Predictive modelling Assignment

Ryan Greenup | 1780-5315

29 April 2018

Part I

Answer Summary

Question 2

The RSS value is 2315.3087

Question 3

The accuracy of the 2nd Model can be found thusly:

```
test.df$pred <- max.col(prob.log.df)+(3-1)
rate <- mean(test.df$pred==test.df$quality  )
```

This provides that the rate of correct prediction is 57%, hence the logistic regression model is slightly better at predicting for wine quality (although it is still not significantly far from the standard deviation of the quality, indicating that the prediction might not be much better than guessing for the average wine qaulity anyway).

Question 4

The computed decision boundary for this particular random split of testing, validation and training data is 0.64.

The model has a 95% accuracy rate so it is a very good model for the data.

Part II

Complete Report

1. Data Preperation and Cleaning

(a) Load the Training Data

Training Data can be loaded by using the command:

```
train.df <- read.csv(file = "trainingwine1.csv", header = TRUE, sep = ",")
```

Column Medians can be calculated for the set of all non-missing data with the following commands: prettty girl

```
f.acid.med  <- median(train.df$fixed.acidity,         na.rm = TRUE)
v.acid.med  <- median(train.df$volatile.acidity,      na.rm = TRUE)
c.acid.med  <- median(train.df$citric.acid,           na.rm = TRUE)
r.sugar.med <- median(train.df$residual.sugar,        na.rm = TRUE)
cl.med      <- median(train.df$chlorides,             na.rm = TRUE)
f.so2.med   <- median(train.df$free.sulfur.dioxide,   na.rm = TRUE)
t.so2.med   <- median(train.df$total.sulfur.dioxide, na.rm = TRUE)
density.med <- median(train.df$density,               na.rm = TRUE)
ph.med      <- median(train.df$pH,                    na.rm = TRUE)
so4.med     <- median(train.df$sulphates,             na.rm = TRUE)
alcohol.med <- median(train.df$alcohol,               na.rm = TRUE)
quality.med <- median(train.df$quality,               na.rm = TRUE)
```

(b) Replace the Missing Values

The missing NA values could be replaced one by one with the following format of code:

```
train.df$fixed.acidity[is.na(train.df$fixed.acidity)] <- f.acid.med
```

however this is cumbersome to do and is prone to typos, hence a `for` loop would be more appropriate, this can be acheived with the following code:

```
for (i in 1:ncol(train.df)) {

      col.vec <- train.df[,i]
      test    <- is.na(col.vec)

      col.vec[test] <- median(col.vec, na.rm = TRUE)

      train.df[,i] <- col.vec

  }
```

(c) Export Cleaned Training Data

The cleaned training data set can be exported thusly:

```
write.csv(x = train.df, file = ``trainingwine2.csv'', row.names = FALSE)
```

(d) Clean Test Data use training means

First it is necessary to import the testing data, which can be done with the following code:

```
test.df <- read.csv(file = "testingwine1.csv", header = TRUE, sep = ",")
```

Next replace the NAs with a similar `for` loop as before, except now:

Use the **training** medians in the **testing** data frame

Nest the `for` loop within a conditional `if` test such that the medians will be replaced if and only if the column order of the dataframes are the same

As before, test for the presence of `NAs` within the data frame afterwards

This can be acheived thusly:

```
if( mean(names(test.df) == names(train.df)) ){

    for (i in 1:ncol(test.df)) {

        col.test.vec <- test.df[,i]
        col.train.vec <- train.df[,i]

        test <- is.na(col.test.vec)

        col.test.vec[test] <- median(col.train.vec, na.rm = TRUE)

        test.df[,i] <- col.test.vec

    }

    #Test if there are any NAs left:

    if(mean(is.na(train.df)) == 0){
        print("No NAs in Training Data, all good")
    } else{
        print("There are NA values in the Training Data Frame")
    }
} else{
        print("The Data Sets aren't identical, don't use a loop")
}

#Write the CSV file

write.csv(x = test.df, file = "testingwine2.csv", row.names = FALSE)
```

(e) Compute Testing Means

The column mean values can be computed again with a for loop:

```
# Q1 (e) - Compute Column Means for Cleaned Testing Data --------------------

  ##We can do this all in a for loop
    ###Create an empty vector with pre-assigned length
      meanvec <- vector(length = (ncol(test.df)-1))
    ###Create the loop
      for(i in 1:(ncol(test.df)-1)){
        meanvec[i] <- c(mean(test.df[,i]))
      }
```

