An algorithmic approach to determining wine quality from chemical data

David Gold

November 2019

Introduction

"Best Wines in the World" (BWITW) is a wine club that prides itself on being able to provide new quality wines to its members at reasonable prices. It currently relies on testers to judge the quality of its wines which is a slow, time consuming and expensive process. Also, by the time that it has tested new wines the market is usually aware of them and the prices rise. It has commissioned this report to assess the viability of using chemical analysis, combined with data science, to determine whether a wine is "quality" or not quickly and cheaply, so that it can get access to new, competively priced and quality wines before its competition. The management of BWITW consider a wine to be "quality" if it receives a score of 7 out of 10 by a panel of testers.

This report details analysis that has been done to determine whether this combined approach of chemical testing and data science would work and makes a judgement on suitability of this approach and possible next steps.

BWITW have gained access to a dataset of the chemical properties of red wines and a quality score for each wine given by a panel of testers. The approach taken in this paper was to first investigate the dataset to understand its structure and the variables within it, including some visualisations. A very simple regression was performed to see what would be simply possible using very simple techniques. After this a range of more sophisticated machine learning algorithms were applied to see whether this would lead to better results. From these results conclusions have been drawn.

NOTE: The data set used here is from https://archive.ics.uci.edu/ml/datasets/Wine+Quality and 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 is cited.

Methods and Analysis

The first step in the analysis was to ensure that necessary packages and the dataset were downloaded. The code for this is below.

```r
## load packages that are required

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

## set seed at the start to ensure consistency of output

set.seed(8934, sample.kind = "Rounding")

## download the red wine quality dataset
```

```
wine_quality <- read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequa
                         sep = ";")
```

After this, the size and shape of the dataset were investigated.

```
## have an initial look at the dataset

head(wine_quality)
```

```
##   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
## 1       5
## 2       5
## 3       5
## 4       6
## 5       5
## 6       5
```

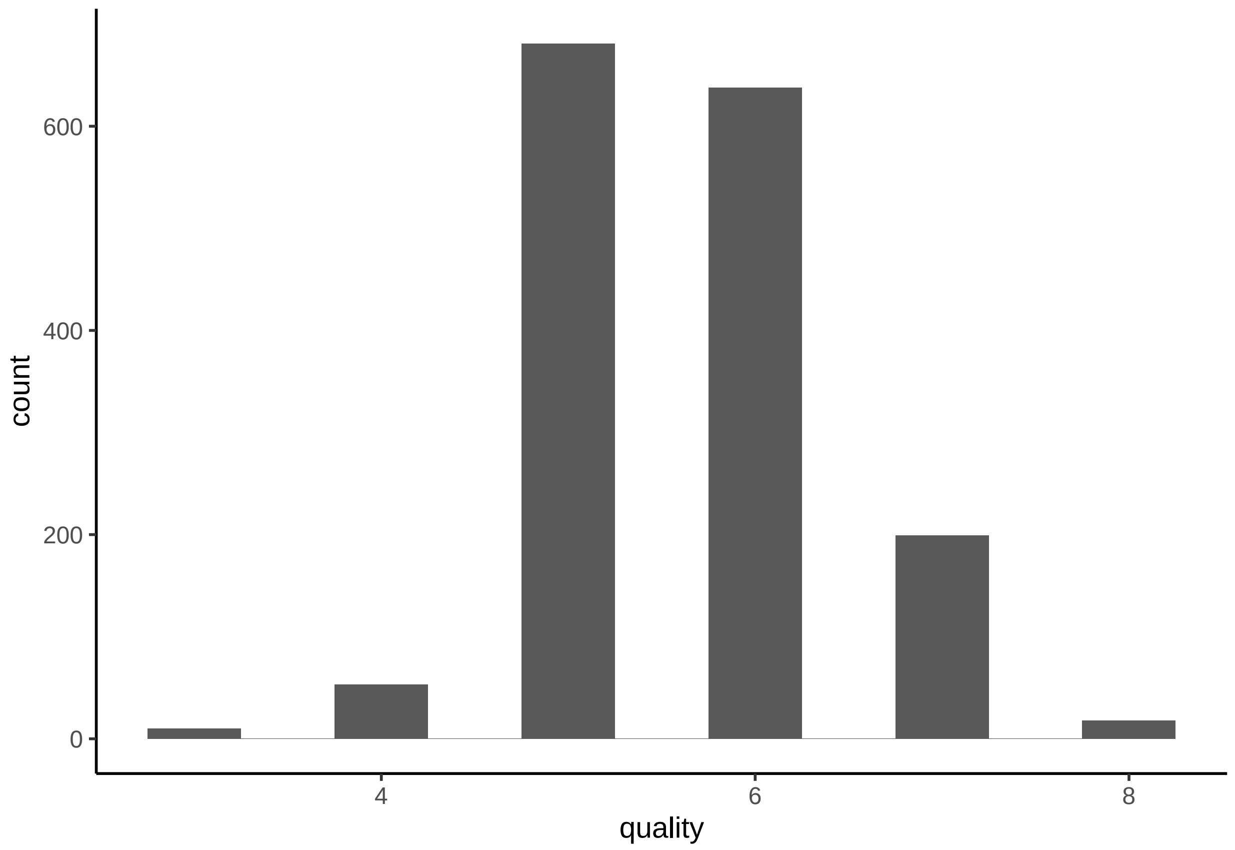
```
dim(wine_quality)
```

```
## [1] 1599   12
```

