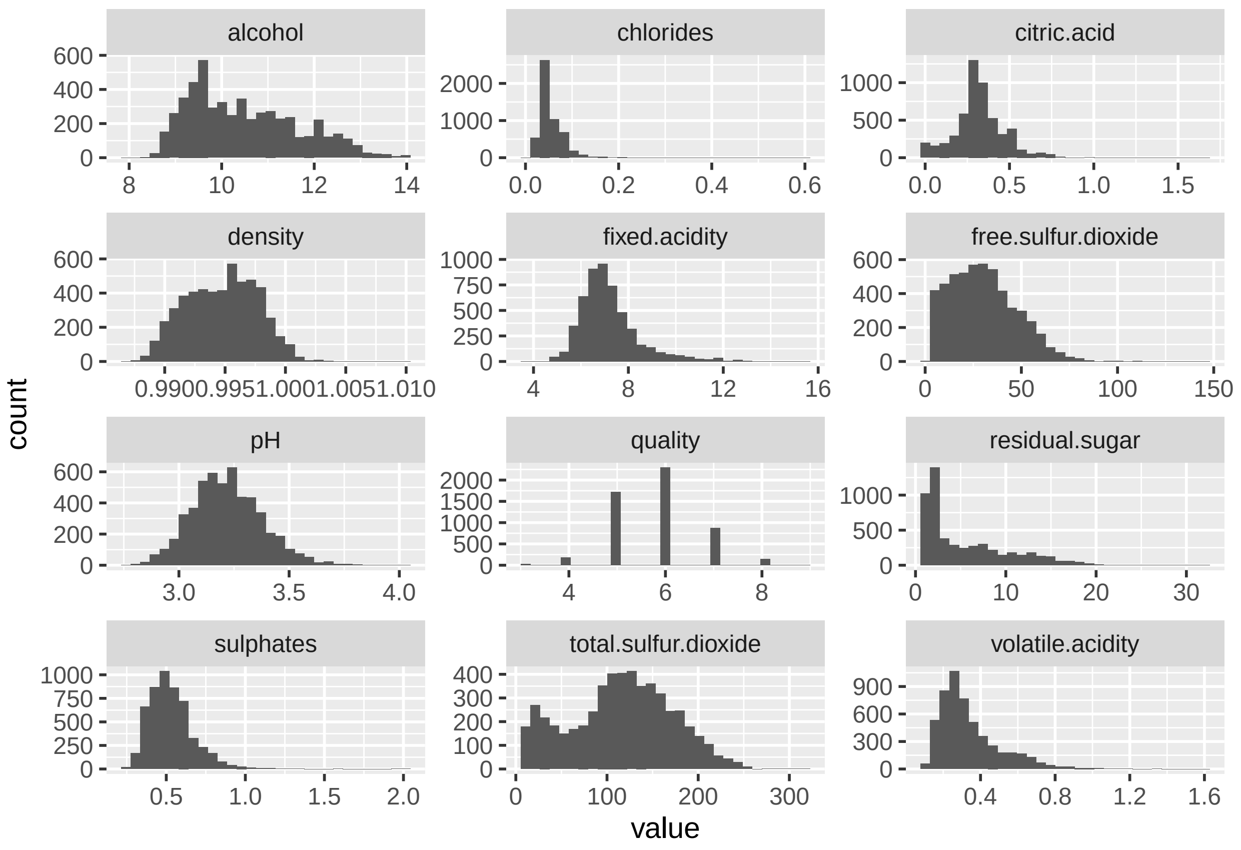


Figure 1: Histograms of all the variables in the train set with the exception of the type variable that we created to keep track of white/red status.

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i}(\hat{y}_i - y_i)^2}$$

which is a simple way to measure the magnitude of the prediction error and has the advantage of being on the same scale as the data. We will use the following function to calculate $RMSE$.

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

We can also model this data with classification, in which we choose a cutoff within the `quality` response to designate values above the cutoff as "Good" and values below as "NotGood". We will need a different loss function for classification models, and we will use the F measure, or

$$F - score(x, \hat{x}) = \frac{1}{\frac{1}{1+\beta^2}\frac{1}{recall_x} + \frac{1}{1+\beta^2}\frac{1}{precision_x}}$$

where x is the true value and $\hat{x}$ is the predicted value. In our case, we will use a value of $\beta = 1$ for the weight, so we are considering precision and recall to be equally important. We could have also chosen to use the area under the precision vs. recall curve, or AUC or Cohen's Kappa. The `Caret` package includes a function to calculate F measure or F-score and can also optimize models using that metric, as we will see in the section on classification models.

Methods

We begin with Regression analysis and then move to Classification.

Regression - Single Value Model

If we predict a single value for all of the wine quality values, the minimum error will be the mean of the `quality` score for the entire `train` set.

```
mu <- mean(train$quality)
```

Then the error is

```
RMSE(test$quality, mu)
```

```
## [1] 0.8400891
```

```
results <- tibble(Method = "Just the average",
                  Error = paste0("RMSE: ",RMSE(test$quality, mu)))

knitr::kable(results)
```

Method	Error
Just the average	RMSE: 0.840089109456635

This basic model actually performs fairly well in terms of $RMSE$, but of course it does a terrible job of predicting the best and worst quality scores.

Regression - Linear Model

Since a single value does not perform well at the ends of the quality scale, perhaps we can improve the performance by computing a linear combination of the available predictors, including wine type. We can use the `caret` package to create this initial model very quickly and easily.

```
fit <- train(quality~., data = train, method="lm")
y_hat <- predict(fit, test)
RMSE(test$quality, y_hat)
```

```
## [1] 0.7102523
```

```
results <- results %>%
  add_row(Method = "Basic Linear Model, All Predictors",
          Error = paste0("RMSE: ", RMSE(test$quality, y_hat)))
```

This is not a bad result, and does offer an improvement over a single value prediction, but if we take a look at the prediction, `y_hat`, versus the values in `test$quality`, we see that the linear predictor doesn't perform well at the ends of the `quality` scale, either, and those values are the ones we would most like to predict correctly. Additionally, if we read the documentation for linear prediction with `caret`, we find that there are multiple pre-processing options. We can center, scale, replace predictors with the Principal Components via Principal Component Analysis (PCA), or replace with independent components. We can examine how pre-processing might improve our results.

```
qplot(test$quality, y_hat)
```