Export Means as a Text file

The mean values can be exported as a text file in various ways, rounding to 3 significant figures:

```
##Create the Data Frame
  test.means.df <- data.frame(Predictor  = names(test.df)[1:(ncol(test.df)-1)],
                              Mean.Value = meanvec)

##Print the Data Frame
  test.means.df$Mean.Value <- signif(test.means.df$Mean.Value, digits = 3)
  print(test.means.df)

##Export the Data Frame
  write.csv(x = test.means.df, file = "testingdatameans.txt", row.names = FALSE)
  write.table(test.means.df, file = "testingdatameans.txt", sep="\t", row.names = FALSE)
  capture.output(print(test.means.df, print.gap=3), file="testingdatameans.txt")
```

Which outputs the following table:

```
                  Predictor   Mean.Value
1           fixed.acidity        6.4300
2        volatile.acidity        0.2870
3             citric.acid        0.3070
4          residual.sugar        6.4400
5               chlorides        0.0465
6     free.sulfur.dioxide       34.0000
7    total.sulfur.dioxide      129.0000
8                 density        0.9930
9                      pH        3.1700
10              sulphates        0.4940
11                alcohol       10.7000
```

2. Building a Linear Regression Model

(a) Create a Model

Using the **training** data a multiple linear regression model may be formed for the data thusly:

```
# Q2 (a) - Create the Linear Regression -----------------------------------

mod.lm <- lm(quality ~ fixed.acidity + volatile.acidity + citric.acid +
residual.sugar + chlorides + free.sulfur.dioxide +
total.sulfur.dioxide + density + pH + sulphates + alcohol,
data=train.df)

RSS     <- sum(resid(mod.lm)^2)
print(RSS)

[1] 2315.309
```

The RSS can be calculated by using the formula $RSS = \sum_i^n [(y_i - \hat{y}_i)^2]$

```
RSS     <- sum(resid(mod.lm)^2)
```

(b) Find RMSE

Find the values predicted by the Model $\hat{y}$

In order to measure the error between the *training model* and the *testing data* it will be necessary to find the predicted values for the wine quality that correspond to the testing data using the training model:

```
# Q2 (b) - Predict Wine Quality on Testing Data ----------------------------

##Import the testing data
test.df <- read.csv(file = "testingwine2.csv", header = TRUE, sep = ",")

## Prediction Value
### Create the new Prediction data
newdata <- test.df[,!(colnames(test.df)=="quality")]
### Create the vector of predicted values
y.hat <- predict(mod.lm, newdata)

### Inspect values
head(data.frame("X1" = test.df$fixed.acidity,
"X2"= test.df$volatile.acidity,
"Observed Quality" = test.df$quality,
"Predicted Quality" = round(y.hat)), 6)
```

Which provides the predicted quality values as:

```
  X1   X2 Observed.Quality Predicted.Quality
1 6.4 0.35               7                 6
2 6.6 0.31               6                 7
3 7.4 0.25               6                 6
4 7.3 0.36               5                 6
5 6.7 0.31               7                 7
6 8.6 0.31               5                 5
```

These appear reasonable so it can be concluded that the predict function worked correctly.

Find the RMSE

The *Root Mean Square Error* ($RMSE$) can be calculated using the following formula:

$$Loss = (y_i - \hat{y}_i)^2 \tag{1}$$

$$T.Error = \frac{1}{n} \cdot \sum_{i}^{n} [(y_i - \hat{y}_i)^2] \tag{2}$$

$$= \text{mean}(Loss) \tag{3}$$

$$RMSE = \sqrt{T.Error} \tag{4}$$

$$= \sqrt{\text{mean}(Loss)} \tag{5}$$

$$= \sqrt{\frac{1}{n} \cdot \sum_{i}^{n} [(y_i - \hat{y}_i)^2]} \tag{6}$$

by using (1), (3) and (4), the RMSE can be calculated thusly:

```
# Q2 (b) - Predict Wine Quality ------------------------------------------
## Calculate the RMSE (USE Testing Data)




## Calculate the RMSE
y             <- test.df$quality

L             <- (quality-y.hat)^2
TrainingError <- mean(L)
RMSE          <- sqrt(TrainingError)

## Print RMSE to the screen
print(RMSE)
```

