Compulsory Exercise 2: Wine prediction

Maximilian Rønseth Jasper Steinberg

14 April, 2024

Abstract

This project aims to analyse a dataset containing information about the quality of a specific type of red wine, Vinho Verde, through statistical learning methods. Our dataset consists of 1599 samples and 11 covariates, related to chemical and subjective criteria. Our goal is to inferand analyse which covariates are important for determining wine quality and to analyse links between them. The analysis is a classification one, where wine quality is interpreted categorically. The dataset comes from Kaggle and originates from a 2009 study modeling wine preferences. The methods we have employed include random forest classification and logistic regression. Results are evaluated based on metrics such as miss-classification rate, sensitivity, precision, and F1-score. Our findings provide insights into the relationship between various covariates and wine quality; we find that Random Forest models perform well with a misclassification rate of around 10%.

Introduction: Scope and purpose of your project

In this analysis, we will be attempting to perform data/statistical analysis on a dataset, that contains information about the quality of a specific type of red wine; more specifically, it contains 1599 samples of a Portuguese red wine, called Vinho Verde, as well as 11 physiochemical and sensory covariates. When we write quality of the wine sample, we refer to a subjective measure of how good the wine is perceived to be. The goal of this report, will be to do inference, and to establish which covariates are essential for deciding upon a good wine, and which ones are correlated. We will do this via a classification route, i.e. a wine will be good, if it's above some certain threshold.

Our dataset is extracted from Kaggle (https://www.kaggle.com/datasets/uciml/red-wine-quality-cortez-et-al-2009/data), and originates in a study done in 2009, attempting to model wine preferences.

The scope of this analysis is two-fold, that is to classify which covariates are relevant for deciding which qualities in a wine are related to the percieved quality (where quality is interpreted categorically), as well as a prediction task, where we try to predict whether a wine is good or bad.

Pre-processing

We begin by setting a seed, for easy replication.

```
set.seed(1)
```

We install packages for visualisation, as well as defining our principal dataset. We also rename the covariates to not have spaces, i.e. volatile acidity becomes volatile_acidity, to avoid possible future errors in R.

```
wine_data <- read_csv("C:/Users/maxim/Downloads/winequality-red.csv")
```

Since we are doing classification, we decide that a wine will be considered "good" (which we denote by 1), if its quality is greater than or equal to 7; otherwise, it will be classified as "bad" (which we denote by 0). We choose 7 as our cut-off, both based on intuition (a 7/10 wine is a "good" wine), as well as our histogram, that is shown below. We see that, with this paradigm, we have 217 good wines, and 1382 bad ones.

```
wine_data$binary_quality <- ifelse(wine_data$quality >= 7, 1, 0)
```

Next, we split our data into a training set, on which our model(s) will be fitted, and a test set, on which our model(s) will be tested.

```
split <- function(data) {
  good_wine <- filter(data, wine_data$binary_quality == 1)
  bad_wine <- filter(data, wine_data$binary_quality == 0)

  sample <- sample(c(TRUE, FALSE), nrow(good_wine), replace = TRUE, prob = c(0.7,0.3))
  train_good_wine <- good_wine[sample, ]
  test_good_wine <- good_wine[!sample, ]

  sample <- sample(c(TRUE, FALSE), nrow(bad_wine), replace = TRUE, prob = c(0.7,0.3))
  train_bad_wine <- bad_wine[sample, ]
  test_bad_wine <- bad_wine[!sample, ]

  train <- bind_rows(train_good_wine, train_bad_wine)
  test <- bind_rows(test_good_wine, test_bad_wine)

  output <- list(train, test)
  names(output) <- c("train", "test")

  return(output)
}

split_data <- split(wine_data)
df.train <- split_data$train
df.test <- split_data$test
```