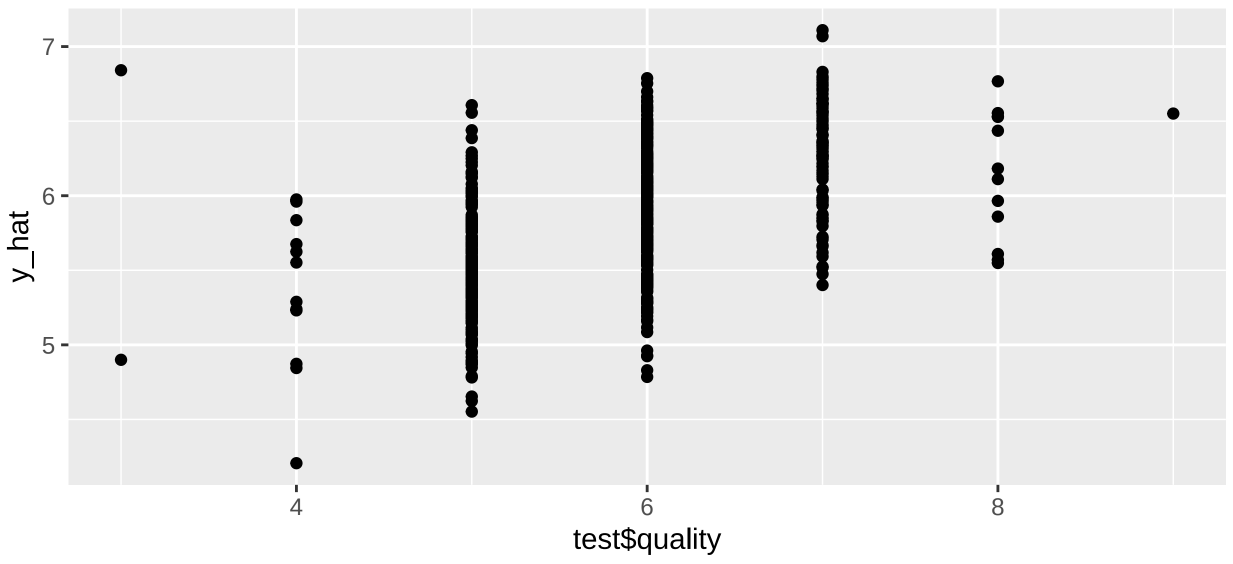


Figure 2: Plot of the linear model prediction for wine quality in the test set against the actual wine quality value.

```r
# Re-do linear model with preprocessing
# First create the parameters for pre-processing by
  # subtracting the column means, dividing by standard deviation, and
  # computing the principal components
preProc <- train %>%
  select(-quality) %>%
  caret::preProcess( . , method = c("center", "scale", "pca"))

# Create transformed test and train sets
train_transformed <- predict(preProc, train)
test_transformed <- predict(preProc, test)

# Re-train the linear model on the transformed data
fit <- train(quality~., data = train_transformed, method="lm")
y_hat <- predict(fit, test_transformed)
RMSE(test_transformed$quality, y_hat)
```

```
## [1] 0.7101706
```

With the transformation of our data via centering, scaling, and PCA we did not improve the performance of the linear model. However, we see that the performance was still best in the mid-range of actual quality scores and poorest in the extremes of the range.

```r
test_transformed %>% select(quality) %>%
  mutate(factored_quality = as_factor(quality), y_hat = y_hat) %>%
  group_by(factored_quality) %>%
  summarize(RMSE = mean(abs(quality - y_hat))) %>%
  ggplot(aes(factored_quality, RMSE)) +
    geom_point()
```

Regression - Multiple Models with Caret::train

The caret package allows us to test many models simultaneously. We can examine how several models predict our quality score, both without pre-processing and with pre-processing. In the following code, we predict quality using the Generalized Linear Model (glm), K-Nearest Neighbors (knn), Generalized Additive Model - Loess (gamLoess), Random Forest (rf), and Multivariate Adaptive Regression Spline (earth). Please note that if you run this code, you may need to install additional packages, but R should prompt you with installation instructions.

```r
models <- c("glm", "knn", "gamLoess", "rf", "earth")
fits <- lapply(models, function(model){
  print(model)
  train(quality ~ ., method = model, data = train)
})
```

```
## [1] "glm"
## [1] "knn"
## [1] "gamLoess"
## [1] "rf"
## [1] "earth"
```

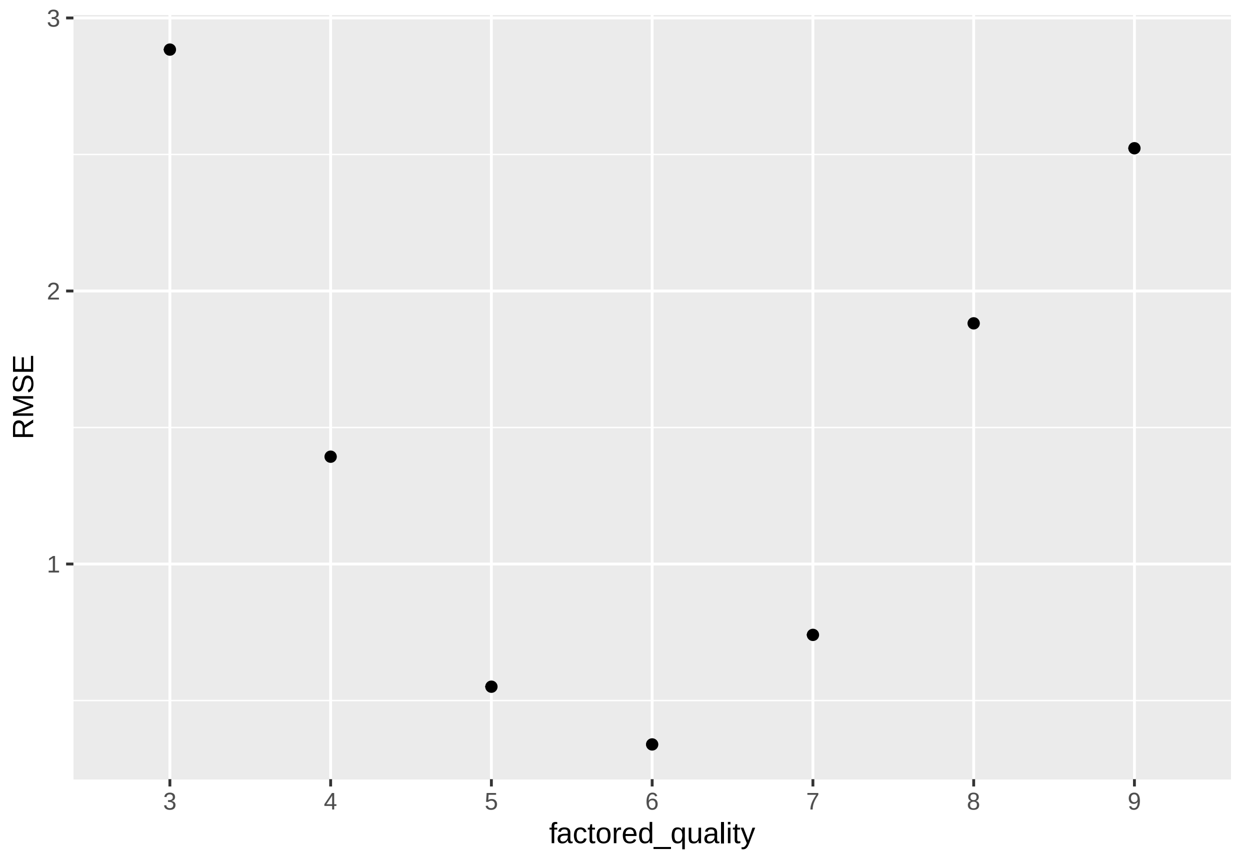


Figure 3: Linear model of transformed test and train sets, RMSE vs factorized quality. Shows best performance in the middle range and poor performance with both low and high actual quality scores.

```
## Loading required package: earth

## Loading required package: Formula

## Loading required package: plotmo

## Loading required package: plotrix

## Loading required package: TeachingDemos
```

```
predictions <- sapply(fits, function(fit){
  predict(fit, test)
})

i <- seq(1,5)
errors <- sapply(i, function(i){
  RMSE(predictions[,i], test$quality)
})

knitr::kable(data.frame(RMSEs = errors, Models = models))
```

RMSEs	Models
0.7102523	glm
0.7583807	knn
2.0682278	gamLoess
0.5542343	rf
0.7119383	earth

```
# Add the best model results to our overall results
results <- results %>%
  add_row(Method = "Random Forest Model, No Pre-processing",
          Error = paste0("RMSE: ", errors[models == "rf"]))
```

The Random Forest was by far the best performer, with the lowest root mean square error, $RMSE$, in our list of regression-appropriate models.

