Machine learning - helping to not make "pour" decisions

John Ansell

15/10/2020

Wine classification and quality prediction

Introduction

In 2017 the global consumption of wine was estimated to be 246,000,000 hectolitres[1]. For reference a hectolitre is 100 litres. A conservative estimate therefore places global wine consumption per capita at approximately 3.2 litres/per person.

This report will make use of the wine quality data sets from the UCI machine learning repository. The data was initially gathered and utilised by Cortez P. and others in: *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. ISSN: 0167-9236.*

The data is comprised of two tables, one for red wines and one for white wines. The data sets are exclusively for the *vinho verde* cultivar from Northern Portugal. The tables consist of eleven different physiochemical components of each wine that was sampled. There is a twelfth variable which is the perceived quality of the wine.

It must be noted that *vinho verde* enjoys geographical protection and official certification of *vindo verde* is under the authority of the CRVV[2].

That the data is already divided between red and white wines provides an excellent opportunity to deal with two distinct machine learning problems:

- Can a machine learning algorithm classify wine as red or white based on the underlying physiochemical measures?
- How accurately can a machine learning algorithm predict the perceived quality of a wine based on the underlying physiochemical measures?

To address the first question raised we will start with the most simple of binary classifiers - a generalised linear model with a binomial predictor. A random forest will then be implemented. While the ranger and RBorist random forest implementations both offer performance gains over the older randomForest package, the randomForest offers slightly more transparent output in the form of the varImp() function. Overall accuracy will be used as the primary performance metric - the distinction between sensitivity and specificity does not have the significant consequences as one would have in certain other applications of binary classifiers.

The second problem outlined above, depending on the success of the binary classification between white and red, will necessarily be carried out as predication problem on white wine and red wine separately; if the physiochemical properties are insufficient to distinguish a red wine from a white wine there would be little point in modeling perceived quality for either type separately. Once again the modeling process will be started simply with linear models, and move onto to more sophisticated models. Support vector machines were used very successfully in the original work by Cortez *et al.*

[1] https://www.statista.com/statistics/232937/volume-of-global-wine-consumption/#
[2] https://www.vinhoverde.pt/en/homepage

The libraries

There are a number of libraries available in R to facilitate machine learning operations. The caret package, by Max Kuhn, package will be implemented in this report due to it's ease of use and the extensive documentation[3]. The tidyverse suite of packages will also used.

The machine learning libraries that will be used is the randomForest package for the implementation of random forests, kernlab for the implementation of non-linear support vector machines and e1071 for linear support vector machines.

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")

if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")

if(!require(ggthemes)) install.packages("ggthemes", repos = "http://cran.us.r-project.org")

if(!require(randomForest)) install.packages("randomForest", repos = "http://cran.us.r-project.org")

if(!require(MASS)) install.packages("MASS", repos = "http://cran.us.r-project.org")

if(!require(e1071)) install.packages("e1071", repos = "http://cran.us.r-project.org")

if(!require(kableExtra)) install.packages("kableExtra", repos = "http://cran.us.r-project.org")
```

After checking for the existence of the libraries, and installing where necessary, on the user's local system, it would now be appropriate to load the required libraries and proceed with retrieving the data for the project.

```
library(tidyverse)
library(caret)
library(ggthemes)
library(randomForest)
library(MASS)
library(e1071)
library(kableExtra)
```

The data

The raw data files are downloaded from the UCI Machine learning repository, they are saved in a folder within the current working directory "datasets":

```
url_red_wine <- "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red

url_white_wine <- "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-wh

download.file(url_red_wine, "datasets/red_wine.csv")
download.file(url_white_wine, "datasets/white_wine.csv")

rm(url_red_wine)
rm(url_white_wine)
```

Reading in the data into the R environment is fairly straight forward:

```
red_wine <- read_delim("datasets/red_wine.csv", ";")
white_wine <- read_delim("datasets/white_wine.csv", ";")
```

The red wine data set consists of 1599 wines and the white wine data set consists of 4898 different wines.

[3] https://topepo.github.io/caret/

When we attempt the binary classification into red or white wine, this is something that must be considered as tree based models will tend to "favour" the dominant classes in unbalanced data sets.

Before embarking on a data science misadventure it is important to verify that the physiochemical properties that are recorded are the same for both red and white wine.

```
identical(colnames(red_wine), colnames(white_wine))
```

```
## [1] TRUE
```

The variables

The eleven physiochemical properties that are measured are:

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

1. "fixed acidity" - acidity in wine is typically divided into fixed acids and volatile acids[4]. Fixed acids are non-volatile (do not evaporate easily and do not have much impact on smell.) These acids include tartaric, malic and succinic. They are measured in grams per litre.
2. "volatile acidity" - these are acids which evaporate easily and can have an impact on the smell, or "nose" of a wine. The acid most prevalent in this group is acetic acid. Measured in grams per litre.
3. "citric acid" - is a measure of the amount of citric acid present in grams per litre.
4. "residual sugar" - sugar which is abundant in grapes is metabolised by the yeast during during the fermentation process. Residual suagr refers to the sugar remaining after the process is finished. Unit of measure is grams per litre.
5. "chlorides" - this is a measure of certain mineral salts which are present in all wines. The unit of measure is grams per per litre.
6. "free sulfur dioxide" - sulphur dioxide is used as a preservative and anti-microbial agent in the fruit industry. Free sulphur dioxide refers to the sulfur dioxide that has not binded with other chemicals in the wine. This is measured in milligrams per litre.
7. "total sulfur dioxide" - this refers to the total sulfur dioxide added during the wine making process. The required amount depends largely on the pH of the wine[5]. Measured again in milligrams per litre.
8. "density" - this refers to the density of the wine. Typically the density of wine is very similar to water. This is measured in grams per cubic centimeter.
9. "pH" - is a measure of acidity of a wine. The pH scale is generally measured from 0-14 with 0 being very acidic and 14 very basic. While the scale is not bound on either end, this discussion is beyond the scope of this project.
10. "sulphates" - please see discussion above on six and seven.
11. "alcohol" - the alcohol content of the wine, expressed as a percentage

The final variable in the data set is the quality. This is measured on a scale of 0-10. The rating is the median score from three assessors who blind taste-test each wine[6].

A brief glimpse at the structure of the numeric data that comprises the eleven predictor variables described above:

[4] https://waterhouse.ucdavis.edu/whats-in-wine/fixed-acidity

[5] https://morewinemaking.com/articles/SO2_management

[6] 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. ISSN: 0167-9236.

Table 1: Red wine summary data

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
	Min. : 4.60	Min. :0.1200	Min. :0.000	Min. : 0.900	Min. :0.01200	Min. : 1.00	Min. : 6.00	Min. :0.9901	Min. :2.740	Min. :0.3300	Min. : 8.40
	1st Qu.: 7.10	1st Qu.:0.3900	1st Qu.:0.090	1st Qu.: 1.900	1st Qu.:0.07000	1st Qu.: 7.00	1st Qu.: 22.00	1st Qu.:0.9956	1st Qu.:3.210	1st Qu.:0.5500	1st Qu.: 9.50
	Median : 7.90	Median :0.5200	Median :0.260	Median : 2.200	Median :0.07900	Median :14.00	Median : 38.00	Median :0.9968	Median :3.310	Median :0.6200	Median :10.20
	Mean : 8.32	Mean :0.5278	Mean :0.271	Mean : 2.539	Mean :0.08747	Mean :15.87	Mean : 46.47	Mean :0.9967	Mean :3.311	Mean :0.6581	Mean :10.42
	3rd Qu.: 9.20	3rd Qu.:0.6400	3rd Qu.:0.420	3rd Qu.: 2.600	3rd Qu.:0.09000	3rd Qu.:21.00	3rd Qu.: 62.00	3rd Qu.:0.9978	3rd Qu.:3.400	3rd Qu.:0.7300	3rd Qu.:11.10
	Max. :15.90	Max. :1.5800	Max. :1.000	Max. :15.500	Max. :0.61100	Max. :72.00	Max. :289.00	Max. :1.0037	Max. :4.010	Max. :2.0000	Max. :14.90

Table 2: White wine summary data

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
	Min. : 3.800	Min. :0.0800	Min. :0.0000	Min. : 0.600	Min. :0.00900	Min. : 2.00	Min. : 9.0	Min. :0.9871	Min. :2.720	Min. :0.2200	Min. : 8.00
	1st Qu.: 6.300	1st Qu.:0.2100	1st Qu.:0.2700	1st Qu.: 1.700	1st Qu.:0.03600	1st Qu.: 23.00	1st Qu.:108.0	1st Qu.:0.9917	1st Qu.:3.090	1st Qu.:0.4100	1st Qu.: 9.50
	Median : 6.800	Median :0.2600	Median :0.3200	Median : 5.200	Median :0.04300	Median : 34.00	Median :134.0	Median :0.9937	Median :3.180	Median :0.4700	Median :10.40
	Mean : 6.855	Mean :0.2782	Mean :0.3342	Mean : 6.391	Mean :0.04577	Mean : 35.31	Mean :138.4	Mean :0.9940	Mean :3.188	Mean :0.4898	Mean :10.51
	3rd Qu.: 7.300	3rd Qu.:0.3200	3rd Qu.:0.3900	3rd Qu.: 9.900	3rd Qu.:0.05000	3rd Qu.: 46.00	3rd Qu.:167.0	3rd Qu.:0.9961	3rd Qu.:3.280	3rd Qu.:0.5500	3rd Qu.:11.40
	Max. :14.200	Max. :1.1000	Max. :1.6600	Max. :65.800	Max. :0.34600	Max. :289.00	Max. :440.0	Max. :1.0390	Max. :3.820	Max. :1.0800	Max. :14.20

```
red_wine %>%
  dplyr::select(-quality) %>%
  summary() %>%
  knitr::kable(format = "latex", caption = "Red wine summary data") %>%
  kableExtra::kable_styling(latex_options = "scale_down")
```