Descriptive data analysis/statistics

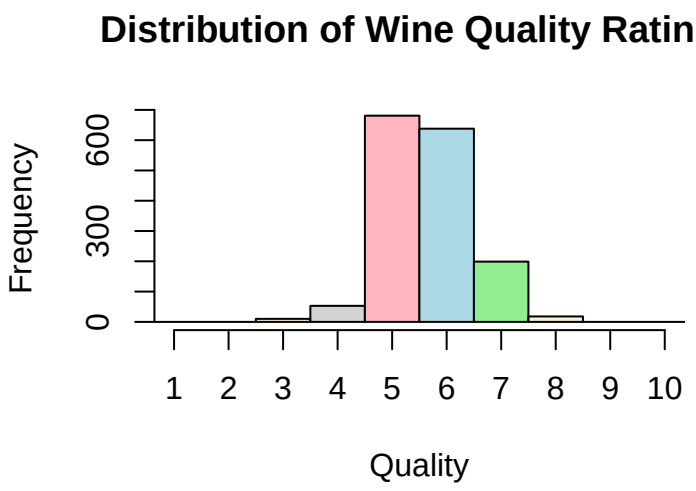
First of all, we compute a summary of our dataset, to give us some preliminary intuition about means, medians, etc. Note, that some of these covariates are noticeably harder to interpret than others: for instance, it is relatively easy to interpret the effect pH will have on the quality of the wine, as low pH values mean more sourness in the wine; on the other hand, free sulfur dioxide is harder to interpret for general consumers. We also observe that there seems to be no need to scale our features.

```
##  fixed_acidity   volatile_acidity citric_acidity  residual_sugar
##  Min.   : 4.60   Min.   :0.1200   Min.   :0.000   Min.   : 0.900
##  1st Qu.: 7.10   1st Qu.:0.3900   1st Qu.:0.090   1st Qu.: 1.900
##  Median : 7.90   Median :0.5200   Median :0.260   Median : 2.200
##  Mean   : 8.32   Mean   :0.5278   Mean   :0.271   Mean   : 2.539
##  3rd Qu.: 9.20   3rd Qu.:0.6400   3rd Qu.:0.420   3rd Qu.: 2.600
##  Max.   :15.90   Max.   :1.5800   Max.   :1.000   Max.   :15.500
##    chlorides       free_sulfur_dioxide total_sufur_dioxide    density
##  Min.   :0.01200   Min.   : 1.00       Min.   :  6.00      Min.   :0.9901
```

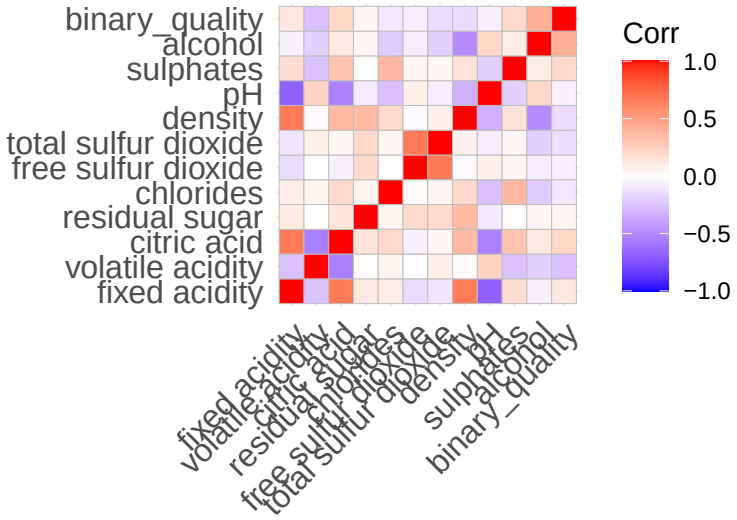
```
##  1st Qu.:0.07000   1st Qu.: 7.00       1st Qu.: 22.00      1st Qu.:0.9956
##  Median :0.07900   Median :14.00       Median : 38.00      Median :0.9968
##  Mean   :0.08747   Mean   :15.87       Mean   : 46.47      Mean   :0.9967
##  3rd Qu.:0.09000   3rd Qu.:21.00       3rd Qu.: 62.00      3rd Qu.:0.9978
##  Max.   :0.61100   Max.   :72.00       Max.   :289.00      Max.   :1.0037
##        pH          sulphates         alcohol         quality
##  Min.   :2.740   Min.   :0.3300   Min.   : 8.40   Min.   :3.000
##  1st Qu.:3.210   1st Qu.:0.5500   1st Qu.: 9.50   1st Qu.:5.000
##  Median :3.310   Median :0.6200   Median :10.20   Median :6.000
##  Mean   :3.311   Mean   :0.6581   Mean   :10.42   Mean   :5.636
##  3rd Qu.:3.400   3rd Qu.:0.7300   3rd Qu.:11.10   3rd Qu.:6.000
##  Max.   :4.010   Max.   :2.0000   Max.   :14.90   Max.   :8.000
##  binary_quality
##  Min.   :0.0000
##  1st Qu.:0.0000
##  Median :0.0000
##  Mean   :0.1357
##  3rd Qu.:0.0000
##  Max.   :1.0000
```