Regression - Preprocessed Random Forest Model

We can further tune the model, this time with cross-validation and hyper-parameter tuning. We can go through this process twice, once with no pre-processing and then with pre-processing to center and scale the data and then transform with Principal Component Analysis.

```
# Do internal cross-validation at 9:1 train:test ratio
control <- trainControl(method = "cv", number = 10, p = 0.9)
# Number of randomly selected predictors to use
grid <- data.frame(mtry = c(2,5,7,9,12))

fit <- train(quality ~ .,
             method = "rf",
```

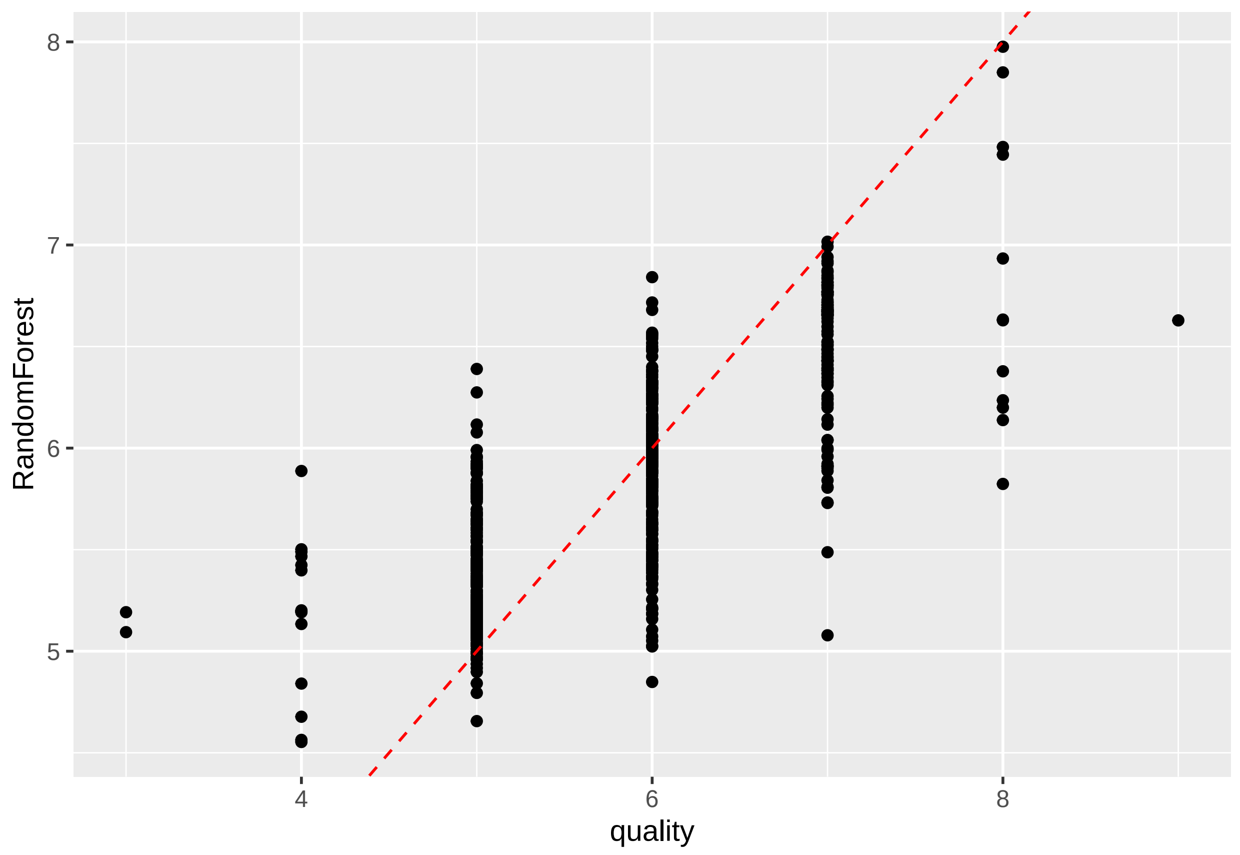


Figure 4: Quality Score vs the Random Forest prediction. The very lowest and very highest actual quality scores were not predicted accurately, but the overall performance was much improved compared to the linear model. The identity line is shown in red, and tells us that our Random Forest model is regressing toward the mean, with a shallower slope than the true relationship.

```r
               ntree = 150,
               trControl = control,
               tuneGrid = grid,
               data = train)
# Predict with the tuned fit
y_hat <- predict(fit, test)


# Add the best model results to our overall results
results <- results %>%
  add_row(Method = "Random Forest Model, No Pre-processing, Tuned",
          Error = paste0("RMSE: ", RMSE(test$quality, y_hat)))

# Store the plot of hyperparameter optimization
p1 <- ggplot(fit) + ggtitle("Regression, RF")
```

```r
# Create a transformed train set and transform the hold-out set
# using that information
preProc <- train %>%
  select(-quality) %>%
  caret::preProcess( . , method = c("center", "scale", "pca"))

# Create transformed test and train sets
train_transformed <- predict(preProc, train)
test_transformed <- predict(preProc, test)

# Maximum number of predictors is 9
grid <- data.frame(mtry = c(2,5,7,9))

# train and tune the model
fit_transformed <- train(quality ~ .,
                         method = "rf",
                         ntree = 150,
                         trControl = control,
                         tuneGrid = grid,
                         data = train_transformed)

# again, predict quality scores with the tuned fit
y_hat <- predict(fit_transformed, test_transformed)

# Add the best model results to our overall results
results <- results %>%
  add_row(Method = "Random Forest Model, Pre-processed, Tuned",
          Error = paste0("RMSE: ", RMSE(test$quality, y_hat)))

# Store the plot of hyperparameter optimization
p2 <- ggplot(fit) + ggtitle("Regression, RF, Center, Scale, PCA")
```

```r
# Plot the two tuning
gridExtra::grid.arrange(p1,p2,nrow=1)
```