```
white_wine %>%
  dplyr::select(-quality) %>%
  summary() %>%
  knitr::kable(format = "latex", caption = "White wine summary data") %>%
  kableExtra::kable_styling(latex_options = "scale_down")
```

Red vs white

As this project compromises two sections, it would be appropriate to add a column to each wine data set for the colour - red or white and merge the tables into a single, tidy data frame. This will also allow us to exploit the full potential of the ggplot2 package.

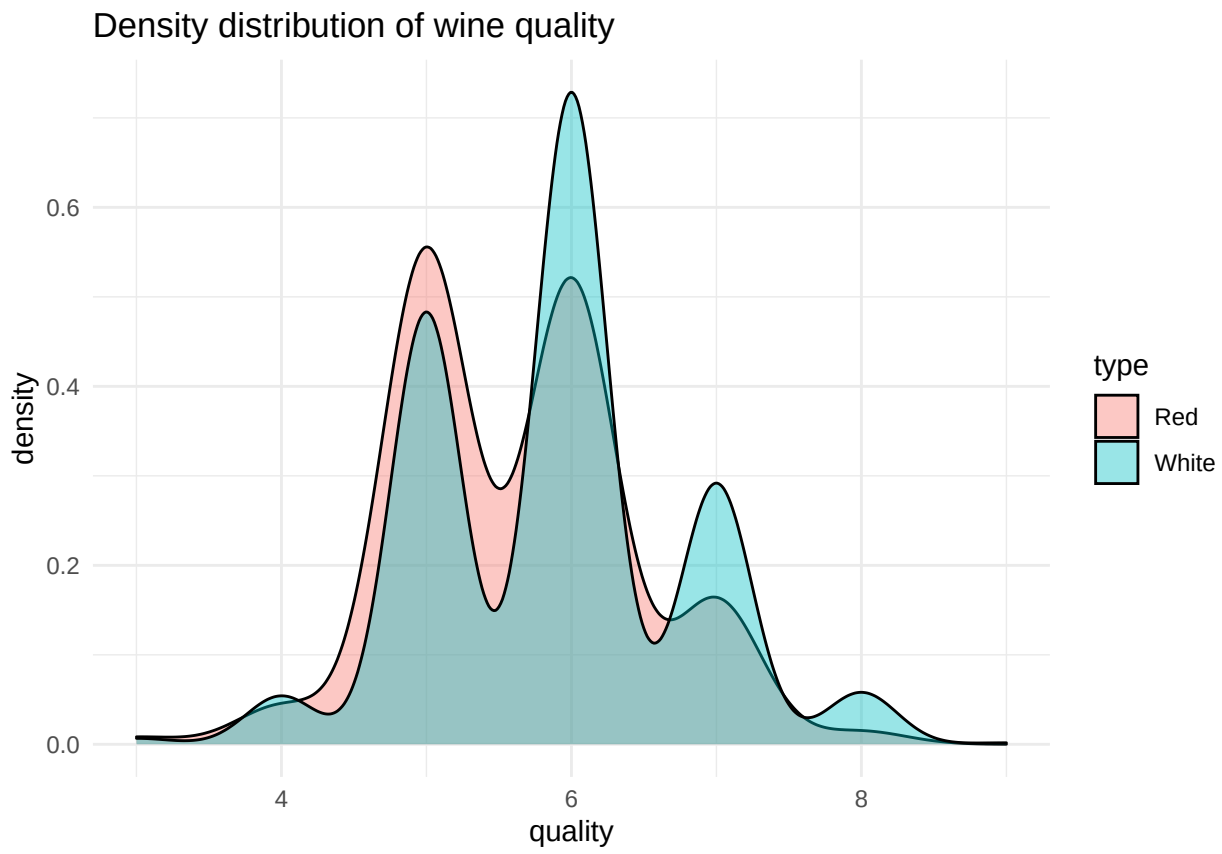
```
red_temp <- red_wine %>%
  mutate(type = "Red")

white_temp <- white_wine %>%
  mutate(type = "White")

wine_combined <- bind_rows(white_temp, red_temp)