Which provides the $RMSE$ value as 0.71:

```
>  print(RMSE)
[1]  0.7080614
```

(c) Find the rate of correct prediction

Correct Rounded Quality Value

The proportion of correct predictions can be found thusly:

```
# Q2 (c) - Correct Predictions -------------------------------------------

## Round the predicted qualities to integers
quality.lm <- round(y.hat)

correct.rate <- mean(quality.lm == quality)
print(paste("The model is correct at a rate of",
signif(correct.rate*100, 3),"%"))
```

which provides:

```
[1] "The model is correct at a rate of 53.9 %"
```

Confidence Interval

Although, for the sake of further consideration, the model can be found to be within a 95% confidence interval of the quality value 97% of the time:

```
#Considering the standard Deviation of quality
correct.rate2 <- mean(abs(
y.hat-test.df$quality)
<
2*sd(c(test.df$quality)))
print(paste(round(correct.rate2*100), "%"))

[1] "97 %"
```

(d) Assess the Model

Random Guess of Wine Quality

In order to make a random guess of wine quality, it is somewhat necessary to consider what is meant by random, although nothing is known about the wine quality from the training data it is known that the minimum observed quality is 3 and the maximum is 8, it can also be deduced that the quality follows a particular distribution $\sim \mathcal{N}(5.8, 0.9)$

Uniform A random guess, without any previous knowledge whatsoever at all, would have to range from a quality of 0 up to a maximum quality of 10. Probability is defined as the ratio between the number of events and the number of possible outcomes hence the probability of a correct random guess would be $\frac{1}{10}$:.

$$\begin{aligned} prob &= \frac{\#\text{ of events}}{\#\text{ of possible outcomes}} \\ &= \frac{1}{10} \end{aligned} \tag{7}$$

If it was assumed from observing training data however, that the minimum value was 3, and the maximum value was 8 then the probability, if we took a uniform choice within that interval, would be $\frac{1}{6}$:

$$\begin{aligned} prob &= \frac{\#\text{ of events}}{\#\text{ of possible outcomes}} \\ &= \frac{1}{\max(\text{quality}) - \min(\text{quality})} \\ &= \frac{1}{6} \end{aligned} \tag{8}$$

Normal However, if the training data had already been assumed, any future random guesses would be made randomly but, it would be more sensible to randomly guess them against a normal distribution, the probability of correctly guessing the quality can be measured using a monte carlo test and found to be about 33%:

```
nmcarlo     <- 100
mcarlomeans <- vector(length = nmcarlo)

for (i in 1:nmcarlo) {
r.norm.sample <- round(rnorm(n = length(test.df$quality),
mean = mean(train.df$quality),
sd = sd(train.df$quality)))
r.norm.sample <- round(r.norm.sample)

mcarlomeans[i] <- mean(r.norm.sample == test.df$quality)
r.guess.3 <- mean(mcarlomeans)
}
print(paste(signif(r.guess.3*100,2), "%"))

[1] "33 %"
```

Comparing Model Performance to Chance

```
# Q2 (d) - Model Assessment ------------------------------------------------

if(correct.rate > max(r.guess.1, r.guess.2, r.guess.3)){
print("Our Model is better at predicting wine quality than a Random Guess")
} else{
print("Our Model is no better than a random guess of wine quality")
}
```

Which provides:

```
[1] "Our Model is better at predicting wine quality than a Random Guess"
```

Hence, it can be concluded that the model, as created from training data, correctly predicts against seperate testing data more often than would be expected to occur by chance, even if random predictions were made with the benefit of having known the general distribution of quality ratings before hand.

This is good evidence for the potential utility of this model.

3. Building a Logistic Regression Model

Q3 (b) - Create Quality Indicators for all Qualities

Whether or not a wine exhibits a quality level of 3 tested for thusly:

```
Q3 (a) - Create the Quality Indicator ------------------------------------

quality3 <- as.factor(as.numeric(train.df$quality==3))
```

Q3 (b) Create Quality Indicators for all Qualities

Although the qualities could be solved as above, one-by-one, for large datasets this isn't always practical, hence we will favour a loop:

```r
# Q3 ( b) - Create Quality Indicators for all qualities --------------------

upper <- max(train.df$quality)
lower <- min(train.df$quality)

train.log.df <- as.data.frame(matrix(data = NA, nrow = nrow(train.df),
ncol = (upper-lower+1)))
colnames(train.log.df) <- c("qis3", "qis4", "qis5", "qis6", "qis7", "qis8", "qis9")

train.log.list <- list()
for (i in lower:upper) {

train.log.list[[i]] <- as.factor(as.numeric(train.df$quality==i))

}

summary(train.log.list)
head(train.log.list[[6]])
```

```
Length  Class   Mode
[1,]     0    -none- NULL
[2,]     0    -none- NULL
[3,] 3918    factor numeric
[4,] 3918    factor numeric
[5,] 3918    factor numeric
[6,] 3918    factor numeric
[7,] 3918    factor numeric
[8,] 3918    factor numeric
[9,] 3918    factor numeric
```