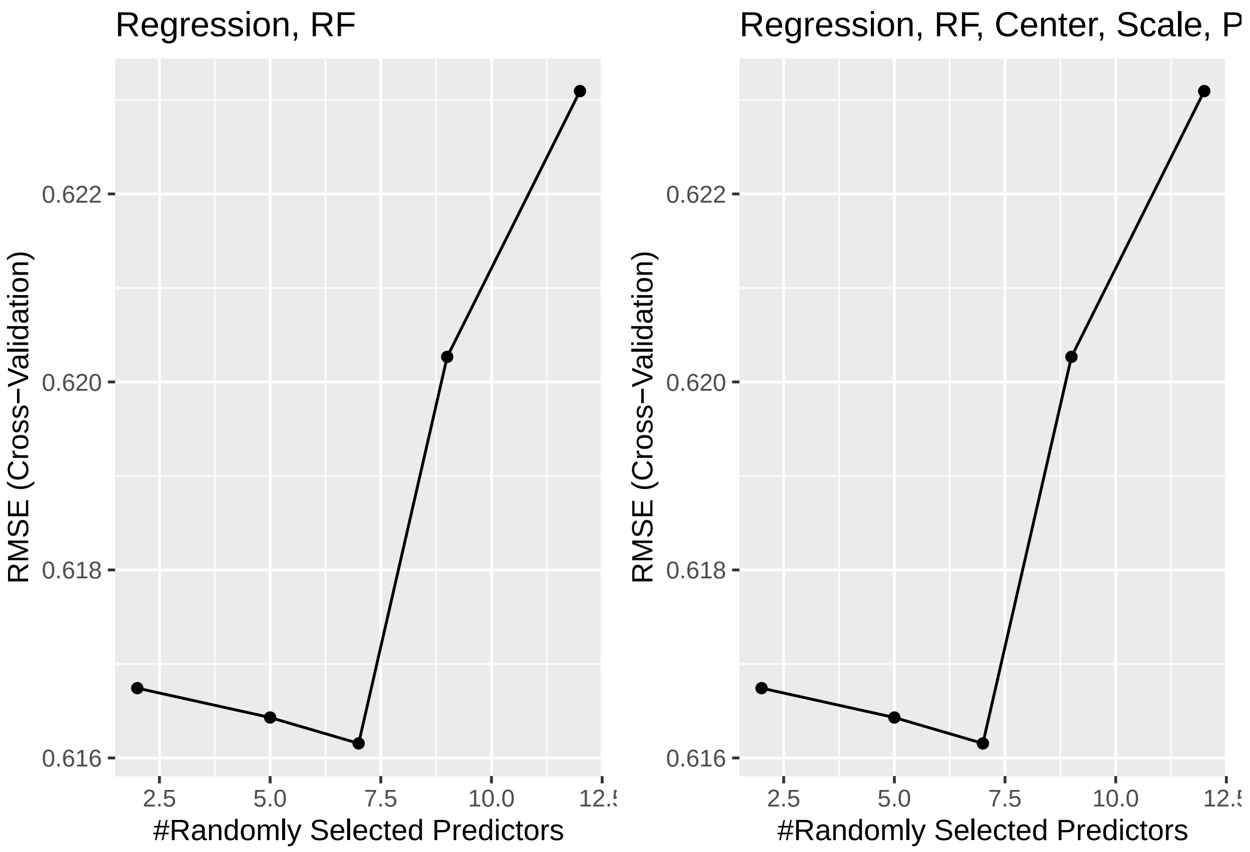


Figure 5: Random Forest Regression tuning, with and without pre-processing.

```r
rm(p1,p2)
```

We see that pre-processing the data and reducing our number of predictors to nine does not really improve the results. We get the best results with just seven randomly selected predictors either way, and our $RMSE$ is approximately the same, as well.

We will now move to the classification approach.

Classification Approach

We can also consider an approach to predicting the quality scores of the wines in our data set with categorical tags rather than attempting to predict the actual score. In practice, what we care about is the ability to predict if a wine is good or not. We could make the problem simpler by just predicting binary categories, ie the wine is either "good" or "not good".

The question then becomes how we assign the labels. If we create a variable, `good`, and assign a value of 1 when a wine is "good" and 0 when it is not, we still have to decide where our cutoff should be. One possible approach is to use a modified Tukey Outlier scheme to choose a quality score that represents "good" wines.

```r
# Third quartile, or 75% of data is less than this value, less 1.5
# inter-quartile range
max_avg_quality <- quantile(train_transformed$quality, 0.75) +
                    1.5 * IQR(train_transformed$quality)
max_avg_quality
```

```
## 75%
## 7.5
```

How many wines in the `train` set fall into the proposed "good" category? This question is important to establish prevalence and will change our approach to the problem depending on how prevalent "good" wines are.

```r
mean(train$quality >= max_avg_quality)
```

```
## [1] 0.02946768
```

This means that about three percent of wines in our development training set are "good" by this measure. Do we really just want to identify just the top three percent of wines, though? Perhaps we would get more utility from identifying the top 25%, instead. So we aren't looking for true outliers, just some "pretty good" wines. Let's try a modified Tukey outlier approach, in which we will just look at one inter-quartile range above the 75th percentile.

```r
max_avg_quality <- quantile(train_transformed$quality, 0.75) +
                    1.0 * IQR(train_transformed$quality)
max_avg_quality
```

```
## 75%
##   7
```

```r
mean(train$quality >= max_avg_quality)
```

```
## [1] 0.1965779
```

Now we are trying to identify the top twenty percent of wines, and that seems more reasonable. We can now create the categorical variables to represent this cutoff choice for both the train and test sets and also examine the proportion of good wines within the red and white sub-categories.

```r
train <- train %>%
  mutate(categorical_quality = factor(ifelse(quality >= max_avg_quality,
                                             "Good", "NotGood")))
test <- test %>%
  mutate(categorical_quality = factor(ifelse(quality >= max_avg_quality,
                                             "Good", "NotGood")))

# Examine proportions
  train %>% group_by(red) %>% summarize(avg = mean(categorical_quality == "Good"))
```

```
## # A tibble: 2 x 2
##     red   avg
##   <dbl> <dbl>
## 1     0 0.215
## 2     1 0.141
```

We do have different proportions of "good" wines in our red and white categories. White wine is about 1.5 times more likely to be "good" than red wine in our training set. This may cause problems as we go forward and we might want to reconsider combining the two variants into the same dataset. However, we will go forward with the analysis using the one-hot-encoded `red` variable and see if we can come up with a model that will predict good wines, independent of grape color.

We also need to note here that we are changing our entire modeling paradigm from regression analysis to classification, so we need to identify our results accordingly. For our regression models, we used $RMSE$ as our error measure. For classification, particularly with the low prevalence of our positive category, "good", we should use something like the area under the precision recall curve, AUC, or the F measure. As discussed in the introduction, we have chosen F measure or F-score and both training and results will be reported accordingly.