It can be seen that there are 1599 records and 12 columns with observations. One of the columns is the wine quality which is a numeric score.

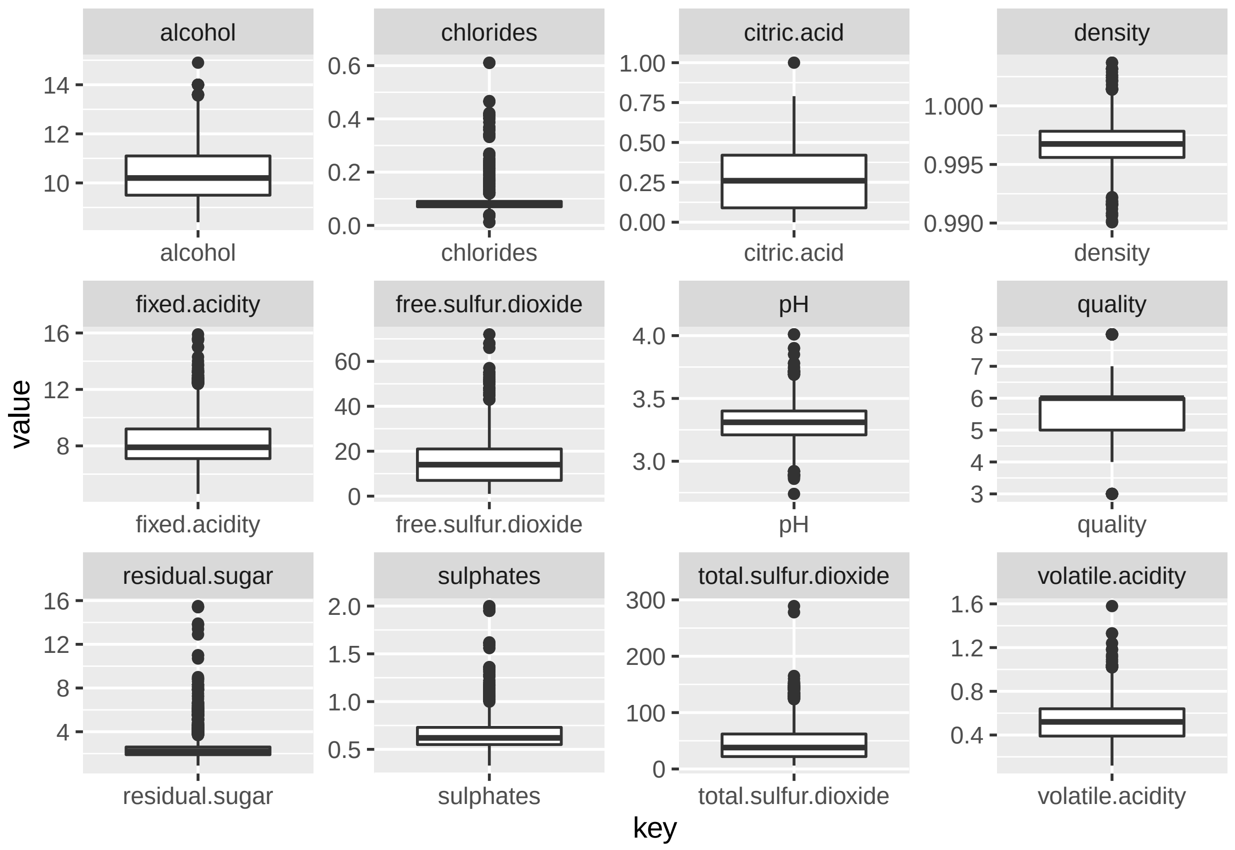
The dataset was investigated with a series of visualisations

```
# first up - take a look at the quality
wine_quality %>% ggplot(aes(x = quality)) +
  geom_histogram(binwidth = 0.5) +
  theme_classic()
```



```
# next - try a box plot to understand the distribution of the different variables
# that we have data for and may determine quality

wine_quality %>%
  gather(key, value) %>%
  ggplot(aes(y = value, x = key)) +
  geom_boxplot() +
  facet_wrap(~ key, scales = "free")
```