Q3 (c) Implement One-Vs-All Classification

A seperate model can be created for each quality level to model the probability of a wine exhibiting that quality level given the predictor values, again, to avoid typing mistakes, this is done via a loop:

```
##Train Multiple Models

train.mod.list <- list()

for (i in lower:upper) {

train.df$pred <- train.log.list[[i]]
mod.log <- glm(pred ~ fixed.acidity + volatile.acidity + citric.acid +
residual.sugar + chlorides + free.sulfur.dioxide +
total.sulfur.dioxide + density + pH + sulphates + alcohol,
data=train.df, family = "binomial")

train.mod.list[[i]] <- mod.log

}

##Predict Quality Probabilities

prob.log.list <- list()

newdata <- test.df[,!(colnames(test.df)=="quality")]

for (i in lower:upper) {

prob.log.list[[i]] <- signif(predict(train.mod.list[[i]], newdata, type = 'response'), 3)


}
```

Now we can use the model created with the *Testing* data to predict probabilities for the *Training* data to exhibit quality classifications:

```
##Predict Quality Probabilities

prob.log.list <- list()

newdata <- test.df[,!(colnames(test.df)=="quality")]

for (i in lower:upper) {

prob.log.list[[i]] <- signif(predict(train.mod.list[[i]], newdata, type = 'response'), 3)


}
```

These probability predictions are best expressed in a data frame:

```
prob.log.df <- as.data.frame(matrix(data = NA, nrow = nrow(test.df),
ncol = (upper-lower+1)))
colnames(prob.log.df) <- c("qis3", "qis4", "qis5", "qis6", "qis7", "qis8", "qis9")

for (i in lower:upper) {

prob.log.df[,i-(3-1)] <- prob.log.list[[i]]

}

head(prob.log.df)

qis3     qis4    qis5   qis6    qis7     qis8      qis9
1 0.000564 0.02430 0.0404 0.522 0.2020 0.04900 6.78e-04
2 0.001670 0.00789 0.0476 0.410 0.4690 0.13300 8.85e-05
3 0.002740 0.02080 0.1620 0.437 0.2020 0.02700 1.84e-06
4 0.007080 0.00922 0.2680 0.387 0.1080 0.02690 2.36e-06
5 0.002430 0.00801 0.0568 0.389 0.4540 0.15400 1.95e-04
6 0.007880 0.13300 0.4830 0.359 0.0609 0.00607 2.24e-06
```

This data frame can be exported as a CSV file as such:

```
write.csv(x = prob.log.df, file = "results.csv")
```

Q3 (d) - Model Accuracy

The accuracy of the model can be assessed by first predicting the most probable wine classification:

```
test.df$pred <- max.col(prob.log.df)+(3-1)
head(test.df)

> head(test.df)
fixed.acidity volatile.acidity citric.acid residual.sugar chlorides
1           6.4             0.35        0.28            1.6     0.037
2           6.6             0.31        0.28            1.4     0.035
3           7.4             0.25        0.37            2.6     0.050
4           7.3             0.36        0.34           14.8     0.057
5           6.7             0.31        0.30            2.4     0.038
6           8.6             0.31        0.30            0.9     0.045
free.sulfur.dioxide total.sulfur.dioxide density   pH sulphates alcohol
1                  31                   113 0.99390 3.12      0.40    14.2
2                  28                   107 0.98836 3.00      0.40    13.2
3                  24                   132 0.99138 3.04      0.53    11.2
4                  46                   173 0.99751 3.14      0.57    10.2
5                  30                    83 0.98867 3.09      0.36    12.8
6                  16                   109 0.99249 2.95      0.39    10.1
quality pred
1       7    6
2       6    7
3       6    6
4       5    6
5       7    7
6       5    5
```

Then by comparing the two values the prediction accuracy can be determined:

```
rate <- mean(test.df$pred==test.df$quality  )
paste("the model correctly predicts the quality of wine at a rate of", round(rate*100, 2),
```

Which hence provides that the accuracy of the logistic regression model is 57% which is slightly more accurace than the linear regression

4. Decision Boundary

Q4 (a) - Import the Wine 3 Dataset

The wine3 dataset can be imported thusly:

```
# Q4 (a) - Import the wine3 dataset ----------------------------------------

wine3.df <- read.csv(file = "wine3.csv", header = TRUE, sep = ",")


# Set the Seed -------------------------------------------------------------
#Such that the code is consistent we will set the seed.
set.seed(seed = 314)
```

