Data61 Job Application Assignment

Eve Slavich

28 September, 2017

Contents

Introduction

This is an assignment performed as part of an application for the position of Data Scientist at CSIRO. I have written up my answers using Rmarkdown, which folds the code and outputs into a pdf report. By reading the .Rmd file you can view the backend code.

Part One: Visualise The Data

Let's first plot the response variable, wine quality. Figure 1 shows that wine quality varies from 3 to 9, with the bulk of observations (Interquartile range) being 5 or 6. It looks like it's safe to treat this response as normal, and use measures like mean squared error to evaluate models.

A plot of the predictor variables (Figure 2), shows that the red and white wines have distinct characteristics- for example red wines have higher volatile acidity, while white wines have higher residual sugar. These distinct characteristics combined with the fact that reds and whites have a similar quality distribution suggests that reds and whites are scored differently and should be modelled separately. Many of the variables have some right skew to them (e.g. chlorides and residual sugar) which suggests log transforming the variables may yield better predictors.

A pairs plot shows trends between the variables. E.g. Density and fixed acidity are quite correlated, and citric acid and fixed acidity are quite correlated.

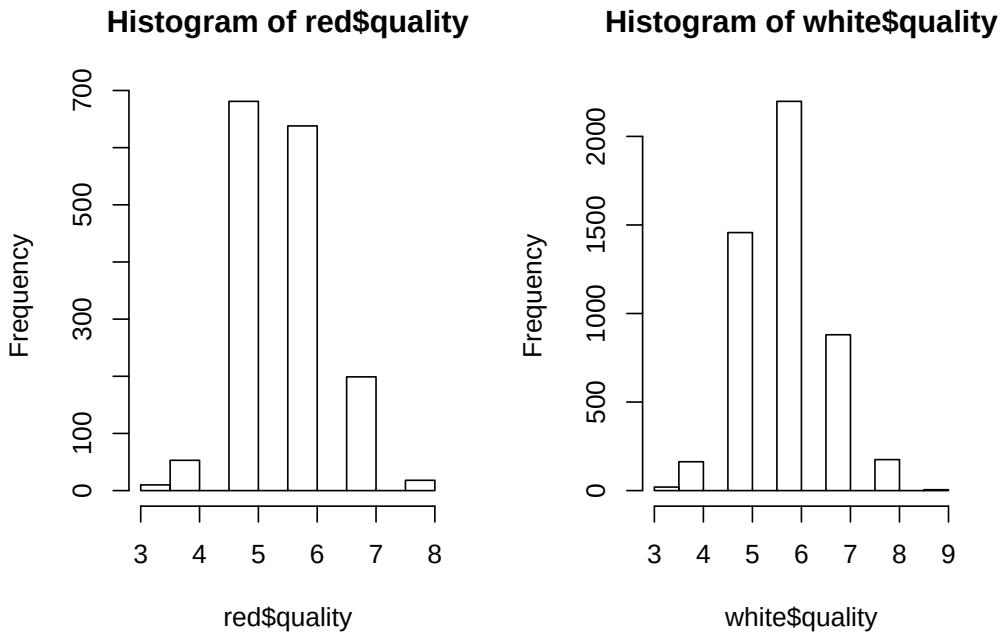


Figure 1: Histograms of the distribution of wine quality.

