ST563 Final Project

Mana Azizsoltani

11 March, 2021

Contents

Introduction

There has been a push towards the implementation of different analytics solutions within the agriculture industry, especially in the realms of crop health and yield. These sort of analyses can help farmers to reduce waste and improve profits. In particular, the viticulture industry stands to benefit from such methods, being that the production of wine is multifaceted. Winemakers can work to optimize revenue by analyzing their input and output (crops and wines). Specifically, costs may be cut out if machine learning methods were able to predict the quality of wine based on different metrics, since wine producers would no longer have to hire expert wine tasters to determine wine quality.

In this study, our objective is to try and predict the quality of wine using a variety of statistical learning techniques. We will look at this problem from a regression standpoint as well as a classification standpoint. Using regression, we will try and predict the exact rating (0 to 10) of a particular wine. On the other hand, when we look at classification, we will try to classify a particular wine as either good or bad, taking wines rated from 0-5 as low quality wines and wines rated from 6-10 as high quality wines. We will be assessing the prediction accuracy of the following methods:

Table 1: ML Methods

Classification	**Regression**
Classification Tree	Boosted Tree
Random Forest	Random Forest
Support Vector Machine (SVM)	Multiple Linear Regression (MLR)
KNN Classification	KNN Regression
Logistic Regression	LASSO Regression

Required Libraries

To run the code for the project, the following libraries are required:

- `caret`: to do the heavy lifting of training and tuning the models
- `tidyverse`: for all the data reading and wrangling
- `knitr`: for rmarkdown table outputs
- `rmarkdown`: for output documents
- `corrplot`: for correlation structure visualization
- `leaps`: for subset selection
- `rpart`: fitting the basic classification tree
- `rpart.plot`: visualization of the classification tree
- `randomForest`: fitting random forest models
- `kernlab`: fitting the support vector machine
- `class`: fitting the k-nearest neighbor model
- `glmnet`: fitting the LASSO model
- `gbm`: fitting the boosted tree model

Data

The data set used is the `Wine Quality` dataset from the UCI Machine Learning Repository. The data set is composed of 1600 observations of different variants of the Portuguese "Vinho Verde" red wine. For each wine, various features of the wine were measured, including a measurement of quality, given by an expert wine taster. Our objective is to use the different features to predict the quality of the wine using different machine learning techniques.

Variable Descriptions

Each entry in the dataset represents the different metrics of the following attributes of a single type of wine.

- **fixed acidity**: the quantity of fixed acids found in the wines. The predominant fixed acids found in wines are tartaric, malic, citric, and succinic, all of which come from the grapes with the exception of the succinic acid, which comes from the yeast in fermentation.
- **volatile acidity**: the steam distillable acids present in wine, primarily acetic acid but also lactic, formic, butyric, and propionic acids. These acids generally come from the fermentation process.
- **citric acid**: added to the wine as a natural preservative or for acidity and tartness.
- **residual sugar**: the quantity of sugar left in the wine after the fermentation process.
- **chlorides**: the amount of salt in a wine.
- **free sulfur dioxide**: the amount of SO_2 that is not bound to other molecules.
- **total sulfur dioxide**: total amount of SO_2 in the wine. Sulfur Dioxide is used throughout all stages of the winemaking process to prevent oxidation and bacteria growth.
- **density**: density of the wine.
- **pH**: pH of the wine.
- **sulphates**: quantity of sulphates in the wine. Sulphates are used as a preservative.
- **alcohol**: alcohol content by volume.
- **quality**: score of quality of the wine given by expert tasters (score between 0 and 10).

Methods

Data Processing

Fortunately, the `wine` dataset that we worked with was very tidy; there was no missing values. We didn't standardize the data initially, but when training the support vector machine as well as the k-nearest neighbors models we centered and scaled the data.