rm(white_temp, red_temp)
```

There is now a single combined data set for both red and white wine of size 6497 * 13.

Some brief tidying up - converting the wine type to a factor for data visualisation.

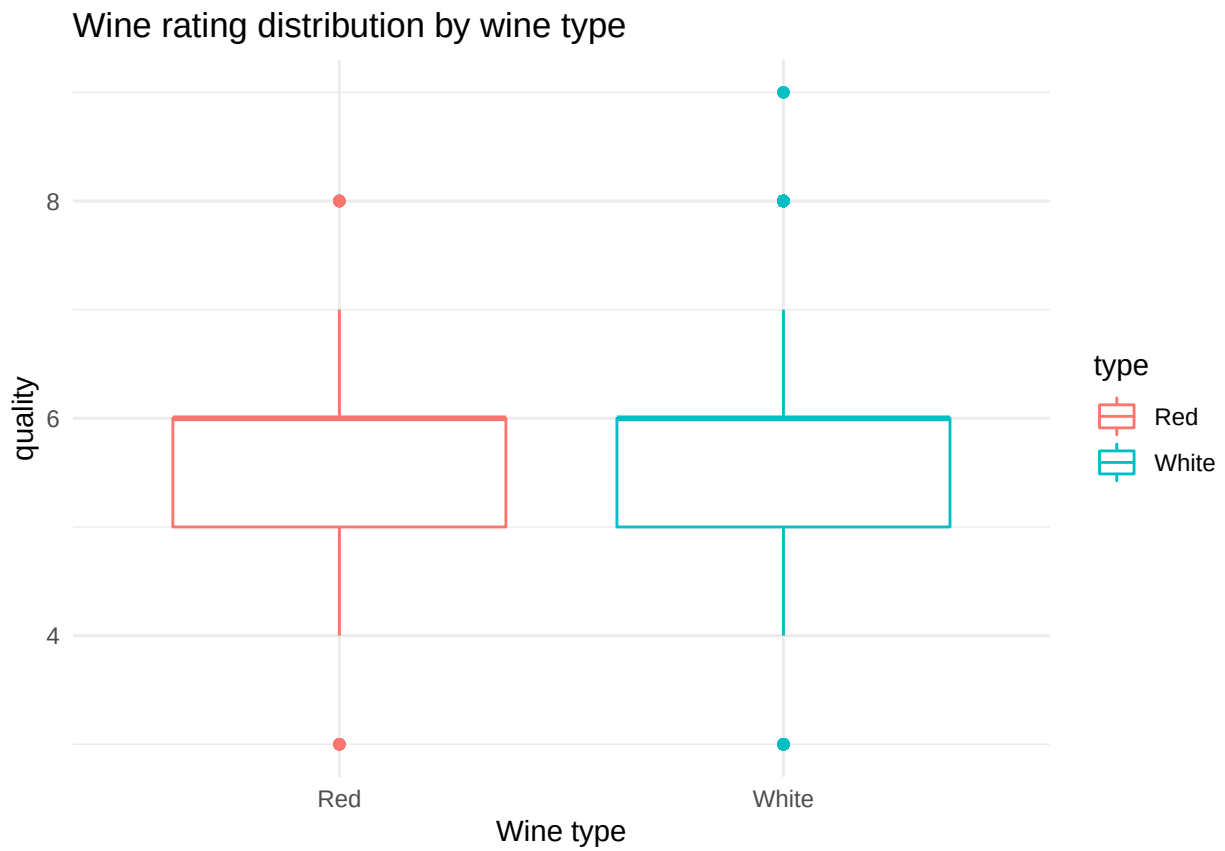
```
wine_combined$type <- as.factor(wine_combined$type)
```

Now we have done some data adjusting it is time to address an age old question - is white or red better?



The quality variable is ordinal and not continuous - thus the density distribution is multi-modal. Though the density function suggest that white wines fare a little better than the red wines.

A better visualisation of the distribution of the "quality" variable would probably be a box plot. This allows for a visual, side-by-side comparison of the variability in the data.



Visually the distribution of the quality ratings are almost identical, with the white wine having one further outlier. A brief numeric summary of the salient data for each wine type should serve to confirm the visual findings:

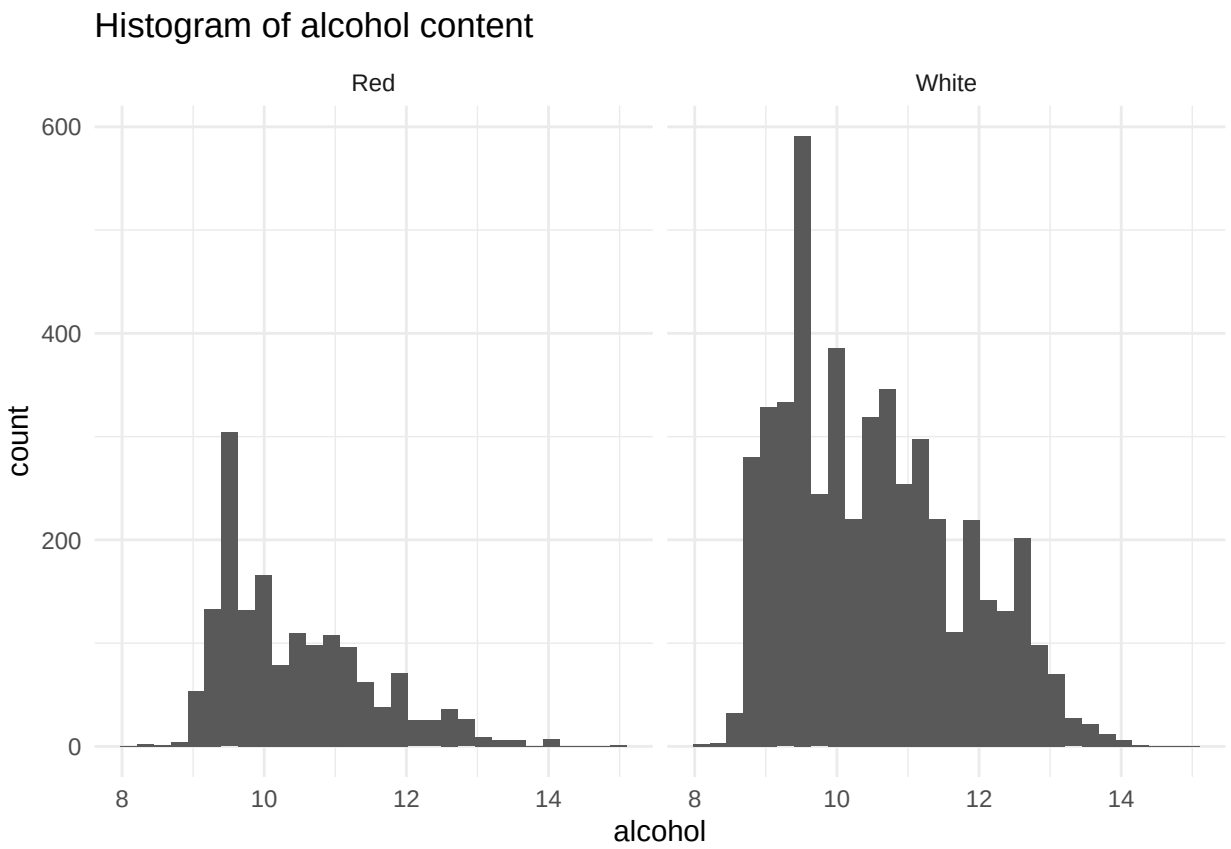
```
wine_combined %>%
  group_by(type) %>%
  summarise(mean = mean(quality), std_dev = sd(quality), median = median(quality)) %>%
  knitr::kable() %>%
  kable_styling(latex_options = "HOLD_position")
```

type	mean	std_dev	median
Red	5.636023	0.8075694	6
White	5.877909	0.8856386	6

In terms of perceived quality there is very little to choose between red or white wine. This is a little surprising as the *vinho verde* cultivar is better known for its white wines.

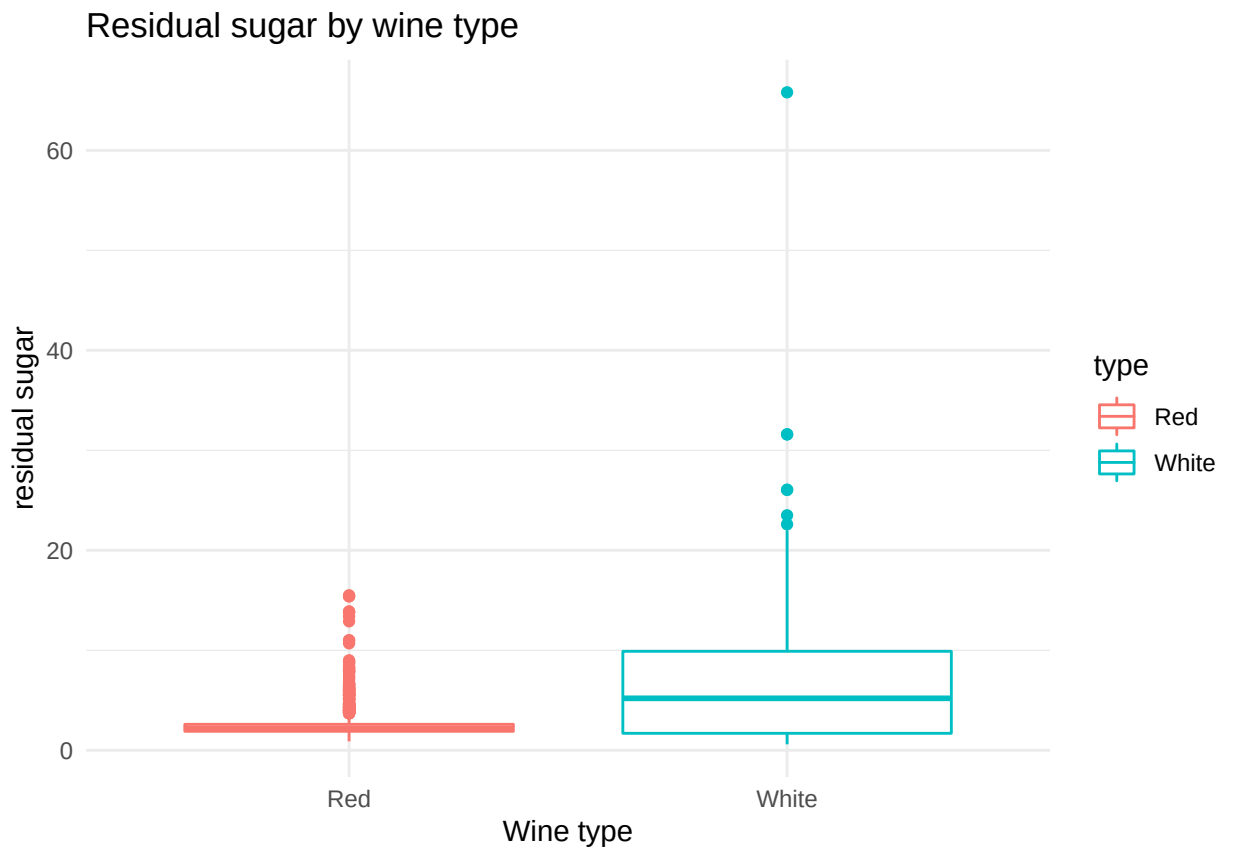
Sugar and alcohol

Alcohol (or more specifically ethanol - alcohol refers to a group of organic compounds) is the second most abundant chemical in wine after water. From the earlier tables we see that in general, for both red and white wine, alcohol content spans from about 8%-14%. The medians and the means for both wine types are close and we therefore assume the alcohol content is normally distributed.



The distributions for both red and white wine alcohol content follow a similar shape. Both are right skewed with median values and mean values in the 10.2%-10.5%. There is little to distinguish white wine from red wine in terms of alcohol content. The shape of the histograms do suggest that the assumption of normality is perhaps not entirely accurate, and re-enforces that humans are more visually intuitive than numerically so.

Red wine is generally considered to be a "drier" or less sweet drink than white wine. A brief look at the tables above would suggest that this is down in large part due to the difference in the residual sugar levels.



Herein lies the first significant difference between the two wine types. The total inter-quartile range for red wine lies within the second quartile of the amount of residual sugar. The outliers in white wine have some relatively high values - between 20-70 grams per litre.

Data conclusions

To document a full exploratory data analysis on the wines data sets in itself could complete a lengthy tome. While there are only eleven features in each data set, the data sets were selected so that more advanced machine learning algorithms would be able to run on a reasonable PC. In the first project the size of the Movielens dataset precluded this approach.

There is no real concern about the "curse of dimensionality"[7] - eleven feature variables are handled with relative ease by a modern PC. That said a variable which is almost constant with very little deviation will not contribute significantly to any classical regression or machine learning model.

The "density" variable is one such variable which shows very little variation or range.

```
wine_combined %>%
  summarise(min_density = min(density), mean_density = mean(density), sd_density = sd(density), max_dens
  knitr::kable()
```

min_density	mean_density	sd_density	max_density
0.98711	0.9946966	0.0029987	1.03898

It is evident that the "density" variable is so closely clustered around 1 $g.cm^{-3}$ that the variable can be safely discarded from the data sets before any models are built without fear of this having a detrimental impact on the models.

[7] A phrase coined by Richard Bellman in: Bellman, Richard Ernest; Rand Corporation (1957). Dynamic programming. Princeton University Press. p. ix. ISBN 978-0-691-07951-6.

The variable is easily removed using the following snippet of code.

```
red_wine <- red_wine %>%
  dplyr::select(-density)

white_wine <- white_wine %>%
  dplyr::select(-density)

wine_combined <- wine_combined %>%
  dplyr::select(-density)
```

The data that we will need to tackle the two machine learning problems is now ready, there are three data frames in the global environment:

- wine_combined - this will be used for the binary classification between red and white wine
- white_wine - this will be used for quality prediction
- red_wine - this will be used for quality prediction

Techniques - Binary Classification - Red wine or white wine?

In this section of the report an attempt will be made to classify wines as either white or red based on the ten physiochemical measurements outlined earlier (the predictor "density" was removed from the data sets). For this section the only dataframe that will be used is wine_combined which is a joined table of the red and white wines with the wine type added as a factor.

At this stage of the project the outcome "quality" is not of interest and will be removed to not interfere with the modeling process.

```
wine_combined <- wine_combined %>%
  dplyr::select(-quality)
```

The first step would be to split the data into a training set and a test set. The model will only be trained on the training data. Cross validation, a technique of re-sampling the training data to create validation data sets for tuning the model's parameters.

A brief discussion of the choice how to split the data would be appropriate at this stage. By including too much of the data set in the training data set gives rise to the following complications:

- There is a risk that the model is over-trained on the data fed into it - by passing most of the variability of the predictor variables into a model may lead to the model selecting parameters optimised to fit the training data. But this model could then perform quite poorly on new data.
- A large training set can be quite computationally troublesome as there is more data to which a model must be fitted. Even sum of least squares regression requires a number of arithmetic steps to prepare. Ensemble machine learning models such as random forests are particularly computationally burdensome during the training process.

There is also a counter argument to all such debates. Having too little data in the training set can have the following drawbacks:

- The outcomes of the models will exhibit much more variability.
- Predictive performance of any such models will be less reliable and confidence intervals will be accordingly larger.

The overall size of the data sets will also impact the proportion of the data withheld for testing. The Movielens data set had 10,000,000 so the train:test split of 90%:10% still gave a large sample for testing.

While a much smaller data set at 6,497 observations a 90:10 split would in most likelihood lead to over training. Whilst there has been some published research into determining the optimal split[8], this is not

[8]I Guyon; AT & T Bell Laboratories. A Scaling Law for the validation-set training set ratio. http://citeseerx.ist.psu.edu/

exhaustive. The *Pareto principle*, loosely states that 80% of outcomes can be explained by a mere 20% of the determining factors.

In conclusion the training:test split will be made at 80%:20%, as this seems appropriate given the trade-offs set out above.

Splitting the data

The first step in any machine learning model would be to partition the data the into training data, on which to "teach" the model and test data, on which to assess the performance of the data.

The necessarily requires the process be done randomly. In order to ensure reproducibility of the results, the random number generator in R must be given a fixed starting point. In this project the seed used throughout will be 1236[9].

As discussed earlier the data is unbalanced - there are 4,898 samples of white wine compared to only 1,599 samples of red wine. Merely predicting all wine as red would lead to an overall accuracy of 75.3%. Certain machine learning models will bias inherently for unbiased data.

The function createDataPartition() from the caret package requires the outcome variable as an argument as it automatically performs the balancing in the sampling process. The output of this function is a list of indices that should be used for the training data. The data can be split into training and test data by sub-setting the the output of the function.