It can be seen that the quality scores approximate a normal distribution and that most scores are near the mean with a 5 or a 6. The other variables in the dataset representing chemical properties/readings are distributed in a range of different ways as shown in the box plots.

BWITW consider a wine to be quality if it receives a score from their panel of testers of 7 or over, so this is defined as success and a new factor with two levels: "quality" where the wine has a score of 7 or over and "sub-quality" where is does not is created.

```r
# define a quality wine as having a score of 7 or above

y <- if_else(wine_quality$quality >= 7, "quality", "sub-quality")
y <- factor(y)
data.frame(y) %>%
  group_by(y) %>%
  summarise(n()) %>%
  knitr::kable()
```

y	n()
quality	217
sub-quality	1382

There are a total of 217 quality wines in the dataset.

The dataset was split into a training set (80%) and a test set (20%) before starting to train models.

```r
# Validation set will be 20% of the total dataset

test_index <- createDataPartition(y = wine_quality$quality, times = 1, p = 0.2, list = FALSE)
training_set <- wine_quality[-test_index,]
test_set <- wine_quality[test_index,]
y_training_set <- y[-test_index]
y_test_set <- y[test_index]
```

As a bit of further data analysis, a test was done to check that the percentage of quality wines was roughly the same in both the training set and the test set.

```r
# check the proportion of the wines that are quality in both the test set and the training set

mean(y_training_set == "quality")
```

```
## [1] 0.1353678
```

```r
mean(y_test_set == "quality")
```

```
## [1] 0.1370717
```

This shows that in both cases, the proportion of "quality" wines is around 14%.

The next step was for an analysis of the correlation coefficients between the various variables and the quality scores for the wines.

```r
# lets now start with the simplest predictive model and see whether a simple correlation would
# give a high degree of accuracy

cor(wine_quality[1:11], wine_quality[12]) %>%
  knitr::kable()
```

	quality
fixed.acidity	0.1240516
volatile.acidity	-0.3905578
citric.acid	0.2263725
residual.sugar	0.0137316
chlorides	-0.1289066
free.sulfur.dioxide	-0.0506561
total.sulfur.dioxide	-0.1851003
density	-0.1749192
pH	-0.0577314
sulphates	0.2513971
alcohol	0.4761663

It can be seen from this that the alcohol content of the wine is the most correlated variable with the wine quality and hence this could be the basis for an initial simple regression model. However, the classification of a wine into "quality" or not is categorical rather than continuous and hence a relevant model needs to be applied here (and to more complicated models that are developed).