Q4 (b) - Train a Logistic Regression

Create Training/Validation/Test Sets

Before a logistic Regression can be implemented, the data needs to be seperated into training, validation and testing sets:

```
 n         <- nrow(wine3.df)

n.train   <- round(n * 0.5)
n.val     <- round(n * 0.25)
n.test    <- round(n * 0.25)

idvals    <- sample(n)

train.df <- wine3.df[idvals[1:n.train],]
val.df   <- wine3.df[idvals[(n.train + 1):(n.train + n.val)],]
test.df  <- wine3.df[idvals[(n.train + n.val + 1):(n)],]

train.df %>% head()
val.df   %>% head()
test.df  %>% head()
```

Once the data has been seperated the Training set can be used to fit a model:

```
wine.logm <- glm(red ~ free.sulfur.dioxide + alcohol + volatile.acidity, family = "binomial"
    train.df)
```

Q4 (c) - Use an ROC curve to select the decision boundary

Predict Validation Data using Training Model

Using the model created with the training data, it is necessary to predict values for the validation data so it can be used to analyse the ROC curve:

```
  #Predict the Probabilites from Validation data
newdata <- val.df[,!(colnames(val.df)=="red")]
wine.pred <- predict(wine.logm, newdata, type = 'response')
```

Calculate the ROC Values

The ROC plot requires the *True Positive Rate* and*False Positive Rate*, relative to *decision boundary* values:

```
# Sensitivity function for Variable Threshold
sensitivity <- function(pred, observation, threshold) {
pred.stat <- ifelse(pred < threshold, 0, 1)
# Calculate sensitivity = (# true positives) / (# condition positive)
TruePos.n <- sum((pred.stat == 1) * (observation == 1)) # Number of corecctly predicted posi
    responses
ObsPos.n  <- sum((observation == 1)) # Number of Observed positive responses.
sensitivity.val <- TruePos.n / ObsPos.n
#Return the Sensitivity Value
return(sensitivity.val)
}

# function to compute specificity for arbitrary threshold
specificity <- function(pred, observation, threshold) {
pred.stat <- ifelse(pred < threshold, 0, 1)
# Calculate specificity is (# true negatives) / (# condition negative)
TrueNeg.n <- sum((pred.stat == 0) * (observation == 0)) # Number of Correctly predicted Nega
    Responses
ObsNeg.n  <- sum(observation == 0) # Number of observed Negative responses
specificity.val <- TrueNeg.n/ ObsNeg.n
#Return the Specificity Value
return(specificity.val)
}

# compute ROC values
steps <- 10^3
thresholds <- seq(0, 1, length.out=steps)
roc <- data.frame(thresh = NA, fpr = NA, tpr = NA)
for (i in 1:steps) {
roc[i, "thresh"] <- thresholds[i]
roc[i, "fpr"] <- 1 - specificity(wine.pred, val.df$red, thresholds[i])
roc[i, "tpr"] <- sensitivity(wine.pred, val.df$red, thresholds[i])
}

head(roc)
```

The ROC curve can hence be plotted:

```
# plot ROC curve
plot(roc$fpr, roc$tpr, type='l', xlab="False Positive Rate", ylab="True Positive Rate", main
    ")
```

Select Best Decision Boundary

The best decision Boundary, is the boundary that minimises the False Positive Rate and Maximises the True positive Rate, it can be found thusly:

```
# Select the Threshold that maximises the True Positive Rate and minimises the False Positiv
thresh.val <- which.max(roc$tpr-roc$fpr)
points(roc$fpr[thresh.val], roc$tpr[thresh.val], pch=7)
text(roc$fpr[thresh.val], roc$tpr[thresh.val], paste("Threshold = ", signif(roc$thresh[thres
     pos = 4)
abline(coef = c(0,1), lty = 3)
```

For this particular split of data, the best decision boundary is .64.

Q4 (d) - Use the Testing Data to assess the Model Accuracy

Now that a model and decision boundary have been chosen, it is possible to evaluate the data with the testing data:

```
# Q4 (d) - Use the Testing Data to assess the Model Accuracy --------------

#determine prediction probabilities
newdata   <- test.df[,!(colnames(test.df) == "red")]
test.prob <- predict(wine.logm, newdata, type = "response")

#Implement a Decision boundary (thresh.val)
test.pred <- ifelse(test.prob > roc$thresh[thresh.val] , 1, 0)

#Inspect the Observed values
test.df$red

#Determine the prediction rate
rate <- sum(test.pred == test.df$red)/length(test.df$red)

#hence,
paste("The rate of correct prediction for the model created with the training data and appli
    testing data with the optimal threshold is approximately", signif(rate, 2)* 100, "%")
```