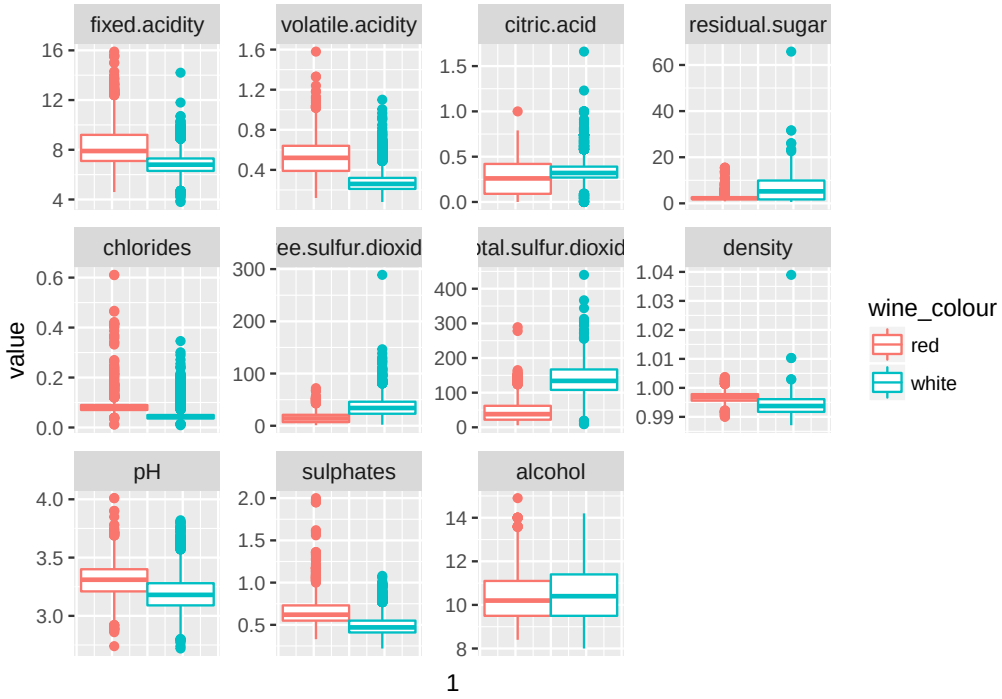
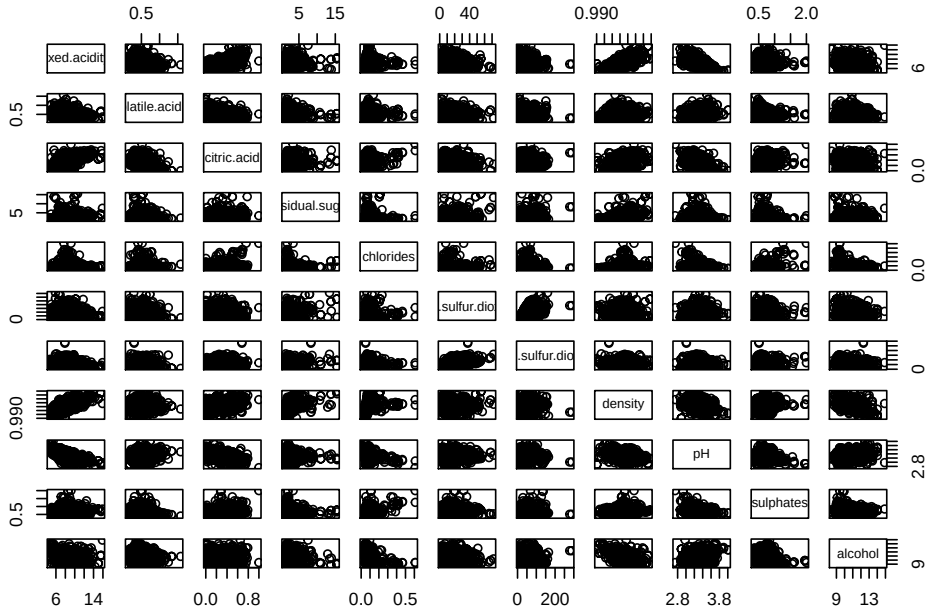
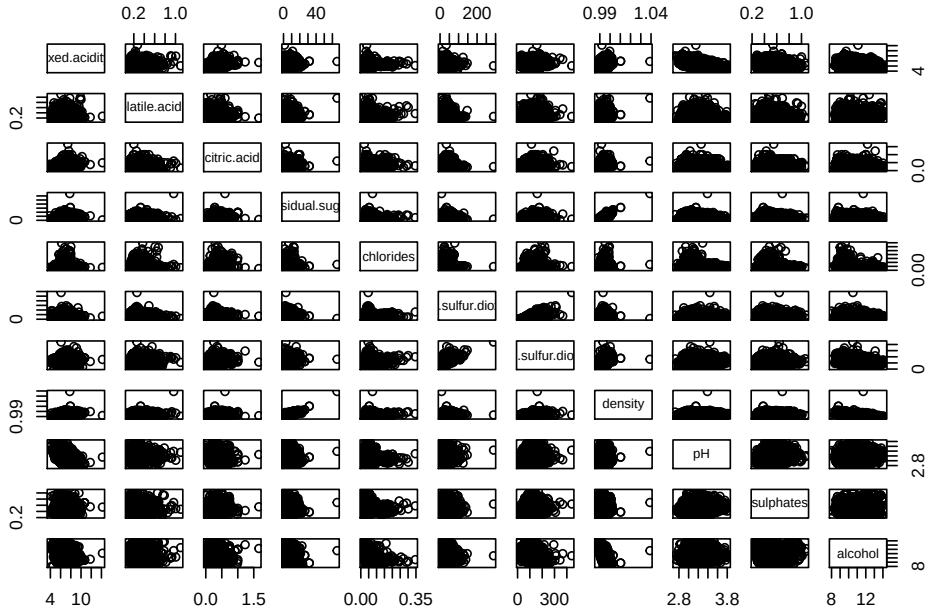