The regression approach can be extended to categorical data. For example, we can try regression to estimate the conditional probability:

$p(x) = Pr(Y = 1|X = x) = \beta_0 + \beta_1 x$

Once we have estimates β_0 and β_1, we can obtain an actual prediction $p(x)$. Then we can define a specific decision rule to form a prediction. The logical decision rule to apply here would be if $\hat{p} > 0.5$ and this is done in the code below.

We now need to define success for model. There are several measures that will be used as part of a judgement on the best approach. The accuracy will be looked at, as well as the sensitivity, specificity and the balance accuracy. In the context of this exercise:

- the **accuracy** represents the total proportion of wine samples that were correctly identified using that approach.
- the **sensitivity** is the true positive rate and represents the proportion of quality wines that are correctly identified as such.
- the **specificity** is the true negative rate and measures the proportion of non-quality wines identified as such.
- the **balanced accuracy** makes an adjustment for the fact that there are significantly less "quality" than "sub-quality" wines in the dataset and in effect normalises the results for this and hence shows the proportion of wines that would be correctly identified in a balanced dataset.
- the **positive predicted accuracy** is the proportion of wines that are predicted to be "quality" that actually are.
- the **negative predicted accuracy** is the proportion of wines that are predicted to be "sub-quality" that actually are.

The code below also creates a dataframe to store the results from the analysis.

```r
# of these variables, it appears that alcohol has the highest correlation with quality,
# so let's build a predictive model
# based on this simple correlation

lm_fit <- mutate(training_set, y = as.numeric(quality >= 7)) %>% lm(y ~ alcohol, data = .)
p_hat <- predict(lm_fit, test_set)
y_hat <- ifelse(p_hat > 0.5, "quality", "sub-quality") %>% factor()
mean(y_hat == y_test_set)
wine_results <- data_frame(method = "regression on alcohol",
                           accuracy =
                             confusionMatrix(y_hat, y_test_set)$overall["Accuracy"],
                           sensitivity =
                             confusionMatrix(y_hat, y_test_set)$byClass["Sensitivity"],
                           specificity =
                             confusionMatrix(y_hat, y_test_set)$byClass["Specificity"],
                           balanced_acc =
                             confusionMatrix(y_hat, y_test_set)$byClass["Balanced Accuracy"],
                           positive_pred_val =
                             confusionMatrix(y_hat, y_test_set)$byClass["Pos Pred Value"],
                           negative_pred_val =
                             confusionMatrix(y_hat, y_test_set)$byClass["Neg Pred Value"])
```

The simple regression model on alcohol gives a base against which to compare the other approaches. The following machine learning algorithms/approaches were subsequently run using all the variables in the dataset:

- A Generalised Linear Model

- A Linear Discriminant Analysis Model
- A Quadratic Discriminant Analysis Model
- A Generalized Additive Model using LOESS
- A Random Forest Model
- A k-Nearest Neighbours Model
- An AdaBoost Classification Trees Model
- A Neural Network Model
- An Ensemble Model which takes the average results of all the models above