```
set.seed(1236, sample.kind = "Rounding")


index <- as.vector(createDataPartition(y = wine_combined$type, times = 1, p = 0.2, list = FALSE))

wine_test <- wine_combined[index,]
wine_train <- wine_combined[-index,]

rm(index)
```

First model - generalised linear model

As stated in the introduction to this project; each part will be attempted using a basic regression model and then further, more sophisticated models will be fitted.

The first attempt to predict whether a wine is a red wine or a white wine will be dome using a generalised linear model. There is a base function in R glm() which could be used for this, but for consistency we will used the caret interface. Caret also provides for cross validation within the training data to help. This is part of the trainControl argument which is passed to the model training function.

```
tr_Ctrl <- trainControl(method = "cv",
                        number = 5)

glm_model <- train(type ~ .,
                   data = wine_train,
                   method = "glm",
                   trControl = tr_Ctrl)
```

Our first model has been trained on the 5,197 randomly selected wines. It would be appropriate to test the model for it's accuracy.

viewdoc/download?doi=10.1.1.33.1337&rep=rep1&type=pdf

[9] 6 is the smallest perfect number. A perfect number is defined as a positive integer whose factors also sum to give the same result ie. 1 x 2 x 3 = 6 = 1 + 2 + 3

```
preds <- predict(glm_model, newdata = wine_test)

(results_glm <- confusionMatrix(wine_test$type, preds))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Red White
##      Red   311     9
##      White   4   976
##
##                Accuracy : 0.99
##                  95% CI : (0.983, 0.9947)
##     No Information Rate : 0.7577
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9729
##
##  Mcnemar's Test P-Value : 0.2673
##
##             Sensitivity : 0.9873
##             Specificity : 0.9909
##          Pos Pred Value : 0.9719
##          Neg Pred Value : 0.9959
##              Prevalence : 0.2423
##          Detection Rate : 0.2392
##    Detection Prevalence : 0.2462
##       Balanced Accuracy : 0.9891
##
##        'Positive' Class : Red
##
```

Surprisingly the most basic model we could have brought to bear on the problem showed remarkable performance. Only 13 wines out of 1300 were misclassified.

The following code will start to collect the results of the different models for final comparison at a later stage.

```
red_white_results <- tibble(model = "GLM", Accuracy = results_glm$overall[1])
```

Linear Support Vector Machine

The initial approach when planning this section of the project was to run a basic GLM, laugh at it's poor performance, and introduce some more sophisticated models. Herein does lie a lesson though - complexity is not always the best approach.

The, unexpected, success of the GLM indicates that a more complex model is perhaps not necessary. A linear SVM will be tuned through the caret interface. The implementation that will be used is the SVM through the e1071 library.

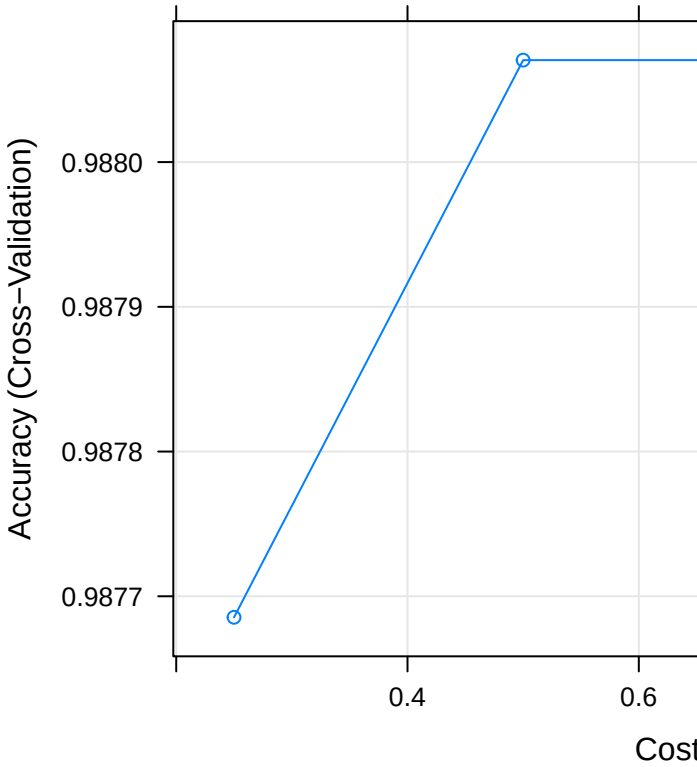
A brief discussion on parameters and hyperparameters would be in order before proceeding. The parameters of a machine learning model are the final settings based on the predictor variables within the data set. Hyperparameters refer to the external settings of a machine learning algorithm. These can be adjusted to assist the algorithm find an optimal solution for the particular problem that is trying to be solved. This can be achieved through the tuneGrid argument in caret's train() function.

The linear SVM model in the e1071 package only has one hyperparameter to adjust - "cost".

At first we will use the defaults that caret uses in tuning the model and then explore the impact that different values of cost has on the model.

```
set.seed(1236, sample.kind = "Rounding")
linear_SVM_model <- train(type ~ .,
                          data = wine_train,
                          method = "svmLinear2",
                          type = "C-classification",
                          trControl = tr_Ctrl)
```

The default options that caret uses in selecting the best cost parameter are: 0.25, 0.5 and 1. Depending on the accuracy each returns from the cross validation, the best performing option will be selected for the final model.



The plot of the cost parameter vs model accuracy is shown below.
The above plot hints that there is perhaps better accuracy to be gained out of the linear SVM if we were to allow the cost parameter to vary beyond the three default values that caret uses to train the model. For those who prefer a numeric summary of the accuracy for each cost value, the table is below.

```
tibble(Cost = linear_SVM_model$results$cost, Accuracy = linear_SVM_model$results$Accuracy) %>%
  knitr::kable(caption = "Caret default cost hyperparameters") %>%
  kable_styling(latex_options = "HOLD_position")
```

Table 3: Caret default cost hyperparameters

Cost	Accuracy
0.25	0.9876855
0.50	0.9880704
1.00	0.9880704

Before exploring alternative cost options, it would be worth exploring the accuracy of the default model on the test data.

```
preds <- predict(linear_SVM_model, newdata = wine_test)
(results_SVM <- confusionMatrix(wine_test$type, preds))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Red White
##      Red   311     9
##      White   4   976
##
##                Accuracy : 0.99
##                  95% CI : (0.983, 0.9947)
##     No Information Rate : 0.7577
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9729
##
##  Mcnemar's Test P-Value : 0.2673
##
##             Sensitivity : 0.9873
##             Specificity : 0.9909
##          Pos Pred Value : 0.9719
##          Neg Pred Value : 0.9959
##              Prevalence : 0.2423
##          Detection Rate : 0.2392
##    Detection Prevalence : 0.2462
##       Balanced Accuracy : 0.9891
##
##        'Positive' Class : Red
##
```

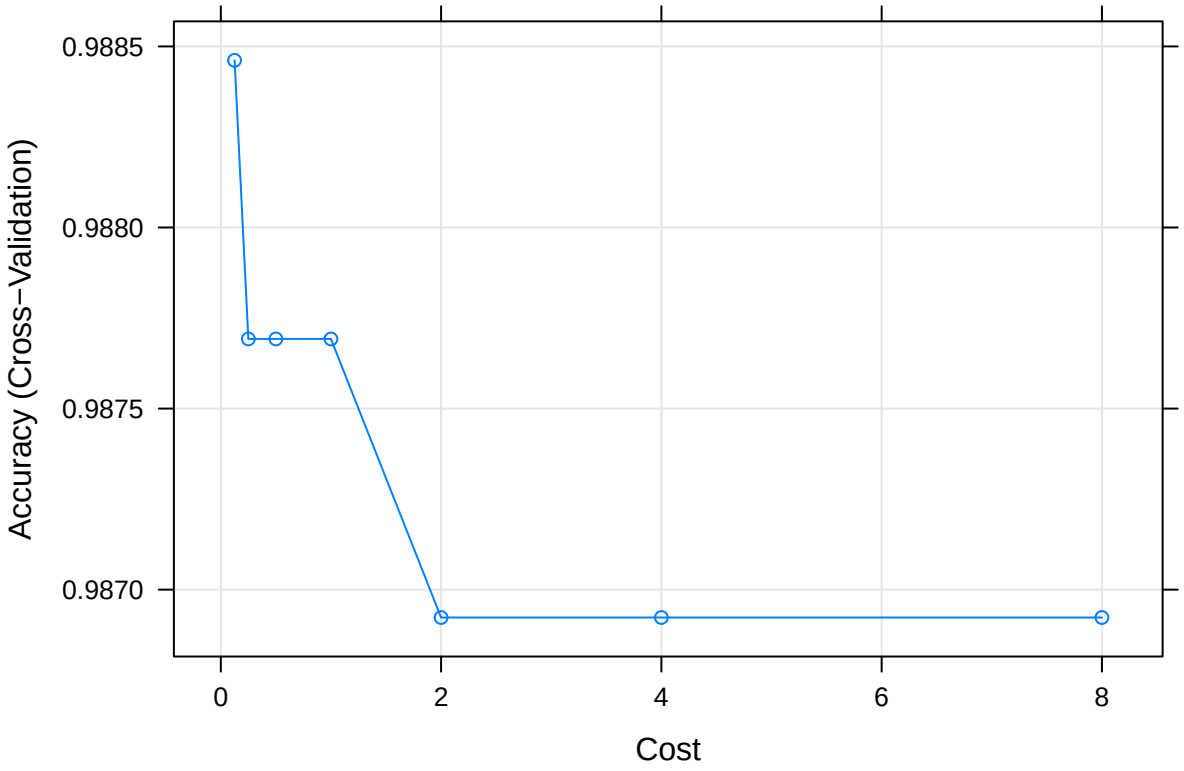
The performance is comparable to that GLM, again with only 13 wines out of 1,300 misclassified

```
red_white_results <- bind_rows(red_white_results, tibble(model = "Linear SVM", Accuracy = results_SVM$o
```

We will now try to see if we can extract better performance from the model by exploring wider values of the cost hyperparamater. By using powers of 2 we can examine values from 0.125 to 8 in merely seven iterations of the model.