We see that, the average pH of the wines, lies around 3.3, and the alcohol percentage lies around 10.

We also construct a histogram for the quality of the wines, to see how the values are distributed.



We see that most wines fall somewhere in the 5/6 range. Lastly, we explicitly compute the correlation matrix.



We see that there isn't a particularly strong relationship between quality and any other covariates, although there is a slight positive correlation between alcohol and quiality, as well as a negative correlation between volatile acidity and quality. Thus, perhaps naïvely, we see that if we increase the alcohol percentage of our wine, we would expect a somewhat noticeable increase in wine quality; moreover, with an increase of volatile acidity (a covariate we are somewhat unsure how to interept), we will decrease the quality of our wine.

Lastly, we plot the densities of the two covariates with the strongest correlation (in absolute values). We see that the plots correspond well to our correlation matrix, as higher quality wines generally have a higher alcohol percentage; also, higher quality wines tend to have slightly lower levels of volatile acidity.

```
wine_data$binary_quality <- as.factor(wine_data$binary_quality)

tema <- theme(plot.title=element_text(size=24, hjust=.5, vjust=1, color="white"),
        axis.title.y=element_text(size=22, vjust=2, color="white"),
        axis.title.x=element_text(size=22, vjust=-1, color="white"),
        axis.text.x=element_text(size=22, color="white"),
        axis.text.y=element_text(size=22, color="white"),
        legend.position="None")


options(repr.plot.width=17, repr.plot.height=13)

distalc <- ggplot(data = wine_data, mapping = aes(x = alcohol, y = binary_quality)) +
                  geom_density_ridges(mapping = aes(fill = binary_quality), bandwidth=0.181,fill="papay
                  theme_solarized(light=FALSE)+
                  scale_colour_solarized('bisque')+
                  xlab("Alcohol") + ylab("Wine quality") +
                  ggtitle("Quality against alcohol") +
                  tema

distvol <- ggplot(data = wine_data, mapping = aes(x = volatile_acidity, y = binary_quality)) +
                 geom_density_ridges(mapping = aes(fill = binary_quality),bandwidth=0.181, fill="ivory3
                 theme_solarized(light=FALSE)+
                 scale_colour_solarized('bisque')+
                 xlab("Volatile acidity") + ylab("Wine quality") +
```