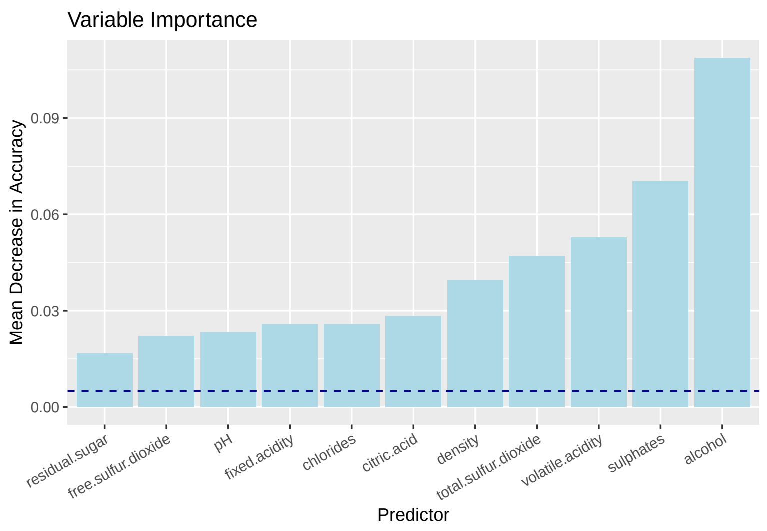
Cross Validation

As mentioned in the introduction, our focus is on the classification accuracy of machine learning methods. In particular, we will be using traditional cross-validation methods to assess the accuracy, sensitivity, and specificity of each of the models. We split the data into a training and test data set in order to later evaluate the model's prediction accuracy. For this project we used a 75/25 split, training the data on the 75% and testing the trained models on the withheld 25%. We will then repeat this process over 5 folds of the data, averaging the results.

The tree-based, LASSO, and K-nearest neighbors models require parameters to be tuned (more on those later). Since we used the `caret` package in R to fit all of our models, we used the tunes of the parameters that were deemed "best" by the `train()` function.