```
set.seed(1236, sample.kind = "Rounding")


tuned_SVM <- train(type ~ .,
                   data = wine_test,
                   method = "svmLinear2",
                   type = "C-classification",
                   trControl = tr_Ctrl,
                   tuneGrid = data.frame(cost = 2^(-3:3)))
```

This model was trained with the same 5-fold cross validation, but this time using seven values for cost, ranging from 0.125 to 8. The impact of this on the accuracy is seen below.



The tabulated results for the expanded hyperparameter model:

```
tibble(Cost = tuned_SVM$results$cost, Accuracy = tuned_SVM$results$Accuracy) %>%
  knitr::kable(caption = "Cartesian grid search of cost hyperparamter") %>%
  kable_styling(latex_options = "HOLD_position")
```

Table 4: Cartesian grid search of cost hyperparamter

Cost	Accuracy
0.125	0.9884615
0.250	0.9876923
0.500	0.9876923
1.000	0.9876923
2.000	0.9869231
4.000	0.9869231
8.000	0.9869231

The expanded hyperparameter search revealed a slight improvement (to the third decimal place). This model now needs to be evaluated on the test data to establish if there is any improvement.

```
preds <- predict(tuned_SVM, newdata = wine_test)
(results_tuned_SVM <- confusionMatrix(wine_test$type, preds))
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction Red White
##      Red   313     7
##      White   4   976
##
##                Accuracy : 0.9915
##                  95% CI : (0.9849, 0.9958)
##     No Information Rate : 0.7562
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9771
##
##  Mcnemar's Test P-Value : 0.5465
##
##             Sensitivity : 0.9874
##             Specificity : 0.9929
##          Pos Pred Value : 0.9781
##          Neg Pred Value : 0.9959
##              Prevalence : 0.2438
##          Detection Rate : 0.2408
##    Detection Prevalence : 0.2462
##       Balanced Accuracy : 0.9901
##
##        'Positive' Class : Red
##
```

There is some improvement - there are now only eleven wines misclassified, a reduction of two (or optimistically an improvement of 15.3%). As the test data set represents 1,300 wines, an accuracy of 99.15% is quite remarkable.

Binary classification conclusion

Classification of wine into red and white based on underlying physiochemical properties seems to be an almost trivially easy problem. Even the simplest model performed at close to 100% accuracy. Applying a support vector machine with some minor hyperparameter tuning saw a minor increase in performance.

```
red_white_results %>%
  knitr::kable(caption = "Binary classifier results") %>%
  kable_styling(latex_options = "HOLD_position")
```

Table 5: Binary classifier results

model	Accuracy
GLM	0.9900000
Linear SVM	0.9900000
Tuned linear SVM	0.9915385

This part of the project was inspired by the now infamous experiment where wine students were given white wine dyed to appear red [10]. The students were unable to distinguish that the wine they were drinking was in fact a white wine and not a red wine. Though it should perhaps be seen as an interaction between our various senses; rather than an outright criticism of wine industry. Criticisms of that study are manifold and can be easily found.

However, it is unlikely that any blindfolded human could perform as well as any of the binary classifier models

[10] https://www.realclearscience.com/blog/2014/08/the_most_infamous_study_on_wine_tasting.html

outlined above when it comes to classifying wines as red or white. In conclusion there is clear underlying physiochemical differences between white wine and red wine that makes their distinction a trivial task for a computer.

It is time to turn to a far more interesting exercise - attempting to predict a wine's quality based on it's underlying physiochemical properties.

Techniques - Wine quality prediction

This second part of the project is perhaps the more interesting section of the report. We will attempt to predict the quality of the wine based on the underlying phsyiochemical properties. The first part of the project, despite largely being a quest of curiosity inspired by reports that wine experts couldn't distinguish between red and white wine, also had an important role in determining whether there is sufficient phsyiochemical differences between red and white for a computer to distinguish between them. As there is a clear difference, it would be appropriate to attempt to predict quality on white and red wine separately. The underlying differences between the wines could introduce "noise" to a predictor model and negatively impact on the performance of the model if we were to try predict quality independently of wine type.

This part of the project is complicated somewhat by the rating process. All the wines are rated as the median score of three experts who blind taste each wine. The use of the median leads to tight clustering of the data. The range in quality is not very large as was seen earlier. This makes distinguishing the outcomes considerably harder than simply classify wine as white or red.

Our models built now will be multi-class classifiers as the outcome variable "quality" is ordinal and not a continuous variable. Analogies could be made to the famous MNIST hand written digits data sets.

It would therefore be appropriate to convert the "quality" outcome variable to a factor. This will aid caret in determining the splits between training and test data as well assisting caret with selecting default hyperparameters.

```
red_wine$quality <- as.factor(red_wine$quality)

white_wine$quality <- as.factor(white_wine$quality)
```

The modelling process will once again begin with a simple model and build up complexity. Unfortunately the glm model is constrained as a binary classifier, so we will need to start off with a slightly more complex base. Proposed models to be used:

- Linear discriminant analysis (LDA). The MASS library will be used for this implementation.
- Random Forest. The randomForest library will be used despite some of its drawbacks discussed earlier.

Splitting the data

Red wine data partitioning Without going into a long discourse on the splitting of data into training and test set data, it must be acknowledged that the red wine data is considerably smaller with only 1,599 observations. In this case it would perhaps be more appropriate to hold back more data for testing. A 70% training set and 30% test set will be applied.

```
set.seed(1236, sample.kind = "Rounding")

index <- as.vector(createDataPartition(y = red_wine$quality, p = 0.3, list = FALSE))

red_wine_test <- red_wine[index,]
red_wine_train <- red_wine[-index,]

rm(index)
```

The red wine test set contains 482 samples and the training data 1117.

Due to the smaller data set for the red wine and the greater proportion of the data held back for testing, it would be appropriate to allow for greater cross validation in the tuning process. The model training control for the red wine will be cross-fold validation with ten iterations.

```
train_red_ctrl <- trainControl(method = "cv",
                               number = 10)
```

White wine data splitting Splitting the white wine into a test and train partition will be done on the 80% train, and 20% test split as the data set is considerably larger with 4898 samples in the data set.

```
set.seed(1236, sample.kind = "Rounding")

index <- as.vector(createDataPartition(y = white_wine$quality, p = 0.2, list = FALSE))

white_wine_test <- white_wine[index,]
white_wine_train <- white_wine[-index,]

rm(index)
```

The white wine test set is 981 and the training data is a quite hefty 3917. This could lead to some model training taking a little longer. The validation of the white wine training set can be reduced in comparison to the red wine.

As there is more data available for the training of the models for white wines, the training controls can be set so to be less computationally onerous as for the red wines. Once again cross validation will be used for training of the models but for this application only 5 fold cross validation will be used.

```
train_white_ctrl <- trainControl(method = "cv",
                                 number = 5)
```

The first predictive model of wine quality - LDA

A linear discriminant model is as close as we can come to a regression model for a multi-class classifier. A regression model when applied to a binary classification task works to place a plane in hyper-space which minimises the sum of square residuals between the the plane and each observed data point. With multiple possible outcomes this task can no longer be done by minimising the sum of squares of the residuals.

A linear discriminant analysis model is an attempt to generalise linear regression to accommodate multiple outputs. The model we will be using is the LDA from the MASS library. There are no hyperparameters to a LDA model making it a good place from which to start. An LDA is limited, the underlying assumption is the predictors are normally distributed around 0. This means the data will need to be centred and scaled before being passed to the model. Fortunately caret provides a preprocessing option which will allow for normalisation of the data.

LDA model for red wine quality Fitting and training the first model for wine quality is done very simply with the following code.

```
red_wine_lda_model <- train(quality ~ .,
                            data = red_wine_train,
                            method = "lda",
                            trControl = train_red_ctrl,
                            preProcess = c("center", "scale"))

preds <- predict(red_wine_lda_model, newdata = red_wine_test)

(results_red_LDA <- confusionMatrix(red_wine_test$quality, preds))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8
##          3   1   0   2   0   0   0
##          4   0   0   9   6   1   0
##          5   0   0 156  46   3   0
##          6   0   1  62 104  25   0
##          7   0   0   3  33  24   0
##          8   0   0   0   4   2   0
##
## Overall Statistics
##
##                Accuracy : 0.5913
##                  95% CI : (0.5459, 0.6355)
##     No Information Rate : 0.4813
##     P-Value [Acc > NIR] : 8.267e-07
##
##                   Kappa : 0.3424
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          1.000000 0.000000   0.6724   0.5389  0.43636       NA
## Specificity          0.995842 0.966736   0.8040   0.6955  0.91569  0.98755
## Pos Pred Value       0.333333 0.000000   0.7610   0.5417  0.40000       NA
## Neg Pred Value       1.000000 0.997854   0.7256   0.6931  0.92654       NA
## Prevalence           0.002075 0.002075   0.4813   0.4004  0.11411  0.00000
## Detection Rate       0.002075 0.000000   0.3237   0.2158  0.04979  0.00000
## Detection Prevalence 0.006224 0.033195   0.4253   0.3983  0.12448  0.01245
## Balanced Accuracy    0.997921 0.483368   0.7382   0.6172  0.67603       NA
```

The model provides reasonable out of the box performance - an accuracy rate of just beneath 60%. For an LDA model there is no opportunity to tune it further. |In the case of red wine there is some predictive power from a LDA model that surpasses the "No information rate". This term is a measure of what one would expect from a classifier with no prior exposure to the data and classified at random.

LDA for white wine quality Once again, as for the red wine, the model is quite easily fitted using the interface provided by caret. Obtaining results is also as straightforward.