Classification - Guessing Model

As a baseline classification model, we can make a random guess for each observation in the test set with a random sample of "good" or "not good" with underlying assumed proportions for each category.

```r
# Set the seed for reproducible sampling
suppressWarnings(set.seed(7, sample.kind = "Rounding"))

# Define list of possible underlying "good" probability
probabilities <- seq(0.01,0.99,length.out = 10)

# Calculate F measure with beta = 1 for each probability
F1s_guess <- sapply(probabilities, function(p){
  # factorize the character vector in train
  ref <- train$categorical_quality

  # Calculate a prediction for category, being careful to list "Good" first
  y_hat <- sample( x = c("Good", "NotGood"),
          size = length(train$categorical_quality),
          replace = TRUE,
          prob = c(p, 1-p)) %>%
    factor(levels = levels(ref))

  # Calculate the F measure for the set of y's we sampled with prob = p
  F_meas(data = y_hat, reference = ref, relevant = "Good")
})

qplot(probabilities,F1s_guess)
```

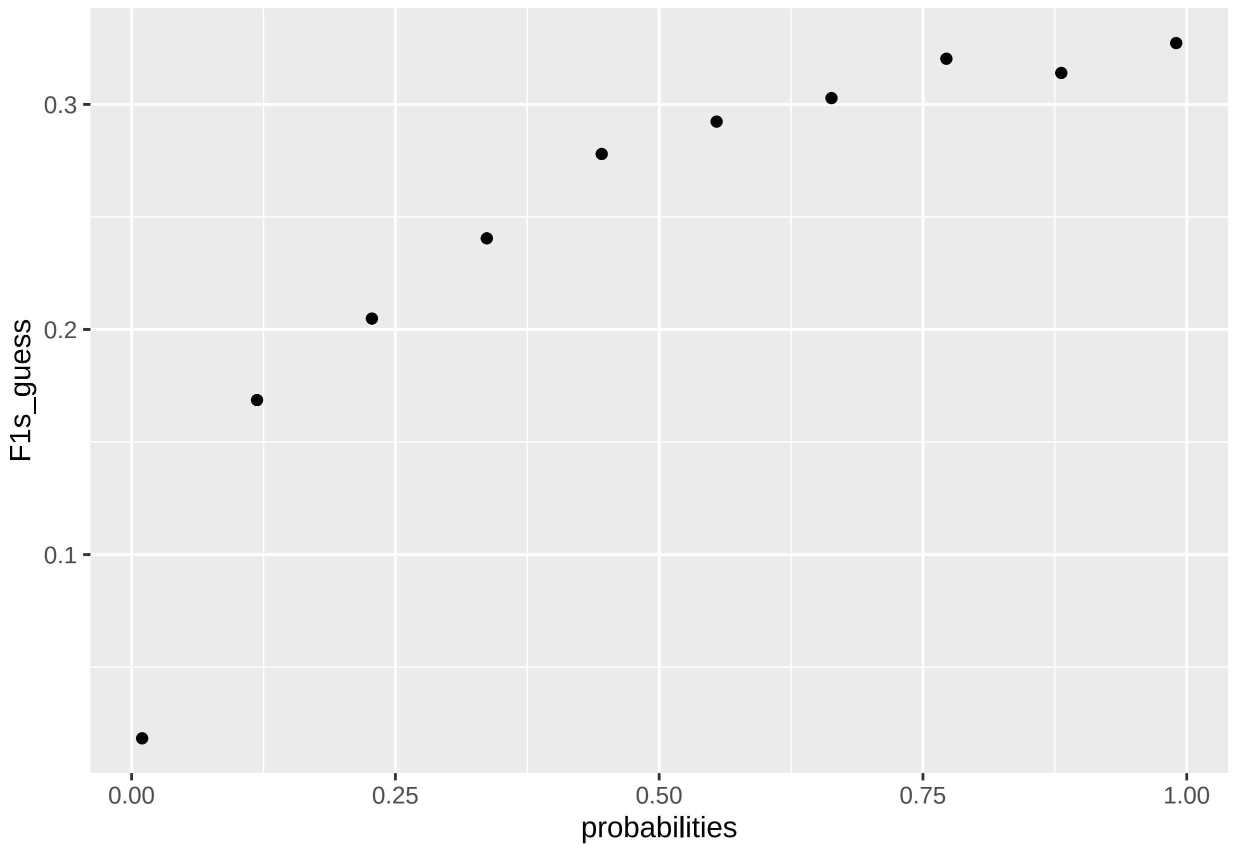


Figure 6: Random sample probability of Good wines versus the resulting F-score for the prediction with that probability of Good.

```
# And the maximum value of F1 is
max(F1s_guess, na.rm = TRUE)
```

```
## [1] 0.3272377
```

```
# at a probability of "Good" equal to
p <- probabilities[which(F1s_guess == max(F1s_guess, na.rm = TRUE))]
p
```

```
## [1] 0.99
```

So we need to predict that all of our wines will be "Good" in order to capture the ones that really are good. However, if we look at the plot of probabilities vs. F-score, we see that a prediction of about 0.75 "Good" would give us nearly the same F-score as the maximum, all of which is consistent with our known proportion of "Good" wines in our training set based on the cutoff we selected to define the categories. We note that whichever probability of "Good" wines we use, our F-score is dismal. We will select the second-best F-score in the guessing model as our probability to make predictions for the test set.

```
p <- probabilities[8]
# factorize the character vector in test
ref <- test$categorical_quality

# Calculate a prediction for category, being careful to list "Good" first
y_hat <- sample( x = c("Good", "NotGood"),
                 size = length(test$categorical_quality),
                 replace = TRUE,
                 prob = c(p, 1-p)) %>%
         factor(levels = levels(ref))

# Calculate the F measure for the set of y's we sampled with prob = p
F_meas(data = y_hat, reference = ref, relevant = "Good")
```

```
## [1] 0.299639
```

```
# Add the categorical guess results to the results tibble
results <- results %>%
  add_row(Method = "Classification Guessing",
          Error = paste0("F1: ",
                   F_meas(data = y_hat, reference = ref, relevant = "Good")))
```

Classification - Cutoffs Model

If we go back to our initial data exploration, we can guess that `alcohol` content or `volatile.acidity` might be good choices for single predictors that we could use to predict the `quality`. This is because `alcohol` is the most positively correlated with `quality`, while `volatile.acidity` is among the most strongly negatively correlated with `quality` while not simultaneously being correlated with `alcohol` content. So, we can attempt a very simple model using each one to predict `quality` with just a cutoff value, where the categorical quality on one side of the cutoff is "Good" and on the other side it is "NotGood".

```
# Create a range of cutoff values, a sequence of 10 between the min and max
# values of the train set alcohol content values
cutoffs <- seq(min(train$alcohol),max(train$alcohol),length.out = 10)

# For each alcohol cutoff value, calculate F1
F1s_cutoffs_Alcohol <- sapply(cutoffs, function(c){
  ref <- train$categorical_quality
```

```
    # Predict wine is good if alcohol is above cutoff
    y_hat <- ifelse(train$alcohol >= c, "Good", "NotGood") %>%
      factor(levels = levels(ref))

    # Calculate the F measure with beta = 1
    F_meas(data = y_hat, reference = ref)
  })

  p1 <- qplot(cutoffs,F1s_cutoffs_Alcohol)

  # And the maximum value of F1 is
  max(F1s_cutoffs_Alcohol, na.rm = TRUE)
```

```
## [1] 0.4963457
```

```
  # at an alcohol cutoff for "Good" equal to
  alcohol_cutoff <- cutoffs[which(F1s_cutoffs_Alcohol ==
                                  max(F1s_cutoffs_Alcohol, na.rm = TRUE))]
  alcohol_cutoff
```

```
## [1] 10.68889
```

```
  # Predict quality for the test set based on the alcohol cutoff obtained from
  # the train set
  ref <- test$categorical_quality
  y_hat <- ifelse(test$alcohol >= alcohol_cutoff, "Good", "NotGood") %>%
    factor(levels = levels(ref))
  # Calculate the F measure on the test set with beta = 1
  F_meas(data = y_hat, reference = ref, relevant = "Good")
```

```
## [1] 0.4971751
```