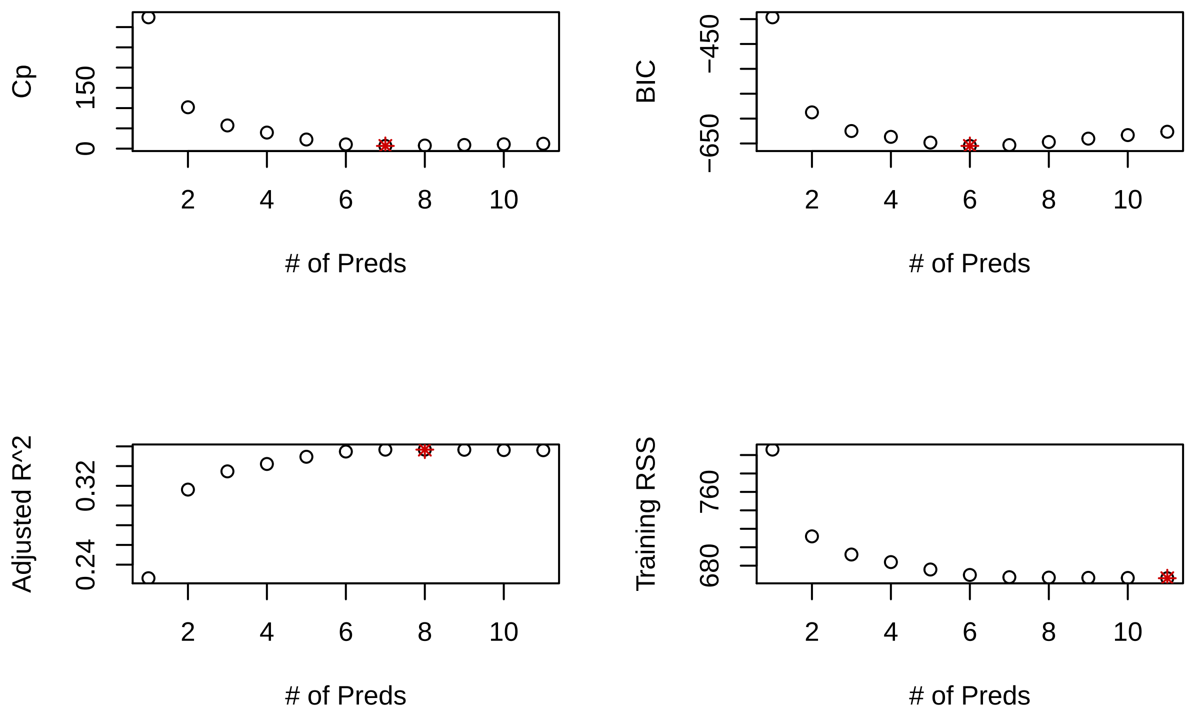
Variable Selection

Since we have two different tasks that we are trying to complete (classification and regression), we went about selecting the variables for each task differently. For the classification task, we ran an initial random forest model on the entire dataset with selected predictors. We then looked at the variable importance of the predictors to decide which variables we would use in our final models. Variable importance, within the context of machine learning, refers to how much a given model "uses" that variable to make accurate predictions. In other words, the more a model relies on a variable to make predictions, the more important it is for the model.



We decided to choose the variables with a variable importance over .005, leaving us with `alcohol`, `sulphates`, `volatile.acidity`, `density`, and `total.sulfur.dioxide`. The fact that density proved to be an important variable for classification was relatively surprising. One would think that the wine quality would be determined based on its attributes that are detectable by the human senses. For this same reason, alcohol, acidity, and sulphates made sense in terms of variable importance. Sulphates are essential for the preservation of the wine, an overly- or underly-acidic wine may be considered poor, and the alcohol content has a connection to the fermentation process, where the must (aka the grape juice before it undergoes fermentation) actually becomes wine.

When looking at the problem in terms of a regression context, we used best subsets selection for feature selection. Using the `leaps` library and `regsubsets` in R, we are able to compare all of the possible models from the given set of predictors. Through best subset selection, we can get the set of predictors for a model with any given amount of predictors. We looked at three different model criteria to compare models of different amounts of predictor variables: Mallow's Cp, BIC, and R^2_{adj}.



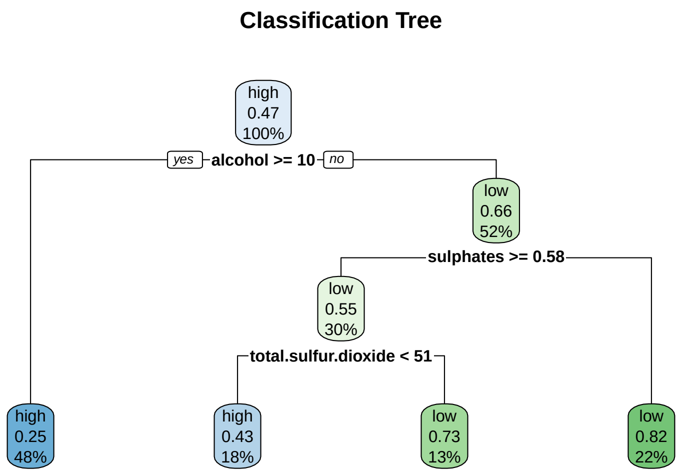
Although each of the three model criteria suggested a different number of predictor variables of a different number, they were all within the same region (between 6 and 8 predictors). Since they were so close, we adhered to the Occam's Razor and went with the simplest, six-variable model. This was the model favored by BIC, which gives the greatest penalty for complexity in models. The six variables used in the regression context are then `volatile.acidity`, `chlorides`, `total.sulfur.dioxide`, `pH`, `sulphates`, and `alcohol`. All of the variables from the classification selection appear here with the exception of density. In the regression context, `pH` was the variable that proved to be a rather unintuitive inclusion, since it is similar to density in the sense that it is not easily detected by the five human senses. The inclusion of these two variables in the model portray the importance of the chemical properties of a bottle of wine when it comes to quality.