```
white_wine_lda <- train(quality ~ .,
                        data = white_wine_train,
                        method = "lda",
                        trControl = train_white_ctrl,
                        preProcess = c("center", "scale"))

preds <- predict(white_wine_lda, newdata = white_wine_test)

(results_white_lda <- confusionMatrix(white_wine_test$quality, preds))
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction   3   4   5   6   7   8   9
##          3   0   0   1   3   0   0   0
##          4   0   3  15  13   2   0   0
##          5   3   9 135 141   3   1   0
##          6   0   2  85 310  42   0   1
##          7   0   0   8 120  48   0   0
##          8   0   0   1  22  12   0   0
##          9   0   0   0   0   1   0   0
##
## Overall Statistics
##
##                Accuracy : 0.5056
##                  95% CI : (0.4738, 0.5373)
##     No Information Rate : 0.6208
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2114
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.000000 0.214286   0.5510   0.5090  0.44444 0.000000
## Specificity          0.995910 0.968976   0.7867   0.6505  0.85338 0.964286
## Pos Pred Value       0.000000 0.090909   0.4623   0.7045  0.27273 0.000000
## Neg Pred Value       0.996929 0.988397   0.8403   0.4473  0.92547 0.998943
## Prevalence           0.003058 0.014271   0.2497   0.6208  0.11009 0.001019
## Detection Rate       0.000000 0.003058   0.1376   0.3160  0.04893 0.000000
## Detection Prevalence 0.004077 0.033639   0.2977   0.4485  0.17941 0.035678
## Balanced Accuracy    0.497955 0.591631   0.6689   0.5798  0.64891 0.482143
##                      Class: 9
## Sensitivity          0.000000
## Specificity          0.998980
## Pos Pred Value       0.000000
## Neg Pred Value       0.998980
## Prevalence           0.001019
## Detection Rate       0.000000
## Detection Prevalence 0.001019
## Balanced Accuracy    0.499490
```

Performance of the LDA models was never expected to be particularly fantastic. The white wine LDA model actually faired worse than one would have done by merely taking the average rating from the training set and applying it to the test set.

Second predictor model for wine quality - random forests

A random forest is an ensemble method for machine learning. It is an extension of the classic decision tree - multiple decision trees are built using a random selection of the feature variables. This allows the classification trees to be smaller (and thus less prone to over training). The number of trees is allowed to grow and is frequently around 500 different trees which "vote" on the final classification. There are a number of random forest implentations available in R. We will use the randomForest package implementation. This has one tuneable hyperparameter - "mtry" the number of variables randomly sampled at each tree split. The default value is the square root of the number of feature variables.

Random forest for red wine prediction Once again caret performs handles the fitting and training of the predictive models with ease. For the initial run the hyperparameter will be left to its default values on which caret trains the models. As this is a stochastic model a seed will need to be specified to obtain reproducible results.

```
set.seed(1236, sample.kind = "Rounding")

red_wine_rf <- train(quality ~ .,
                     data = red_wine_train,
                     method = "rf",
                     trControl = train_red_ctrl)

preds <- predict(red_wine_rf, newdata = red_wine_test)

(results_red_rf <- confusionMatrix(red_wine_test$quality, preds))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8
##          3   0   0   2   1   0   0
##          4   1   1   9   4   1   0
##          5   0   2 164  37   2   0
##          6   0   0  42 136  14   0
##          7   0   0   3  22  35   0
##          8   0   0   0   3   2   1
##
## Overall Statistics
##
##                Accuracy : 0.6992
##                  95% CI : (0.6561, 0.7398)
##     No Information Rate : 0.4564
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5178
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.000000 0.333333   0.7455   0.6700  0.64815 1.000000
## Specificity          0.993763 0.968685   0.8435   0.7993  0.94159 0.989605
## Pos Pred Value       0.000000 0.062500   0.8000   0.7083  0.58333 0.166667
## Neg Pred Value       0.997912 0.995708   0.7978   0.7690  0.95498 1.000000
## Prevalence           0.002075 0.006224   0.4564   0.4212  0.11203 0.002075
## Detection Rate       0.000000 0.002075   0.3402   0.2822  0.07261 0.002075
## Detection Prevalence 0.006224 0.033195   0.4253   0.3983  0.12448 0.012448
## Balanced Accuracy    0.496881 0.651009   0.7945   0.7346  0.79487 0.994802
```

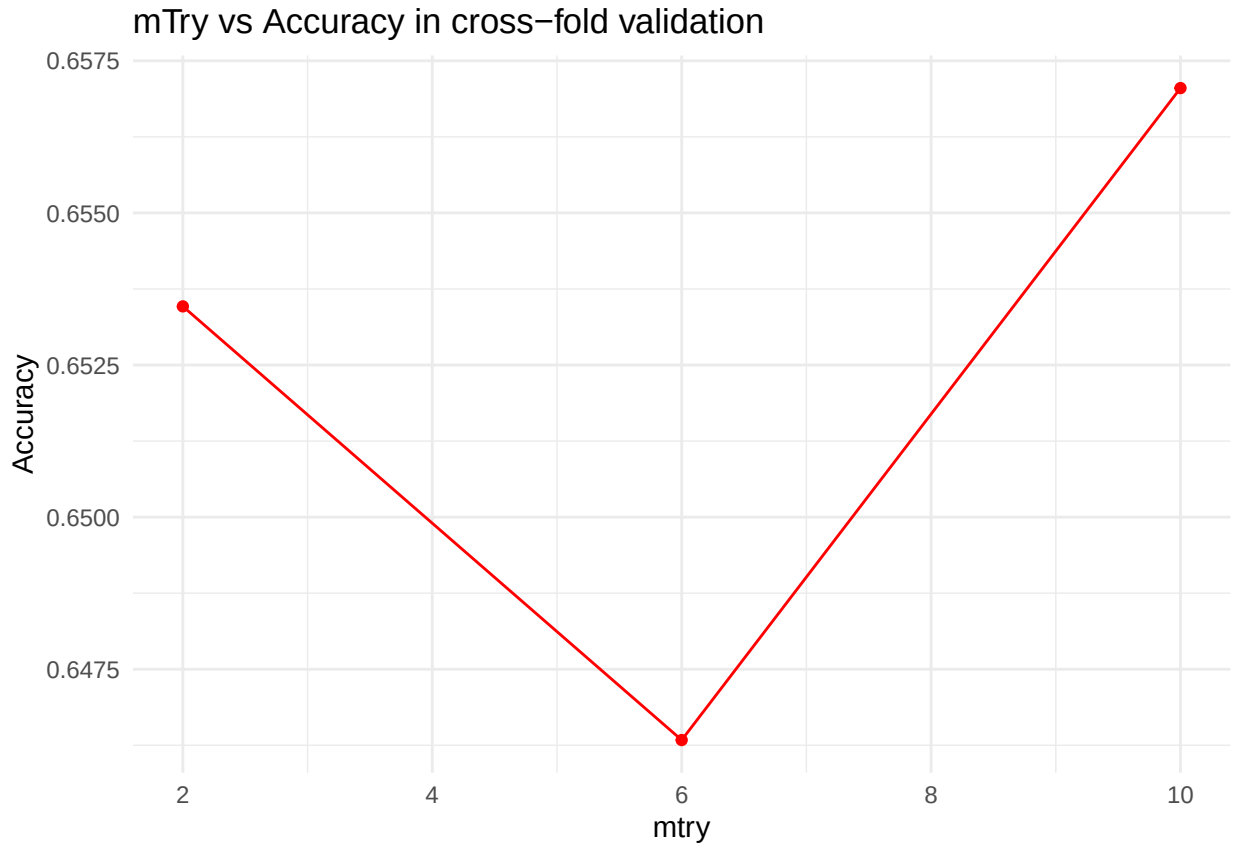
By default caret only fits three different values for "mtry". The results for the default values are shown below.