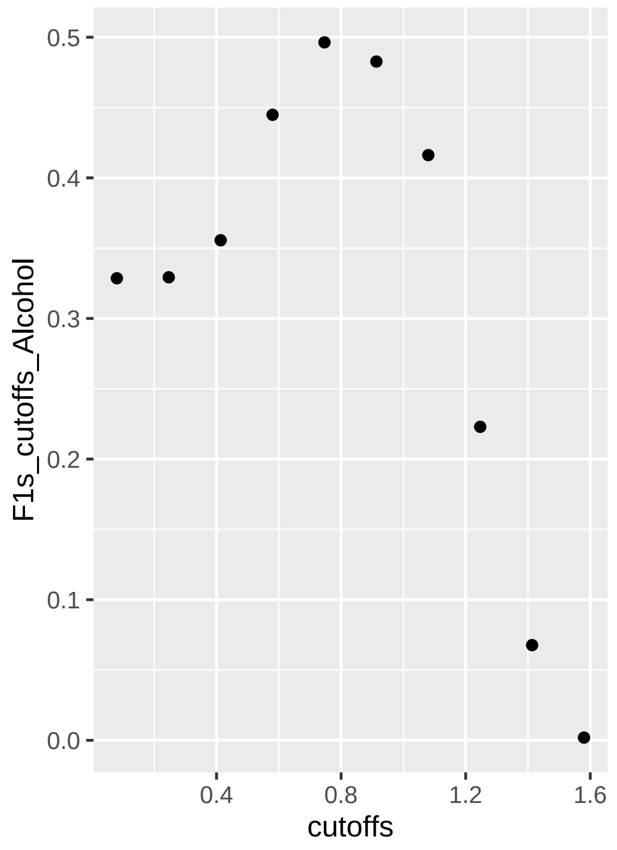
```
  # Add the categorical guess results to the results tibble
  results <- results %>%
    add_row(Method = "Classification Cutoff by Alcohol Content",
            Error = paste0("F1: ",
                      F_meas(data = y_hat, reference = ref, relevant = "Good")))
```

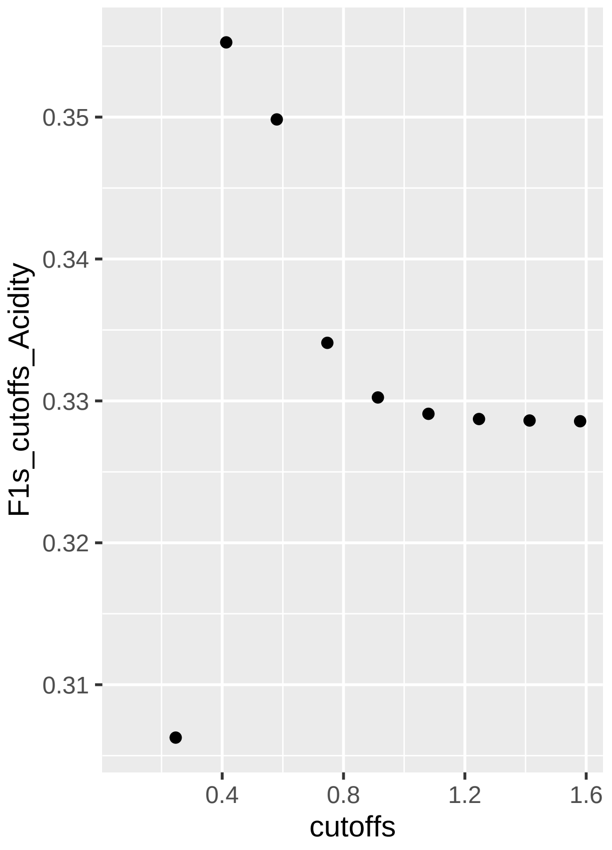
And we can do the same with `volatile.acidity` to get

```
## [1] 0.3552658
```

```
## [1] 0.4133333
```

```
  gridExtra::grid.arrange(p1,p2,nrow=1)
```





```
rm(p1,p2)
```

```
# Add the categorical guess results to the results tibble
results <- results %>%
  add_row(Method = "Classification Cutoff by Volatile Acidity",
          Error = paste0("F1: ",
                    F_meas(data = y_hat, reference = ref, relevant = "Good")))
```

```
kable(results)
```

Method	Error
Just the average	RMSE: 0.840089109456635
Basic Linear Model, All Predictors	RMSE: 0.710252325270849
Basic Linear Model, Centered, Scaled, PCA	RMSE: 0.710170557044147
Random Forest Model, No Pre-processing	RMSE: 0.55423432825903
Random Forest Model, No Pre-processing, Tuned	RMSE: 0.556251127549885
Random Forest Model, Pre-processed, Tuned	RMSE: 0.577027610912259
Classification Guessing	F1: 0.299638989169675
Classification Cutoff by Alcohol Content	F1: 0.497175141242938
Classification Cutoff by Volatile Acidity	F1: 0.355265757755384

Classification - Caret Package Models

We have evaluated some very basic approaches to classifying the wines into “Good” and “NotGood” categories. We can now explore how the `Caret` package can help us to quickly evaluate the performance of several different models at once.

```r
models <- c("lda", "naive_bayes", "knn", "gamLoess", "qda", "rf")
fits <- lapply(models, function(model){
  # print(model)  # progress for command line
  train %>% select(-quality) %>%
    train(categorical_quality~., data = ., method= model, metric = "F1")
})

predictions <- sapply(fits, function(fit){
  factor(predict(fit, test, type = "raw"), levels = levels(train$categorical_quality))
})

predictions <- data.frame(predictions)
colnames(predictions) <- models

F1s <- sapply(predictions, function(y_hat){
  F_meas(data = factor(y_hat,
                       levels = levels(train$categorical_quality)),
         reference = test$categorical_quality)
})

kable(data.frame(Model = models, F_1 = F1s))
```

	Model	F_1
lda	lda	0.4153005
naive_bayes	naive_bayes	0.5482625
knn	knn	0.2682927
gamLoess	gamLoess	0.4519774
qda	qda	0.5441696
rf	rf	0.7113402

Again, Random Forest is by far the best performer. We can do the same analysis with pre-processing, as well, and we get the following results

	Model	F_1
lda	lda	0.4408602
naive_bayes	naive_bayes	0.4536082
knn	knn	0.5279188
gamLoess	gamLoess	0.4357542
qda	qda	0.5134100
rf	rf	0.6451613

Again, Random Forest gives the best result with the transformed data, but didn’t perform as well as the full set of predictors, or the unpreprocessed `train` set. The final model training results were

```r
kable(results)
```

Method	Error
Just the average	RMSE: 0.840089109456635
Basic Linear Model, All Predictors	RMSE: 0.710252325270849
Basic Linear Model, Centered, Scaled, PCA	RMSE: 0.710170557044147
Random Forest Model, No Pre-processing	RMSE: 0.55423432825903
Random Forest Model, No Pre-processing, Tuned	RMSE: 0.556251127549885
Random Forest Model, Pre-processed, Tuned	RMSE: 0.577027610912259
Classification Guessing	F1: 0.299638989169675
Classification Cutoff by Alcohol Content	F1: 0.497175141242938
Classification Cutoff by Volatile Acidity	F1: 0.355265757755384

If we further tune the Random Forest algorithm and apply it to the entire `full_train` set, we can then predict values for the `hold_out` set, which will be our final result.