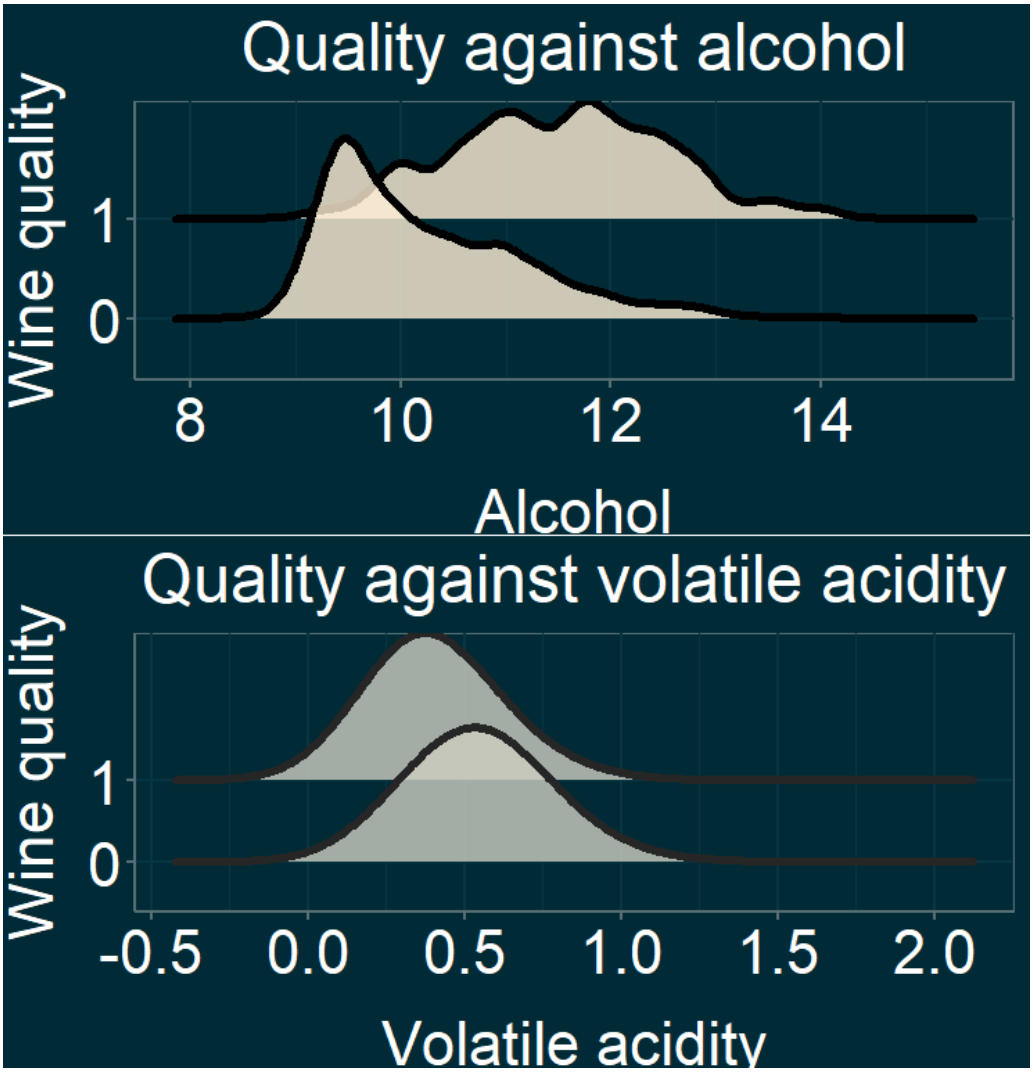
```
                  ggtitle("Quality against volatile acidity") +
                  tema


#plot_grid(distalc, distvol, nrow = 2, ncol = 1)

#for some reason this code will run just fine, so I have the plot stored as a PNG,

#but it won't knit as a pdf (but it will as HTML), so I hope you will excuse me

#pasting it in through a pdf-editor
```



Again, we see that the densities support what has been written: the curve for good wines, are more skewed towards higher levels of alcohol, and oppositely for volatile acidity.