```
red_wine_rf$results[,1:2] %>%
  knitr::kable(caption = "Default tuning parameters for red wine random forest") %>%
  kable_styling(latex_options = "HOLD_position")
```

Table 6: Default tuning parameters for red wine random forest

mtry	Accuracy
2	0.6534639
6	0.6463363
10	0.6570516

The graph of these these values shows the best value of "mtry" as being 10. However, only three points have been sampled and it is possible that being trying further values between 2-10 that a different local maximum accuracy could be found.



To give the model the best chance to find an optimal solution we need to give it more options to try more values of "mtry". We will just allow caret to do a search over all possible values of "mtry"

```
set.seed(1236, sample.kind = "Rounding")

red_wine_tunedRF <- train(quality ~ .,
                          data = red_wine_train,
                          method = "rf",
                          trControl = train_red_ctrl,
                          tuneGrid = data.frame(mtry = 2:10))

preds <- predict(red_wine_tunedRF, newdata = red_wine_test)

(results_red_wine_tunedRF <- confusionMatrix(red_wine_test$quality, preds))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8
##          3   0   0   2   1   0   0
##          4   1   1   9   4   1   0
##          5   0   0 166  37   2   0
##          6   0   1  45 136  10   0
##          7   0   0   2  23  35   0
##          8   0   0   0   2   3   1
##
## Overall Statistics
##
##                Accuracy : 0.7033
##                  95% CI : (0.6603, 0.7438)
##     No Information Rate : 0.4647
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5224
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.000000 0.500000   0.7411   0.6700  0.68627 1.000000
## Specificity          0.993763 0.968750   0.8488   0.7993  0.94200 0.989605
## Pos Pred Value       0.000000 0.062500   0.8098   0.7083  0.58333 0.166667
## Neg Pred Value       0.997912 0.997854   0.7906   0.7690  0.96209 1.000000
## Prevalence           0.002075 0.004149   0.4647   0.4212  0.10581 0.002075
## Detection Rate       0.000000 0.002075   0.3444   0.2822  0.07261 0.002075
## Detection Prevalence 0.006224 0.033195   0.4253   0.3983  0.12448 0.012448
## Balanced Accuracy    0.496881 0.734375   0.7950   0.7346  0.81413 0.994802
```

Allowing to caret to iterate over further values of "mtry" resulted in an improved accuracy to just over 70% on the final test set.

The values of mtry and their accuracies obtained during cross validation can be seen below.

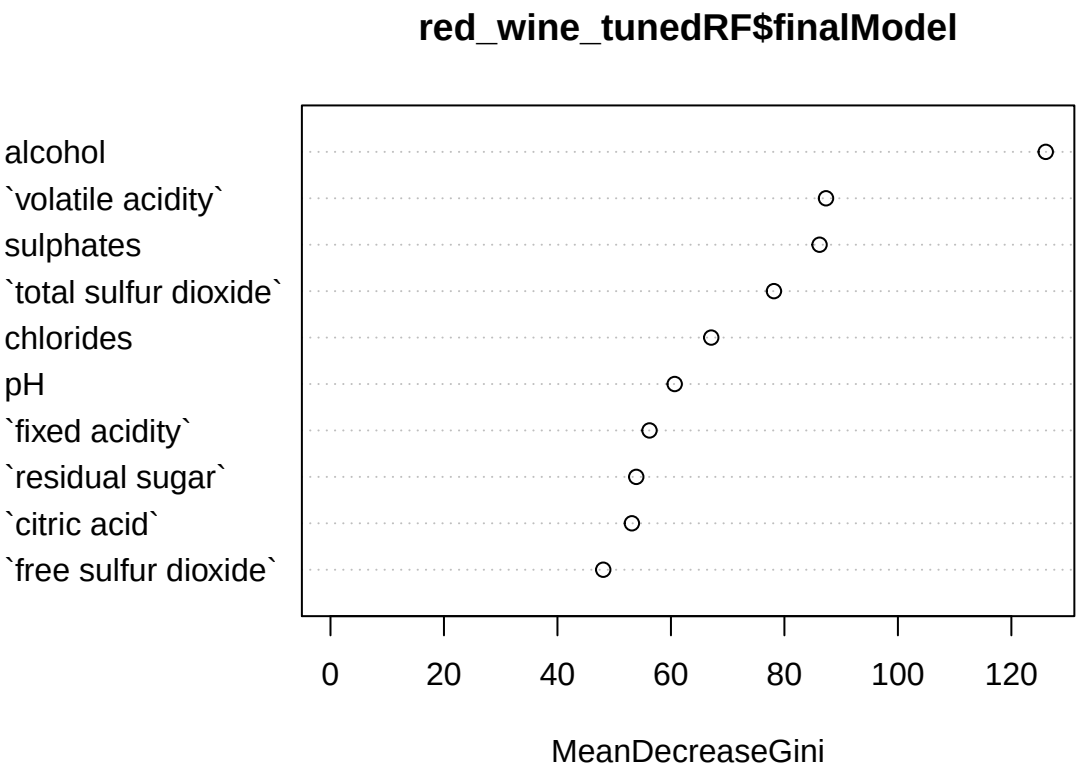
Table 7: Exapnded mtry results

mtry	Accuracy
2	0.6597382
3	0.6614998
4	0.6560945
5	0.6508655
6	0.6615159
7	0.6418072
8	0.6480097
9	0.6516297
10	0.6525301

A graph depicting these results will show how the accuracy moved around under different values of mtry.

The random forest with further freedom did return better accuracy than the default random forest, although the improvement was not enormously significant.

The advantage of the randomForest implementation of the random forest algorithm is that it is slightly more transparent than other random forest implementations. We can see the importance of each variable to the model.



Surprisingly the alcohol content is the most important factor in the random forest model we have built. There is not much separating the remaining physiochemical factors.

Random forest for white wine quality prediction Before proceeding with the modelling process for the white wine, given the poor performance of the LDA, it would be appropriate to amend the training control. The cross validation will be increased to ten fold to give a model better chance to find better solutions. This will add to computational burden.

```
train_white_ctrl <- trainControl(method = "cv",
                                 number = 10)
```

As before we will start with a random forest model using the default hyperparameters within caret for the randomForest model implementation. A seed will be passed to the random number generator to ensure reproducibility of results.

```
set.seed(1236, sample.kind = "Rounding")

white_wine_rf <- train(quality ~ .,
                       data = white_wine_train,
                       method = "rf",
                       trControl = train_white_ctrl)

preds <- predict(white_wine_rf, newdata = white_wine_test)

(results_white_rf <- confusionMatrix(white_wine_test$quality, preds))
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction   3   4   5   6   7   8   9
##          3   0   0   2   2   0   0   0
##          4   0   6  12  14   1   0   0
##          5   0   1 201  87   3   0   0
##          6   0   0  61 363  16   0   0
##          7   0   0   1  83  92   0   0
##          8   0   0   0  13   3  19   0
##          9   0   0   0   0   1   0   0
##
## Overall Statistics
##
##                Accuracy : 0.6942
##                  95% CI : (0.6643, 0.7229)
##     No Information Rate : 0.5729
##     P-Value [Acc > NIR] : 3.545e-15
##
##                   Kappa : 0.5198
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity                NA 0.857143   0.7256   0.6459  0.79310  1.00000
## Specificity          0.995923 0.972279   0.8707   0.8162  0.90289  0.98337
## Pos Pred Value             NA 0.181818   0.6884   0.8250  0.52273  0.54286
## Neg Pred Value             NA 0.998945   0.8897   0.6322  0.97019  1.00000
## Prevalence           0.000000 0.007136   0.2824   0.5729  0.11825  0.01937
## Detection Rate       0.000000 0.006116   0.2049   0.3700  0.09378  0.01937
## Detection Prevalence 0.004077 0.033639   0.2977   0.4485  0.17941  0.03568
## Balanced Accuracy          NA 0.914711   0.7982   0.7311  0.84800  0.99168
##                      Class: 9
## Sensitivity                NA
## Specificity          0.998981
## Pos Pred Value             NA
## Neg Pred Value             NA
## Prevalence           0.000000
## Detection Rate       0.000000
## Detection Prevalence 0.001019
## Balanced Accuracy          NA
```

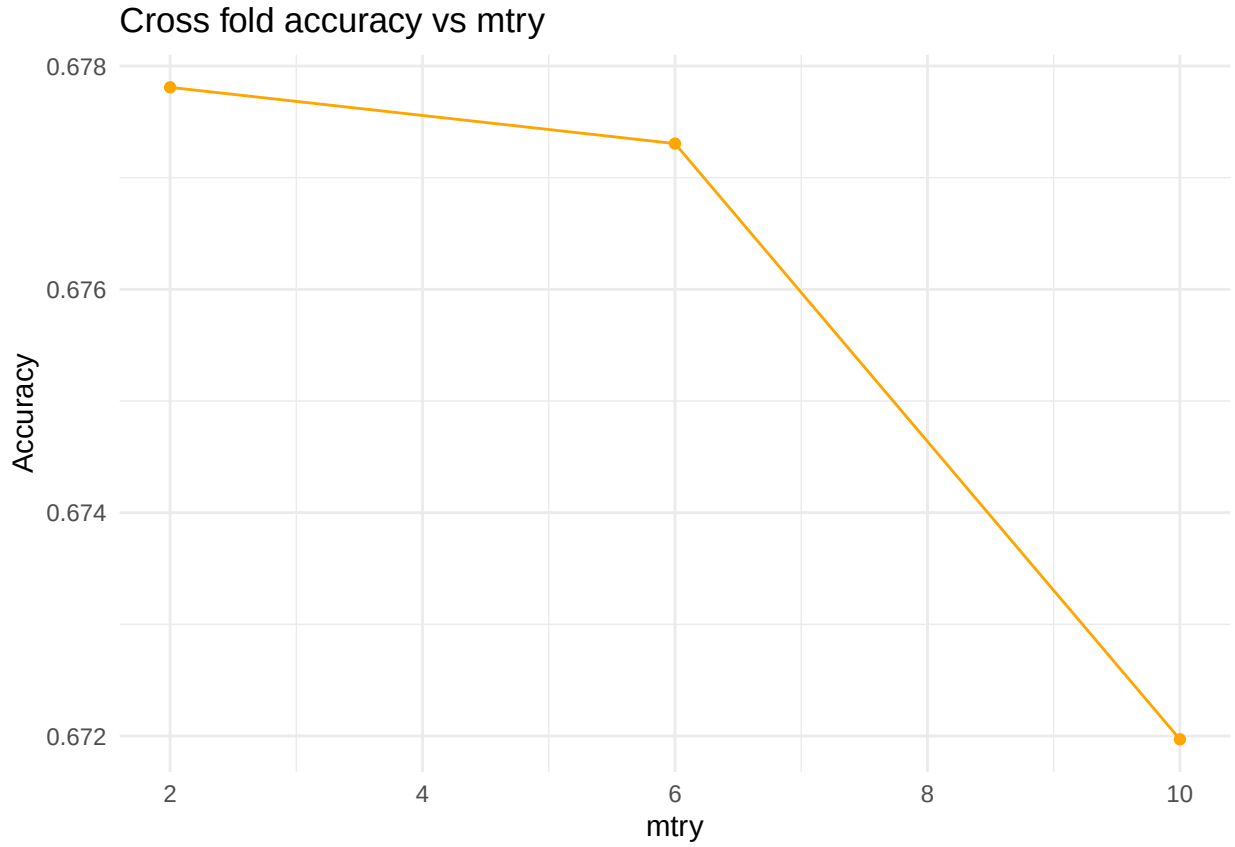
There is a staggering improvement in performance in terms of accuracy over the LDA model. Total accuracy using the default hyperparameters showed a model accuracy of 69.42%.

A brief look at the accuracy of the cross validation, for each value of mtry used by caret.