Results

```r
# Calculate the quality categories for the full train set, again with the
# modified Tukey outlier method.
# Third quartile, or 75% of data is less than this value, less 1.0 *
# inter-quartile range
max_avg_quality <- quantile(full_train$quality, 0.75) +
                    1.0 * IQR(full_train$quality)

# If quality is a high outlier, call it good
# others are not good
full_train <- full_train %>%
  mutate(categorical_quality = factor(ifelse(quality >= max_avg_quality,
                                             "Good", "NotGood")))

# Assign the same categorical values for the test set using values calculated
# with train set
hold_out <- hold_out %>%
  mutate(categorical_quality = factor(ifelse(quality >= max_avg_quality,
                                             "Good", "NotGood")))

# Save the predictions added to train control
control <- trainControl(method = "cv", number = 10, p = 0.9,
                        savePredictions = TRUE)

# Randomly select predictors to use
grid <- data.frame(mtry = c(3,5,7,9,12))

# train and tune the model
fit <- full_train %>%
  select(-quality) %>%
  train(categorical_quality ~ .,
        method = "rf",
          ntree = 150,
          trControl = control, # use previously defined cross-validation scheme
          tuneGrid = grid, # try predicting with different rand. selected pred's
          metric = "F1", # select best performance based on F measure
          data = .)

# again, predict quality scores with the tuned fit
y_hat <- predict(fit, hold_out, type = "raw")

F_final <- F_meas(data = factor(y_hat,
                        levels = levels(full_train$categorical_quality)),
         reference = hold_out$categorical_quality)

# Add the final F measure to the results tibble
results <- results %>%
  add_row(Method = "Classification Random Forest Final",
          Error = paste0("F1: ", F_final))
kable(results)
```

Method	Error
Just the average	RMSE: 0.840089109456635
Basic Linear Model, All Predictors	RMSE: 0.710252325270849
Basic Linear Model, Centered, Scaled, PCA	RMSE: 0.710170557044147
Random Forest Model, No Pre-processing	RMSE: 0.55423432825903
Random Forest Model, No Pre-processing, Tuned	RMSE: 0.556251127549885
Random Forest Model, Pre-processed, Tuned	RMSE: 0.577027610912259
Classification Guessing	F1: 0.299638989169675
Classification Cutoff by Alcohol Content	F1: 0.497175141242938
Classification Cutoff by Volatile Acidity	F1: 0.355265757755384
Classification Random Forest Final	F1: 0.685185185185185

```r
confusionMatrix(predict(fit, hold_out, type = "raw"),
              as_factor(hold_out$categorical_quality))$overall[["Accuracy"]]
```

```
## [1] 0.8955453
```

```r
# Plot precision/recall curve
fit$pred %>% select(pred, obs, mtry) %>%
  group_by(mtry) %>%
  mutate(recall = sensitivity(pred, obs),
         precision = precision(pred, obs)) %>%
  ggplot(aes(recall, precision, label = mtry)) +
  geom_line() +
  geom_point() +
  geom_text(hjust = 0.3, vjust = 1.5) #+
```

```r
#    scale_x_continuous(limits = c(0.56,0.59)) #+
#    scale_y_continuous(limits = c(0.7,1))
```

The automatically selected best tune for this Random Forest model was

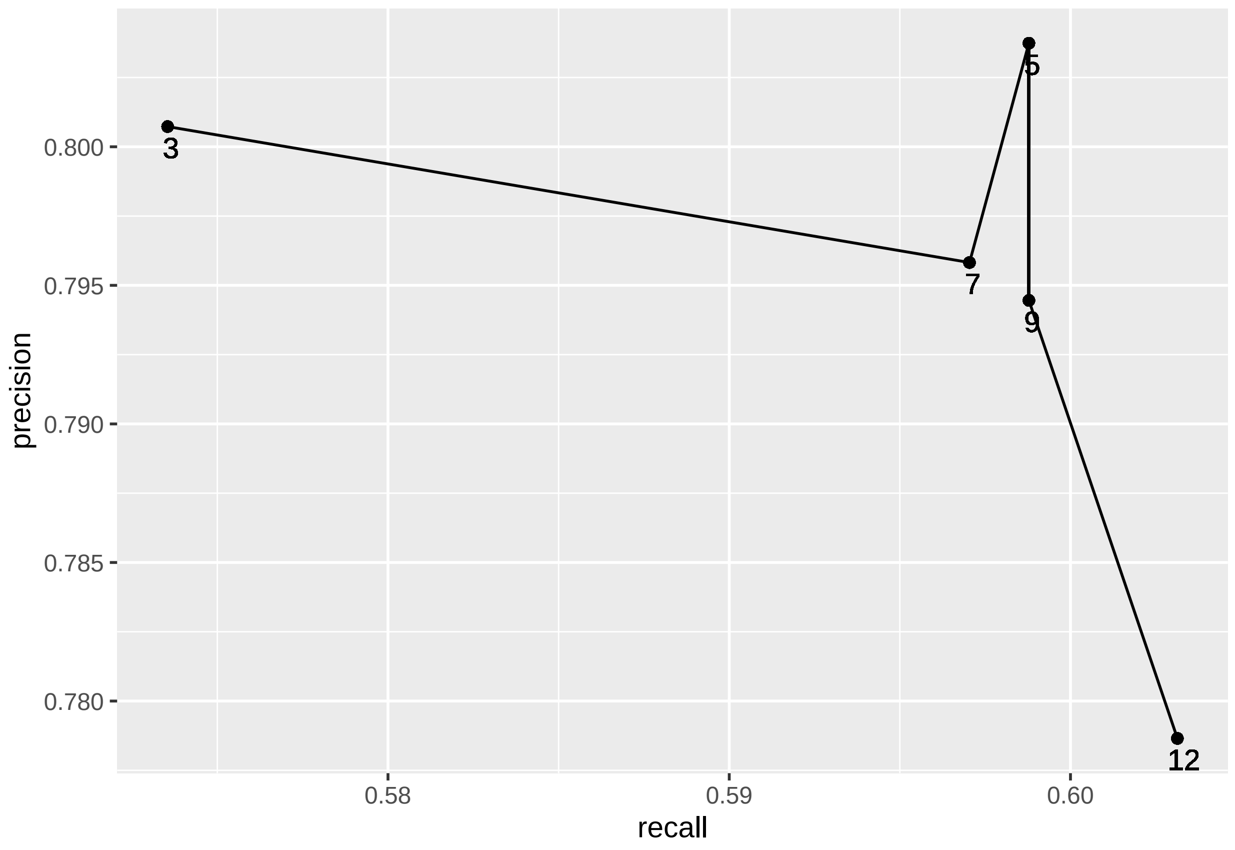


Figure 7: Precision Recall Curve for Final Random Forest Model. Please note that the axes are not on the usual 0 to 1 range, but are zoomed in so that the data is readable. The number of randomly selected predictors does not strongly affect the precision or recall.

```
fit$bestTune
```

```
##   mtry
## 2    5
```

But we should note that changing `mtry` didn't have a large effect on either precision or recall.