Basic Tree

The basic classification or regression tree is based on partitioning the data into subgroups using simple binary splitting. Initially, all objects are considered a single group. Then, the group is binarily split into two subgroups based on the criteria of a certain variable. In the classification setting, we classify the observations in a specific region with majority vote, while in the context of regression the mean of the group is used for prediction. We want to grow the tree as big as we can, and then prune it back using cost-complexity pruning. This is done to not overfit the data, but pruning increases the bias. The pruning parameter needs to be tuned, which is done automatically in the `caret` package.

The advantage of using a basic tree is that it is easy to understand and has a good interpretability, which is not something that translates over to ensemble methods like the random forest. Additionally, the basic classification or regression tree is not very computationally expensive, unlike the random forest, boosting, and the support vector machine. We used the `caret` and `rpart` packages in R to fit our trees.

A visualization of the classification tree can be found below.



Random Forest

The random forest model is an ensemble tree-based method, which creates multiple trees from bootstrap samples and averages the results. Many bootstrap samples are created with replacement and then a classification tree is fitted on each bootstrap sample with a random subset of the predictors. Once a prediction has been made by all of the bootstrapped trees, the final classification is based on majority vote of the bootstrap predictions. Similarly, the prediction in the regression context is an average of all of the predictions of the bootstrapped trees.

The advantage for using an ensemble method over a regular classification or regression tree is that because there are many bootstrap samples being averaged together, there is less variance. This is similar to how the variance of the sample mean goes down as the sample size increases. Although it will probably increase our prediction accuracy, the random forest loses the interpretability that the basic trees have. Furthermore, the algorithm that is used to fit the random forest is very computationally expensive.

To train our model, we used the `randomForest` and `caret` packages in R. The maximum number of predictors for a bootstrap sample as well as the number of trees in the forest are both parameters that need tuning. Again, we will use the "best" tune, automatically given to us by the `caret` package.

Support Vector Machine

A support vector machine is a type of classification rule where it essentially maximizes the margin between groups by choosing the "line" with the widest "margin", but allowing for error. We put "line" in quotation marks because with higher-dimensional data, the "line" is actually a hyper-plane-thing. Similarly, the "margin" doesn't actually exist the in sense of a clean margin between two things, since we are allowing for some sort of error. The support vectors are the points closest to the middle that carry the most weight when classifying, since they are the most prone to error and are deciding factors of where the classification line could go. We used a radial kernel function to deal with the non-linearity and higher-dimensions.

The support vector machine is pretty effective in higher-dimensional spaces, but doesn't perform very well with very large data sets with lots of noise. If we were to look at hotels with millions of records, a support vector machine may not be suitable. We fit our SVM using the `kernlab` and `caret` packages in R.

K-Nearest Neighbors (KNN)

K-NN is a "model free" approach, meaning that it doesn't assume any probability model on the data. Given an observation, $\boldsymbol{x}$, we want to find the k training observations that are "closest" to $\boldsymbol{x}$, and then either classify the new observation using majority vote among these k neighbors or predict the new observation by averaging the closest k neighbors. We define "closeness" based on Euclidean distance in our model. As this is the case, we need to center and scale the data, so that different scales of measurement are kept constant.

The K-nearest neighbors model is relatively intuitive and simple and has no underlying assumptions, making it a good model to consider. In our classification, we are only using it for a binary classification, but it is also very easy to extend the K-NN model to multiple classes. On the other hand, the K-NN algorithm is very sensitive to outliers, high dimensional data, and imbalanced data.

The number (k) of closest neighbors is a parameter that needs tuning. We used the `class` and `caret` packages in R to fit this model. Based on our cross-validation, we used a k value of 5.