Methods

To start of, we remove the quality variable from the training and test data, as we already have our binary version of it (as we are doing classification). We also create functions, that measure the missclassification rate, the precision and the sensitivity of our model. Recall that we define the missclassificaion rate of a model by $\frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i)$, where $I(P)$ is an indicator function that takes the value 1 if the proposition P is true, and 0 otherwise.

```
missclass <- function(model, preds, testRespons) {
  mc <- table( preds, testRespons )
  return(1 - sum( diag(mc) ) / sum( mc ) )
}

prec <- function(model, pred, testRespons) {
mc <- table(pred, testRespons)
return(mc[2, 2]/(sum(mc[2, ])))
}

sens <- function(model, pred, testRespons) {
mc <- table(pred, testRespons)
return(mc[2, 2]/(sum(mc[, 2])))
}

df.test <- subset(df.test, select = - quality)
df.test$binary_quality <- as.factor(df.test$binary_quality)
df.train <- subset(df.train, select = -quality)
df.train$binary_quality <- as.factor(df.train$binary_quality)
```

Next, we incorporate a random forest model. We define a function RF to simplify the visual aspects.

```
RF <- function(d, m, n) {
  rf.mod <- randomForest(formula = binary_quality ~ ., mtry = m, data = d, importance = T,ntree = n)
  return(rf.mod)
}
```

Our parameters mean the following: d is our dataset; m is a parameter that chooses the amount of randomly sampled covariates, and n is the amount of trees in the model. Here we have that $p := 11$ is the amount of predictors. We choose our mtry to be $m \in \{p, \sqrt{p}\}$; we also choose $n \in \{10, 50, 100\}$; this means that we have a total of $3 \cdot 2 = 6$ different "scenarios". Here, notice that $\lfloor \sqrt{p} \rfloor = 3$. We also try and find an optimal value for m, and we find, given our choice of parameters, that $\lfloor \sqrt{p} \rfloor = 3$, is a solid choice. As our ultimate goal is to buy wine at the vinmonopolet, we want to avoid classifying a good wine as a bad wine, as this will be a waste of our monetary resources. Therefore, we want the classification of good wines (1) to be as accurate as possible.

```
optm <- tuneRF(df.train, df.train$binary_quality, stepFactor = 1.2, improve = 0.01, trace = T, plot = F
```

```
## mtry = 3  OOB error = 0%
## Searching left ...
## Searching right ...
```

```
p <- 11

rf.mod1.sqrt <- RF(df.train, floor(sqrt(p)), 50)
predict11 <- predict(rf.mod1.sqrt, newdata = df.test)
error11 <- missclass(rf.mod1.sqrt, predict11, df.test$binary_quality)
prec11 <- prec(rf.mod1.sqrt, predict11, df.test$binary_quality)
sens11 <- sens(rf.mod1.sqrt, predict11, df.test$binary_quality)
mu11 <- mean(predict11 == df.test$binary_quality)
F1_1 <- 2 * (sens11*prec11)/(sens11+prec11)



rf.mod2.sqrt <- RF(df.train, floor(sqrt(p)), 100)
predict12 <- predict(rf.mod2.sqrt, newdata = df.test)
error12 <- missclass(rf.mod2.sqrt, predict12, df.test$binary_quality)
prec12 <- prec(rf.mod2.sqrt, predict12, df.test$binary_quality)
sens12 <- sens(rf.mod2.sqrt, predict12, df.test$binary_quality)
mu12 <- mean(predict12 == df.test$binary_quality)
F1_2 <- 2 * (sens12*prec12)/(sens12+prec12)


rf.mod3.sqrt <- RF(df.train, floor(sqrt(p)), 10)
predict13 <- predict(rf.mod3.sqrt, newdata = df.test)
error13 <- missclass(rf.mod3.sqrt, predict13, df.test$binary_quality)
prec13 <- prec(rf.mod3.sqrt, predict13, df.test$binary_quality)
sens13 <- sens(rf.mod3.sqrt, predict13, df.test$binary_quality)
mu13 <- mean(predict13 == df.test$binary_quality)
F1_3 <- 2 * (sens13*prec13)/(sens13+prec13)
```