```
white_wine_rf$results[,1:2] %>%
  knitr::kable(caption = "Mtry vs cross validation accuracy") %>%
  kable_styling(latex_options = "HOLD_position")
```

Table 8: Mtry vs cross validation accuracy

mtry	Accuracy
2	0.6778091
6	0.6773055
10	0.6719704



As we saw when applying an extension of options of the mtry hyperparameter to the random forest for the red wine we were able to improve overall classification performance. We will do likewise now with the white wine.

We will now proceed to build the random forest with the full range of plausible values of mtry (2-10). This model will take a little while to run depending on the hardware on which it is being executed.

```
set.seed(1236, sample.kind = "Rounding")

white_wine_tunedRF <- train(quality ~ .,
                            data = white_wine_train,
                            method = "rf",
                            trControl = train_white_ctrl,
                            tuneGrid = data.frame(mtry = 2:10))

preds <- predict(white_wine_tunedRF, newdata = white_wine_test)

(results_white_tunedRF <- confusionMatrix(white_wine_test$quality, preds))
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction   3   4   5   6   7   8   9
##          3   0   0   2   2   0   0   0
##          4   0   6  12  15   0   0   0
##          5   0   1 202  87   2   0   0
##          6   0   0  60 365  15   0   0
##          7   0   0   1  82  93   0   0
##          8   0   0   0  13   3  19   0
##          9   0   0   0   0   1   0   0
##
## Overall Statistics
##
##                Accuracy : 0.6983
##                  95% CI : (0.6685, 0.7269)
##     No Information Rate : 0.5749
##     P-Value [Acc > NIR] : 1.137e-15
##
##                   Kappa : 0.5258
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity                NA 0.857143   0.7292   0.6472   0.8158  1.00000
## Specificity          0.995923 0.972279   0.8722   0.8201   0.9043  0.98337
## Pos Pred Value             NA 0.181818   0.6918   0.8295   0.5284  0.54286
## Neg Pred Value             NA 0.998945   0.8911   0.6322   0.9739  1.00000
## Prevalence           0.000000 0.007136   0.2824   0.5749   0.1162  0.01937
## Detection Rate       0.000000 0.006116   0.2059   0.3721   0.0948  0.01937
## Detection Prevalence 0.004077 0.033639   0.2977   0.4485   0.1794  0.03568
## Balanced Accuracy          NA 0.914711   0.8007   0.7337   0.8600  0.99168
##                      Class: 9
## Sensitivity                NA
## Specificity          0.998981
## Pos Pred Value             NA
## Neg Pred Value             NA
## Prevalence           0.000000
## Detection Rate       0.000000
## Detection Prevalence 0.001019
## Balanced Accuracy          NA
```

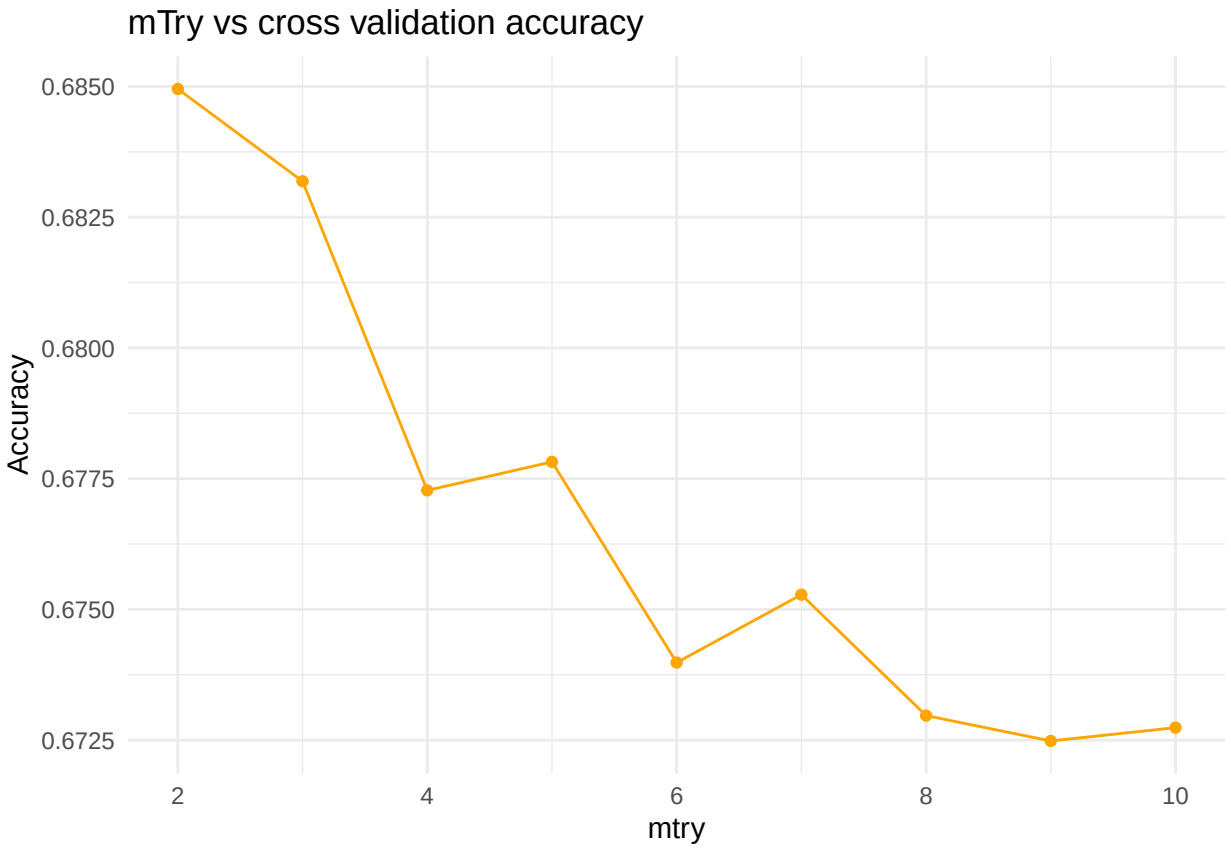
There has been a marginal improvement in performance. Encourangingly for such a tightly bound data range most values in the confusion matrix are clustered around the main diagonal.

A brief overview of how the accuracy profile changed with the additional values passed to "mtry" is set out below.

```
white_wine_tunedRF$results[,1:2] %>%
  knitr::kable(caption = "mTry results 2-10") %>%
  kable_styling(latex_options = "HOLD_position")
```

Table 9: mTry results 2-10

mtry	Accuracy
2	0.6849508
3	0.6831865
4	0.6772762
5	0.6778209
6	0.6739832
7	0.6752816
8	0.6729699
9	0.6724845
10	0.6727389



There is a curiosity that arises from the stochastic nature of the random forest algorithms. Both the "default" model and the model which was allowed to tune over the entire range of "mtry" returned the same value of "mtry" (in this instance 2), but due to the random nature of the process the second model performed better on the test data than the first model despite using the same random seed and same hyperparameter values.

Quality prediction conclusion

Not unsurprisingly the first attempt to model the quality of a wine using a linear discriminant analysis model was fairly unsuccessful. Though interestingly enough it performed far better in the instance of red wine. As discussed earlier this is a particularly challenging task for machine learning. Only 11 eleven physiochemical predictors were recorded. It is estimated that most wine can contain between 800-1,000 distinct chemicals[11].

[11] https://www.compoundchem.com/2014/05/28/redwinechemicals

Additional physiochemicals observations may have made this an easier data set to classify.

However given the nature of the range of the quality being so tightly clustered, to have random forest models perform with just shy of 70% accuracy is, in fact, quite remarkable. All the random forest models performed well above the no information rate. The misclassifications were also closely clustered around the main diagonals of the confusion matrices, indicating there were little margins within the classifications.

A summary of the models is set out below.

```
quality_results %>%
  knitr::kable(caption = "Final results") %>%
  kable_styling(latex_options = "HOLD_position")
```

Table 10: Final results

Model	Wine_type	Accuracy
LDA	Red	0.5912863
LDA	White	0.5056065
Default RF	Red	0.6991701
Tuned RF	Red	0.7033195
Default RF	White	0.6941896
Tuned RF	White	0.6982671

Project conclusions

This project set out to answer two questions:

- Could a computer be trained to distinguish between red and white wine?
- Could a computer then be trained to predict the quality of a wine?

The first question was shown to be an almost trivial problem for a modern machine learning algorithm. Accuracy obtained on classifying red from white was close to 100%. As mentioned earlier, humans cannot necessarily come close in this regard.

The answer to the second question is a little more subtle and nuanced. A machine learning algorithm definitely performs better than merely taking an average or guessing. The nature of the data does however make extreme accuracy very difficult to achieve. *P. Cortez et al.* suggested an accuracy of 90% was possible with a linear support vector machine. No linear SVM tested was able to perform better than the no information rate. In preparing this report radial SVM's were also explored and they also failed to deliver performance close to the random forests. As alluded to earlier the misclassifications also tended to fall close to the main diagonals of the confusion matrices.

Should machine learning algorithms be used to predict wine quality? The answer to this is almost certainly not - wine, like most things edible, is entirely subjective. it could however, if you happened to have the 11 physiochemical measures of your prospective tipple to hand, give you a broad indication if it would likely be a good or bad wine according to three anonymous judges.