Logistic Regression

The logistic regression model uses the "logit" function, $log(\frac{p}{1-p})$, which links the mean to the linear form of the regression model, $\boldsymbol{X\beta}$. Using it for binary classification, we round the fitted values either up or down to 1 or 0. The logistic regression function is defined as

$$\ln\left(\frac{p(x)}{1-p(x)}\right) = x'\beta, \qquad \text{where } p(x) = (1 + e^{-x'\beta})$$

The logistic regression model assumes that each observation is independent, that there is little or no multicollinearity among the predictors, and that there is a linear relationship between the predictor variables and log odds. This differs from the typical regression model, where the residuals must be normally distributed and have constant variance.

To fit the logistic regression model, we used the `glm()` function in base R. The advantages to using a logistic regression model are similar to those for the basic classification tree: it is not computationally expensive and easy to interpret the results. On the other hand, one of the major drawbacks to using a logistic regression model is that it is subject to the distributional assumptions on the data and errors mentioned above. If our assumptions are broken, our predictions may not be very reliable.

LASSO Regression

LASSO, which stands for Least Absolute Shrinkage and Selection Operator works to shrink beta coefficient estimators towards zero using a penalty term. This is different from traditional shrinkage methods since the form of the LASSO penalty actually forces some of the coefficients to exactly zero. This helps achieve variable selection and yields sparser models, similar to subset selection. In particular, the LASSO estimate minimizes:

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| = RSS + \lambda\sum_{j=1}^{p}|\beta_j|$$

where λ is the penalty parameter, with p different predictors.

The main benefits of using LASSO are the variable selection property, the computational efficiency, and the interpretability. Plus, as with any regularization method, it can help to avoid overfitting with a very flexible model. On the other hand, some of the pitfalls of LASSO are that the selected variables could be highly biased, correlation among predictors is not very well dealt with, and the feature selection may not be stable over different bootstrapped samples.

To fit the LASSO regression model, we used the `glmnet` package in R.

Boosted Tree

Boosting is the process of slowly training trees to avoid overfitting. The general approach is as follows: the trees are grown sequentially, where each subsequent tree is grown on a modified version of the original data, while the predictions are continuously updated as the trees are grown. The model has three parameters that need tuning:

- B - the number of trees
- λ - the shrinkage parameter, which is a small positive number that controls the rate at which the boosting learns
- d - the number of splits in each tree, which controls the complexity of the boosted ensemble

The main benefit of using boosting is to reduce overfitting. It often yields the best results in terms of the ensemble learning methods. Like the random forest model, since boosting is an ensemble learning technique, it will have a lower variance than other methods. Also, there is no need to scale down variables or create any dummy variables for categorical variables. All that said, boosting is not without its shortcomings; it does not do well with high dimensionality, it's not very interpretable (since it is a sort of blackbox algorithm), and it is relatively expensive computationally.

Multiple Linear Regression

Multiple Linear Regression is the extension of simple linear regression to a set of multiple predictor variables. The goal of multiple linear regression is to model the linear relationship between a continuous response and two or more predictor variables. Like simple linear regression, coefficient estimates are found by minimizing the sum of the squared errors. The formula for a multiple linear regression model with p predictor variables is as follows:

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + ... + \beta_p X_{ip} + \epsilon_i,$$

$$\begin{aligned}
&\text{where for } i = n \text{ observations,}\\
&y_i = \text{dependent variable (response)}\\
&x_{ji} = \text{indpendent variables } (j = 1, ..., p)\\
&\beta_0 = \text{y-intercept}\\
&\beta_j = \text{slope coefficients for each variable}\\
&\epsilon_i = \text{residuals (error term of model)}
\end{aligned}$$

The multiple linear regression model is subject to the following assumptions about the data: * Linear relationship between response and predictors * No collinearity between predictor variables * y_i's are iid * $\epsilon_i \sim N(0, \sigma^2)$

The biggest benefit to using a multiple linear regression model is its interpretability and ease of use. The coefficients can be computed efficiently and they are have clear and easy interpretations. It is also possible to make inference on any of the variables or linear combination of variables. The most signficiant drawback of using the multiple linear regression model is that, like the logistic regression model, it is subject to distributional assumptions, meaning that if the assumptions are violated, our estimates and inference will be unstable at best and invalid at worst.