Our remaining outstanding question was if it was appropriate to combine the red and white wine data into one dataset. With our best-performing model, we can simply examine how important the `red` variable was in the final model.

```
varImp(fit)
```

```
## rf variable importance
##
##                      Overall
## alcohol               100.00
## density                61.85
## volatile.acidity       49.77
## sulphates              47.53
## chlorides              47.09
## residual.sugar         46.89
## total.sulfur.dioxide   44.87
## free.sulfur.dioxide    44.10
## pH                     44.06
## citric.acid            40.17
## fixed.acidity          37.34
## red                     0.00
```

We see that `red` has the lowest importance by a large margin. This means that the color of grape was not predictive in this model in terms of quality.

Finally, we can compare how the model compared to the true data.

```
# Predict the probability of a wine being good
p_hat <- predict(fit, hold_out, type = "prob")

p1 <- hold_out %>%
        mutate(p_Good = p_hat$Good) %>%
        ggplot(aes(x=alcohol, y=volatile.acidity, color=p_Good)) +
        geom_point(alpha = 0.5) +
        scale_color_gradient2(low = 'red', mid = 'white',
                              high = 'blue', midpoint = 0.5)

p2 <- hold_out %>%
        ggplot(aes(x=alcohol, y=volatile.acidity,
                   color = categorical_quality)) +
        geom_point(alpha = 0.5) +
        scale_color_manual(values = c("Good" = "blue", "NotGood" = "red"))

gridExtra::grid.arrange(p1,p2,nrow=2)
```

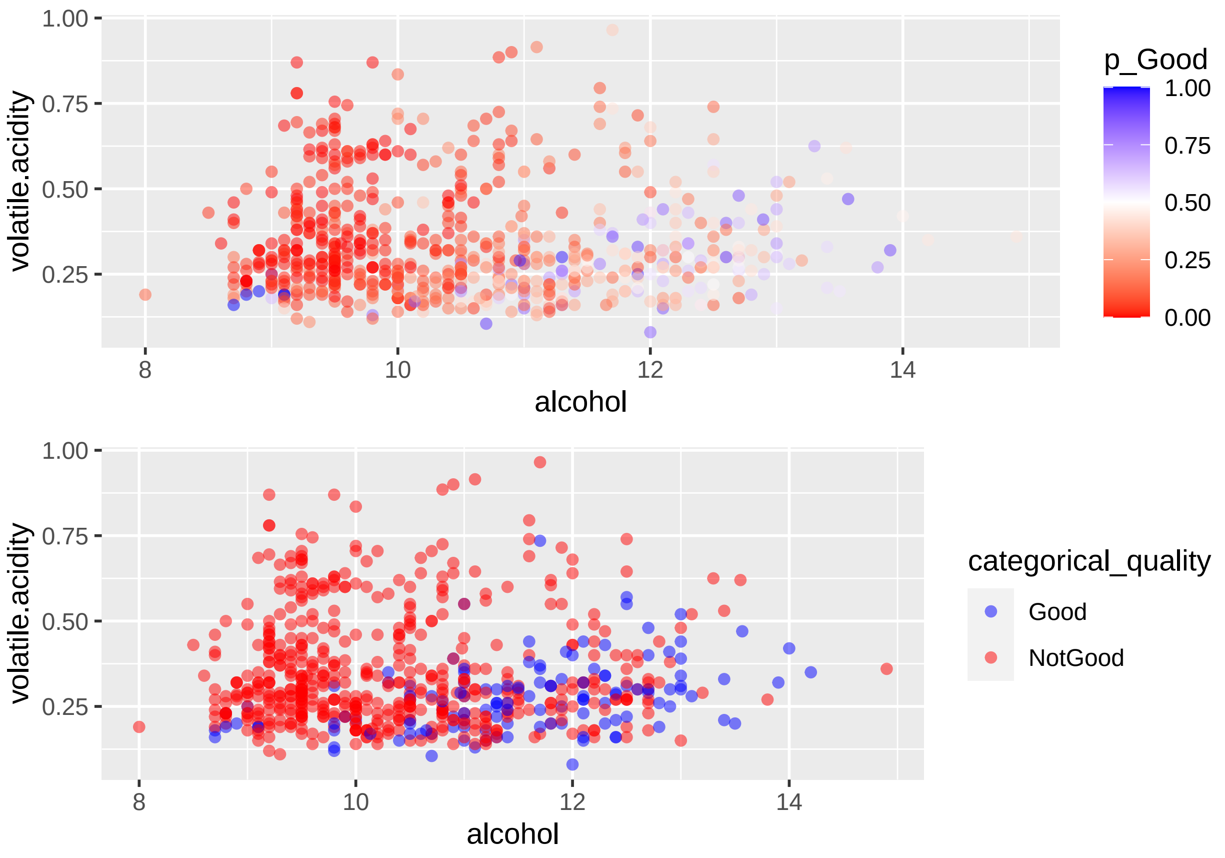


Figure 8: Comparison of the best model prediction of quality based on the two top variables that are not strongly correlated and the actual quality scores for the hold_out set.

```r
rm(p1,p2)
```

We see that the final Random Forest model did, in fact, capture the overall shape of quality distribution, at least for the two most important variables that are not strongly correlated with each other. Even some of the good wines in the low `alcohol` and low `volatile.acidity` area were captured fairly well. The area with muddled predictions was at the lower right, where `alcohol` content was high, but `volatile.acidity` was low.

Conclusion

We approached the problem from two directions, first as a regression problem and then as a classification problem. Our final results with Random Forest on a classification dataset provided us with nearly 90% overall accuracy and an F-score near 0.7. We were able to predict the basic shape of the quality score relative to the most important uncorrelated predictors.

Given that there were multiple predictors with intercorrelations, it was surprising that transforming the data through Principal Component Analysis did not improve the resulting predictions. There is certainly more work that could be done with pre-processing schemes, including identification of independent predictors. We could also tune the selection of the cutoff value.

There are also many more algorithms available to explore, such as conditional inference trees. Other analysts working on this dataset have had very good results with `ctree`, for instance, so exploration of recursive partitioning could prove interesting.