The code to generate these is shown below and note that it does take some time to run. The first section in the code below removes the quality score from the dataset, so that this is not included in dataset as it is obviously highly correlated with the "quality" classification that has been established.

```
# Remove the quality value from the data table to ease the application of machine learning approaches

training_set <- training_set[,-12]
test_set <- test_set[,-12]

# start with a simple linear regression model to see what results that will give

train_glm <- train(training_set, y_training_set,
                   method = "glm")
glm_preds <- predict(train_glm, test_set)
mean(glm_preds == y_test_set)
wine_results <- bind_rows(wine_results,
                          data_frame(method="general linear model",
                                     accuracy =
                                       confusionMatrix(glm_preds,
                                                       y_test_set)$overall["Accuracy"],
                                     sensitivity =
                                       confusionMatrix(glm_preds,
                                                       y_test_set)$byClass["Sensitivity"],
                                     specificity =
                                       confusionMatrix(glm_preds,
                                                       y_test_set)$byClass["Specificity"],
                                     balanced_acc =
                                       confusionMatrix(glm_preds,
                                                       y_test_set)$byClass["Balanced Accuracy"],
                                     positive_pred_val =
                                       confusionMatrix(glm_preds,
                                                       y_test_set)$byClass["Pos Pred Value"],
                                     negative_pred_val =
                                       confusionMatrix(glm_preds,
                                                       y_test_set)$byClass["Neg Pred Value"]))

# next try the LDA model

train_lda <- train(training_set, y_training_set,
                   method = "lda")
lda_preds <- predict(train_lda, test_set)
mean(lda_preds == y_test_set)
wine_results <- bind_rows(wine_results,
                          data_frame(method="LDA model",
                                     accuracy =
                                       confusionMatrix(lda_preds,
                                                       y_test_set)$overall["Accuracy"],
                                     sensitivity =
                                       confusionMatrix(lda_preds,
                                                       y_test_set)$byClass["Sensitivity"],
                                     specificity =
                                       confusionMatrix(lda_preds,
                                                       y_test_set)$byClass["Specificity"],
                                     balanced_acc =
                                       confusionMatrix(lda_preds,
                                                       y_test_set)$byClass["Balanced Accuracy"],
                                     positive_pred_val =
                                       confusionMatrix(lda_preds,
                                                       y_test_set)$byClass["Pos Pred Value"],
                                     negative_pred_val =
                                       confusionMatrix(lda_preds,
                                                       y_test_set)$byClass["Neg Pred Value"]))


# and the QDA model

train_qda <- train(training_set, y_training_set,
                   method = "qda")
qda_preds <- predict(train_qda, test_set)
mean(qda_preds == y_test_set)
wine_results <- bind_rows(wine_results,
                          data_frame(method="QDA model",
                                     accuracy =
                                       confusionMatrix(qda_preds,
                                                       y_test_set)$overall["Accuracy"],
                                     sensitivity =
                                       confusionMatrix(qda_preds,
                                                       y_test_set)$byClass["Sensitivity"],
                                     specificity =
                                       confusionMatrix(qda_preds,
                                                       y_test_set)$byClass["Specificity"],
                                     balanced_acc =
                                       confusionMatrix(qda_preds,
                                                       y_test_set)$byClass["Balanced Accuracy"],
                                     positive_pred_val =
                                       confusionMatrix(qda_preds,
                                                       y_test_set)$byClass["Pos Pred Value"],
                                     negative_pred_val =
                                       confusionMatrix(qda_preds,
                                                       y_test_set)$byClass["Neg Pred Value"]))

# and a Loess model

train_loess <- train(training_set, y_training_set,
                   method = "gamLoess")
loess_preds <- predict(train_loess, test_set)
mean(loess_preds == y_test_set)
wine_results <- bind_rows(wine_results,
                          data_frame(method="Loess",
                                     accuracy =
                                       confusionMatrix(loess_preds,
                                                       y_test_set)$overall["Accuracy"],
                                     sensitivity =
                                       confusionMatrix(loess_preds,
                                                       y_test_set)$byClass["Sensitivity"],
                                     specificity =
                                       confusionMatrix(loess_preds,
                                                       y_test_set)$byClass["Specificity"],
                                     balanced_acc =
                                       confusionMatrix(loess_preds,
                                                       y_test_set)$byClass["Balanced Accuracy"],
                                     positive_pred_val =
                                       confusionMatrix(loess_preds,
                                                       y_test_set)$byClass["Pos Pred Value"],
                                     negative_pred_val =
                                       confusionMatrix(loess_preds,
                                                       y_test_set)$byClass["Neg Pred Value"]))

# and a random forest model

train_rf <- train(training_set, y_training_set,
                     method = "rf")
rf_preds <- predict(train_rf, test_set)
mean(rf_preds == y_test_set)
wine_results <- bind_rows(wine_results,
                          data_frame(method="Random Forest",
                                     accuracy =
                                       confusionMatrix(rf_preds,
                                                       y_test_set)$overall["Accuracy"],
                                     sensitivity =
                                       confusionMatrix(rf_preds,
                                                       y_test_set)$byClass["Sensitivity"],
                                     specificity =