Next we do essentially the same but for the different value of m.

```
rf.mod1.p <- RF(df.train, p, 50)
predict21 <- predict(rf.mod1.p, newdata = df.test)
error21 <- missclass(rf.mod1.p, predict21, df.test$binary_quality)
prec21 <- prec(rf.mod1.p, predict21, df.test$binary_quality)
sens21 <- sens(rf.mod1.p, predict21, df.test$binary_quality)
mu21 <- mean(predict21 == df.test$binary_quality)
F2_1 <- 2 * (sens21*prec21)/(sens21+prec21)



rf.mod2.p <- RF(df.train, p, 100)
predict22 <- predict(rf.mod2.p, newdata = df.test)
error22 <- missclass(rf.mod2.p, predict22, df.test$binary_quality)
prec22 <- prec(rf.mod2.p, predict22, df.test$binary_quality)
sens22 <- sens(rf.mod2.p, predict22, df.test$binary_quality)
mu22 <- mean(predict22 == df.test$binary_quality)
F2_2 <- 2 * (sens22*prec22)/(sens22+prec22)


rf.mod3.p <- RF(df.train, p, 10)
predict23 <- predict(rf.mod3.p, newdata = df.test)
error23 <- missclass(rf.mod3.p, predict23, df.test$binary_quality)
```

```r
prec23 <- prec(rf.mod3.p, predict23, df.test$binary_quality)
sens23 <- sens(rf.mod3.p, predict23, df.test$binary_quality)
mu23 <- mean(predict23 == df.test$binary_quality)
F2_3 <- 2 * (sens23*prec23)/(sens23+prec23)
```

To compare against the random forest model, we have attempted to create a logistic regression model to perform the same task. We choose the cut-off to be $p > 0.5$, which is a standard choice, and seems to fit well for our data set.

```r
logmodel <- glm(df.train$binary_quality ~., family = "binomial", data = df.train)
logtest <- predict(logmodel, type ="response", newdata = df.test)
logpred <- ifelse(logtest > 0.5, 1,0)



errorlog <- missclass(logmodel,logpred,df.test$binary_quality)
senslog<- sens(logmodel,logpred,df.test$binary_quality)
preclog<-prec(logmodel,logpred,df.test$binary_quality)
mulog <- mean(logpred == df.test$binary_quality)
F1_log <- 2*(senslog*preclog)/(senslog+preclog)
```

Results and interpretation

Lastly, we plot a table to gain an overarching understanding of how well our models have performed.

```r
knitr::kable(table, "latex", booktabs = TRUE, escape = FALSE, caption = "Model evaluations for Random f
             col.names = c(
               "$m_{try}$",
               "Trees",
               "Missclassification rate",
               "Sensitivity",
               "Precision",
               "$F_1$ score"))  %>%
             kable_styling(latex_options = "HOLD_position")
```

Table 1: Model evaluations for Random forest and logistic regression

	m_{try}	Trees	Missclassification rate	Sensitivity	Precision	F_1 score
RF11	sqrtp	50	0.1068	0.4925	0.6471	0.5593
RF12	sqrtp	100	0.0945	0.5373	0.7059	0.6102
RF13	sqrtp	10	0.0945	0.5075	0.7234	0.5965
RF21	p	50	0.0924	0.6119	0.6833	0.6457
RF22	p	100	0.0945	0.597	0.678	0.6349
RF23	p	10	0.1027	0.5522	0.6491	0.5968
LOG	N/A	N/A	0.1314	0.3433	0.5349	0.4182

Summary

Our results indicate that random forest models outperform logistic regression, achieving slightly lower misclassification rates and higher F1-scores across different parameter choices. This superiority may stem from

the inherent robustness of random forest algorithms, which handle complex interactions between predictors more effectively than logistic regression. Additionally, random forest models are less prone to overfitting and can capture nonlinear relationships between covariates and wine quality more accurately. Interestingly, we see that we achiece the lowest missclassification rate, and the highest F_1-score when $(m_{try}, \#Trees) = (p, 50)$; this combination probably achieves a balance of model complexity and overfitting, according to the bias-variance trade-off. If there is a lot of non-linearity in the data, then allowing the model to be more complex can capture this relationship better than if we had a less flexible model.

Overall, our analyses suggest that random forest models are better suited for predicting wine quality in this data set, as they are more efficient at handling complex data with non linear relationships.