Results

Table 2: Regression Model Results

Regression Method	Root MSE
Boosted Tree	0.6268
Random Forest	0.5830
Multiple Linear Regression (MLR)	0.6433
KNN Regression	0.6403
LASSO Regression	0.6434

Table 3: Classification Model Results

Classification Method	Accuracy	Sensitivity	Specificity
Classification Tree	0.7068	0.8263	0.5699
Random Forest	0.8195	0.8216	0.8172
Support Vector Machine (SVM)	0.7469	0.7324	0.7634
KNN Classification	0.7143	0.7418	0.6828
Logistic Regression	0.7343	0.7136	0.7581

Based on the output, the best model for both classification and regression was the random forest. This was not a surprise, since the ensemble methods tend to outperform regular methods. This is further exemplified by the lower-than-average root MSE of the boosted tree model. The other models all did about the same in terms of prediction accuracy. For classification, all of the non-aforementioned methods were in the low 70s in terms of prediction accuracy, while the non-ensemble regression models had root MSEs right around 0.65.

Conclusion

Appendix

```r
#### SETUP STUFF ####
# Set default chunk options
knitr::opts_chunk$set(echo = FALSE, message = FALSE, warning = FALSE)

# Load necessary libraries
library(tidyverse) # so many things
library(knitr) # making nice tables
library(caret) # training models & CV
library(leaps) # subset selection
library(corrplot) # visualize correlation
library(rmarkdown) # for output documents
library(rpart) # fitting classification tree
library(rpart.plot) # visualization of classification tree
library(randomForest) # fitting random forest model
library(kernlab) # fitting SVM
library(class) # fitting kNN model
library(glmnet) # fitting LASSO
library(gbm) # fitting the boosted tree model
library(kableExtra) # making bolded headers for tables

# Read in data set
wine <- read.csv2(file = "winequality-red.csv", header = TRUE, dec = ".",
                  colClasses = c(rep("double", 7), rep("double", 4), "integer"))

# Set seed for reproducibility
set.seed(702)

#### METHODS TABLE ####
# Make table of methods
cl <- c("Classification Tree", "Random Forest", "Support Vector Machine (SVM)",
        "KNN Classification", "Logistic Regression")
reg <- c("Boosted Tree", "Random Forest", "Multiple Linear Regression (MLR)",
          "KNN Regression", "LASSO Regression")
kable(data.frame("Classification" = cl, "Regression" = reg), caption = "ML Methods") %>%
  row_spec(0, bold=TRUE) %>% collapse_rows(columns = 1, latex_hline = "none") %>%
  kable_styling(latex_options = "hold_position")

#### DATA PARTITIONING ####
# Prepare data for classification
wine$class <- ifelse(wine$quality <= 5, "low", "high") %>% as.factor()

# Create Data Partition
ind <- createDataPartition(wine$quality, p = .75, list = FALSE)

# Create train/test data for H1
trn <- wine[ind,]
tst <- wine[-ind,]

#### CROSS VALIDATIONN ####
# Specify the cross-validation method (5-fold CV)
trctrl <- trainControl(method = "cv", number = 5)

#### VARIABLE SELECTION ####
# Write preliminary model
model.class.pre <- class ~ .

# Fit preliminary random forest on entire data set
rf.class.pre <- randomForest(model.class.pre, data = wine[,-12],
                             ntree = 500, mtry = 3, importance = TRUE)

# Create data frame of variable importance
df.class.pre <- rf.class.pre$importance %>% as.data.frame()
X <- rownames(df.class.pre)

# Sort the data frame and prep for plot
VIrf.pre.class.sort <- df.class.pre[order(df.class.pre[,"MeanDecreaseAccuracy"]), , drop = FALSE]