                                       confusionMatrix(rf_preds,
                                                       y_test_set)$byClass["Specificity"],
                                     balanced_acc =
                                       confusionMatrix(rf_preds,
                                                       y_test_set)$byClass["Balanced Accuracy"],
                                     positive_pred_val =
                                       confusionMatrix(rf_preds,
                                                       y_test_set)$byClass["Pos Pred Value"],
                                     negative_pred_val =
                                       confusionMatrix(rf_preds,
                                                       y_test_set)$byClass["Neg Pred Value"]))


# and a knn model

train_knn <- train(training_set, y_training_set,
                  method = "knn")
knn_preds <- predict(train_knn, test_set)
```

```r
mean(knn_preds == y_test_set)
wine_results <- bind_rows(wine_results,
                          data_frame(method="k nearest neighbour",
                                     accuracy =
                                       confusionMatrix(knn_preds,
                                                       y_test_set)$overall["Accuracy"],
                                     sensitivity =
                                       confusionMatrix(knn_preds,
                                                       y_test_set)$byClass["Sensitivity"],
                                     specificity =
                                       confusionMatrix(knn_preds,
                                                       y_test_set)$byClass["Specificity"],
                                     balanced_acc =
                                       confusionMatrix(knn_preds,
                                                       y_test_set)$byClass["Balanced Accuracy"],
                                     positive_pred_val =
                                       confusionMatrix(knn_preds,
                                                       y_test_set)$byClass["Pos Pred Value"],
                                     negative_pred_val =
                                       confusionMatrix(knn_preds,
                                                       y_test_set)$byClass["Neg Pred Value"]))

# and an adaboost model

train_adaboost <- train(training_set, y_training_set,
                   method = "adaboost")
adaboost_preds <- predict(train_adaboost, test_set)
mean(adaboost_preds == y_test_set)
wine_results <- bind_rows(wine_results,
                          data_frame(method="adaboost",
                                     accuracy =
                                       confusionMatrix(adaboost_preds,
                                                       y_test_set)$overall["Accuracy"],
                                     sensitivity =
                                       confusionMatrix(adaboost_preds,
                                                       y_test_set)$byClass["Sensitivity"],
                                     specificity =
                                       confusionMatrix(adaboost_preds,
                                                       y_test_set)$byClass["Specificity"],
                                     balanced_acc =
                                       confusionMatrix(adaboost_preds,
                                                       y_test_set)$byClass["Balanced Accuracy"],
                                     positive_pred_val =
                                       confusionMatrix(adaboost_preds,
                                                       y_test_set)$byClass["Pos Pred Value"],
                                     negative_pred_val =
                                       confusionMatrix(adaboost_preds,
                                                       y_test_set)$byClass["Neg Pred Value"]))

# and a neural network model

train_nnet <- train(training_set, y_training_set,
                        method = "nnet")
```

```r
nnet_preds <- predict(train_nnet, test_set)
mean(nnet_preds == y_test_set)
wine_results <- bind_rows(wine_results,
                          data_frame(method="neural network",
                                     accuracy =
                                       confusionMatrix(nnet_preds,
                                                       y_test_set)$overall["Accuracy"],
                                     sensitivity =
                                       confusionMatrix(nnet_preds,
                                                       y_test_set)$byClass["Sensitivity"],
                                     specificity =
                                       confusionMatrix(nnet_preds,
                                                       y_test_set)$byClass["Specificity"],
                                     balanced_acc =
                                       confusionMatrix(nnet_preds,
                                                       y_test_set)$byClass["Balanced Accuracy"],
                                     positive_pred_val =
                                       confusionMatrix(nnet_preds,
                                                       y_test_set)$byClass["Pos Pred Value"],
                                     negative_pred_val =
                                       confusionMatrix(nnet_preds,
                                                       y_test_set)$byClass["Neg Pred Value"]))

# finally include an ensemble model, which takes an average across all the models

ensemble <- cbind(glm = glm_preds == "quality", lda = lda_preds == "quality",
                  qda = qda_preds == "quality",
                  loess = loess_preds == "quality", rf = rf_preds == "quality",
                  knn = knn_preds == "quality", adaboost = adaboost_preds == "quality",
                  nnet = nnet_preds == "quality")

ensemble_preds <- ifelse(rowMeans(ensemble) > 0.5, "quality", "sub-quality") %>% factor()
mean(ensemble_preds == y_test_set)
wine_results <- bind_rows(wine_results,
                          data_frame(method="ensemble",
                                     accuracy =
                                       confusionMatrix(ensemble_preds,
                                                       y_test_set)$overall["Accuracy"],
                                     sensitivity =
                                       confusionMatrix(ensemble_preds,
                                                             y_test_set)$byClass["Sensitivity"],
                                     specificity =
                                       confusionMatrix(ensemble_preds,
                                                       y_test_set)$byClass["Specificity"],
                                     balanced_acc =
                                       confusionMatrix(ensemble_preds,
                                                       y_test_set)$byClass["Balanced Accuracy"],
                                     positive_pred_val =
                                       confusionMatrix(ensemble_preds,
                                                       y_test_set)$byClass["Pos Pred Value"],
                                     negative_pred_val =
                                       confusionMatrix(ensemble_preds,
                                                       y_test_set)$byClass["Neg Pred Value"]))
```

Results

The final table of results can be seen below. This shows that the Random Forest model delivers the best results looking across all of the models. It delivers the best results across all the measures defined above.

```r
# print the results

wine_results %>%
  knitr::kable()
```

method	accuracy	sensitivity	specificity	balanced_acc	positive_pred_val	negative_pred_val
regression on alcohol	0.8598131	0.0227273	0.9927798	0.5077535	0.3333333	0.8647799
general linear model	0.8753894	0.4090909	0.9494585	0.6792747	0.5625000	0.9100346
LDA model	0.8691589	0.4545455	0.9350181	0.6947818	0.5263158	0.9151943
QDA model	0.8411215	0.4772727	0.8989170	0.6880948	0.4285714	0.9154412
Loess	0.8816199	0.4318182	0.9530686	0.6924434	0.5937500	0.9134948
Random Forest	0.9127726	0.5000000	0.9783394	0.7391697	0.7857143	0.9249147
k nearest neighbour	0.8722741	0.1818182	0.9819495	0.5818838	0.6153846	0.8831169
adaboost	0.9034268	0.4318182	0.9783394	0.7050788	0.7600000	0.9155405
neural network	0.8660436	0.3863636	0.9422383	0.6643010	0.5151515	0.9062500
ensemble	0.8753894	0.3181818	0.9638989	0.6410404	0.5833333	0.8989899

The Random Forest model gets the correct results in 91% of cases in the test set. It can correctly identify quality wines in 50% of cases and correctly identify sub-quality wines in 98% of them. However, the most interesting figure here is probably the positive predicted value which is 78%. This represents that percentage of wines that are identified as quality that actually are. Whilst the approach discussed here would help to reduce the number of wines that need to be screened, the fact that around 1 in 5 wines identified as quality are not, would mean that human testers would probably still be required before wines could be sent to customers.

Conclusion

The Random Forest model combined with the chemical analysis should allow BWITW to quickly screen a wide number of wines to identify which of these are highly likely (around a 4 in 5 chance) to be quality and meet their requirements. However, they do need to be careful as some of those that are expected to be quality via this approach will not be and they need to assess the risk of sending these to their customers, probably combining an initial chemical screening/data science approach with human testers.

Possibilities for further analysis

The following are possibilities for future analysis:

- The parameters of the model could be changed, so that positive predicted value is ranked more highly as it is tuned. This would mean that less quality wines overall would be identified, but if the sensitivity increases then it would mean that it was less likely that a non-quality wine was identified as quality. If the positive predicted value were to be increased enough then it may be possible to identify wines purely via the chemical analysis with the Random Forest model and dispatch these directly to customers. This could enable significant additional cost savings and possibly remove the need for having any testers.
- The use of clustering approaches (either hierarchical clustering or k-means clustering) and then using some functionality from the "recommenderlab" package to create a model for predicting.