VIrf.pre.class.sort$X <- rownames(VIrf.pre.class.sort)
VIrf.pre.class.sort$X <- factor(VIrf.pre.class.sort$X, levels = VIrf.pre.class.sort$X)

# Plot variable importance
```

```r
vimp.class <- ggplot(VIrf.pre.class.sort, aes(x = X, y = MeanDecreaseAccuracy)) +
                geom_bar(stat = 'identity', position = 'dodge', fill="lightblue") +
                theme(axis.text.x = element_text(angle = 30, hjust = 1)) +
                ylab("Mean Decrease in Accuracy") + xlab("Predictor") +
                ggtitle("Variable Importance") +
                geom_hline(yintercept = .005, lty = 2, colour = "darkblue")
vimp.class

# Create model and get summary output
lm.sub <- regsubsets(quality ~ ., data = wine[,-13], nvmax = 11)
smre <- lm.sub %>% summary()

# Plot the Cp, BIC, R^2, RSS results in one plot
par(mfrow = c(2,2))
plot(smre$cp, xlab = "# of Preds", ylab = "Cp")
points(which.min(smre$cp), smre$cp[which.min(smre$cp)], pch = 8, col = "#CC0000")
plot(smre$bic, xlab = "# of Preds", ylab = "BIC")
points(which.min(smre$bic), smre$bic[which.min(smre$bic)], pch = 8, col = "#CC0000")
plot(smre$adjr2, xlab = "# of Preds", ylab = "Adjusted R^2")
points(which.max(smre$adjr2), smre$adjr2[which.max(smre$adjr2)], pch = 8, col = "#CC0000")
plot(smre$rss, xlab = "# of Preds", ylab = "Training RSS")
points(which.min(smre$rss), smre$rss[which.min(smre$rss)], pch = 8, col = "#CC0000")

# Best subset selection
best.betas <- coef(lm.sub, id = 6) %>% round(2)

#### WRITE FORMULAS ####
# Write formula for regression models
form.reg <- quality ~ volatile.acidity + chlorides + pH +
            total.sulfur.dioxide + sulphates + alcohol

# Write formula for classification models
form.class <- class ~ alcohol + sulphates + density +
              volatile.acidity + total.sulfur.dioxide

#### BASIC CLASSIFICATION TREE ####
# Fit classification tree model
tree.class <- train(form.class, data = trn, method = "rpart",
                    trControl = trctrl)

# Obtain confusion matrix and results for H2
tree.class.pred <- predict(tree.class, newdata = tst)
tree.class.CM <- confusionMatrix(tree.class.pred, tst$class)
tree.class.res <- c(tree.class.CM$overall[1], tree.class.CM$byClass[1:2])

# Print out classification tree visualization
tree.class.out <- rpart(form.class, data = trn, cp = 0.01971326)
rpart.plot(tree.class.out, main="Classification Tree")

#### RANDOM FOREST ####
# Fit the regression random forest model
rf.reg <- train(form.reg, data = trn, method = "rf",
                trControl = trctrl, importance = TRUE)
```

```r
# Obtain RMSE and results
rf.reg.pred <- predict(rf.reg, newdata = tst)
rf.reg.rmse <- sqrt(mean((rf.reg.pred - tst$quality)^2))

# Fit the classification random forest model
rf.class <- train(form.class, data = trn, method = "rf",
                trControl = trctrl, importance = TRUE)

# Obtain confusion matrix and results
rf.class.pred <- predict(rf.class, newdata = tst)
rf.class.CM <- confusionMatrix(rf.class.pred, tst$class)
rf.class.res <- c(rf.class.CM$overall[1], rf.class.CM$byClass[1:2])

#### SUPPORT VECTOR MACHINE ####
# Fit support vector machine for H1
svm.class <- train(form.class, data = trn, method = "svmRadial",
                   preProcess = c("center", "scale"),
                   trControl = trctrl, tuneLength = 10)

# Obtain confusion matrix and results for H1
svm.class.pred <- predict(svm.class, newdata = tst)
svm.class.CM <- confusionMatrix(svm.class.pred, tst$class)
svm.class.res <- c(svm.class.CM$overall[1], svm.class.CM$byClass[1:2])

#### KNN ####
# Fit knn regression model
knn.reg <- train(form.reg, data = trn, method = "knn",
                 trControl = trctrl, preProcess = c("center", "scale"))

# Obtain RMSE and results
knn.reg.pred <- predict(knn.reg, newdata = tst)
knn.reg.rmse <- sqrt(mean((knn.reg.pred - tst$quality)^2))

# Fit knn classification model
knn.class <- train(form.class, data = trn, method = "knn",
                   trControl=trctrl, preProcess = c("center", "scale"))

# Obtain confusion matrix and results
knn.class.pred <- predict(knn.class, newdata = tst)
knn.class.CM <- confusionMatrix(knn.class.pred, tst$class)
knn.class.res <- c(knn.class.CM$overall[1], knn.class.CM$byClass[1:2])

#### LOGISTIC REGRESSION ####
# Fit logistic regression model
glm.class <- glm(form.class, data = trn, family = binomial(link = "logit"))

# Obtain confusion matrix and results for H2
glm.class.pred <- predict(glm.class, newdata = tst, type = "response")
glm.class.pred <- glm.class.pred %>% round() %>% as.factor()
glm.class.CM <- confusionMatrix(glm.class.pred, ifelse(tst$class == "low", 1, 0) %>% as.factor())
glm.class.res <- c(glm.class.CM$overall[1], glm.class.CM$byClass[1:2])

#### LASSO ####
```

```r
# Fit LASSO to the data
lasso.reg <- train(form.reg, data = trn, method = "glmnet", trControl = trctrl,
                   tuneGrid = expand.grid(.alpha=1, .lambda=0.00175))

lasso.reg.pred <- predict(lasso.reg, newdata = tst)
lasso.reg.rmse <- sqrt(mean((lasso.reg.pred - tst$quality)^2))

#### BOOSTING ####
# Fit the regression boosted tree model
boost.reg <- train(form.reg, data = trn, method = "gbm",
                trControl = trctrl, verbose = FALSE)

# Obtain RMSE and results
boost.reg.pred <- predict(boost.reg, newdata = tst)
boost.reg.rmse <- sqrt(mean((boost.reg.pred - tst$quality)^2))

#### MLR ####
lm.reg <- train(form.reg, data = trn, method = "lm",
                trControl = trctrl)

# Obtain RMSE and results for linear model
lm.reg.pred <- predict(lm.reg, newdata = tst)
lm.reg.rmse <- sqrt(mean((lm.reg.pred - tst$quality)^2))

#### OUTPUT RESULTS ####
# Create data frame output for regression models
df.reg <- rbind.data.frame(boost.reg.rmse, rf.reg.rmse,
                           lm.reg.rmse, knn.reg.rmse,
                           lasso.reg.rmse) %>% round(4)
df.reg$method <- reg
colnames(df.reg) <- c(LETTERS[1:2])
df.reg <- df.reg %>% select("B", "A")
colnames(df.reg) <- c("Regression Method", "Root MSE")

# Create data frame output for classification models
df.class <- rbind.data.frame(tree.class.res, rf.class.res,
                             svm.class.res, knn.class.res,
                             glm.class.res) %>% round(4)
df.class$method <- cl
colnames(df.class) <- c("A", "B", "C", "D")
df.class <- df.class %>% select("D", "A", "B", "C")
colnames(df.class) <- c("Classification Method", "Accuracy", "Sensitivity", "Specificity")

# Smash tables next to each other using kable
kable(df.reg, caption = "Regression Model Results", format = "latex") %>%
  row_spec(0, bold=TRUE) %>% collapse_rows(columns = 1, latex_hline = "none") %>%
  kable_styling(latex_options = "HOLD_position")
kable(df.class, caption = "Classification Model Results", format = "latex") %>%
  row_spec(0, bold=TRUE) %>% collapse_rows(columns = 1, latex_hline = "none") %>%
  kable_styling(latex_options = "HOLD_position")
```