Figure 2: Boxplots of predictor variables. White and Red wines are plotted as side by side boxplots- they often inhabit separate 'zones' of the predictor variables.





Now let's look for trends between wine quality and the predictor variables (Figures 3,4). There are some trends visible (e.g. wine quality increases with alcohol, and decreases

with volatile acidity), although none are paricularly strong. Often the smoother is being swayed heavily by a couple of influential points. Cross validation should help me select the method that is less swayed by these points (I chose not to delete any outliers).

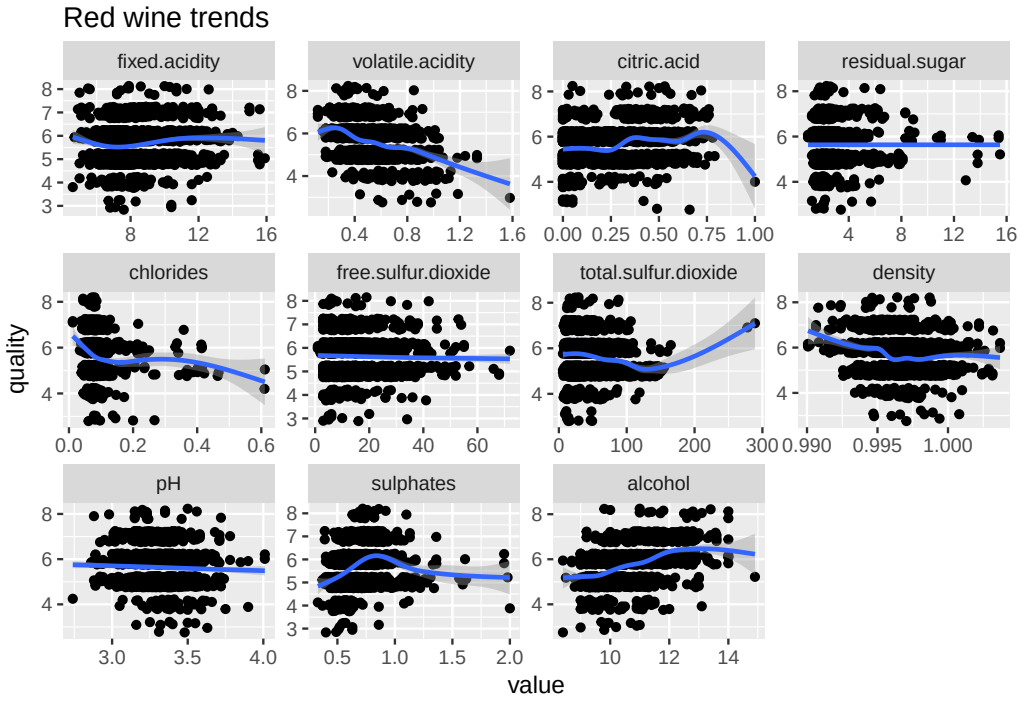


Figure 3: Trends in red wine quality against the predictor variables. Here "jittering" has been used to display the trends better (a small amount of deviation is added to the quality score to make the score less discrete). The smoother (using generalised additive modelling) displays the trend in the true (unjittered) data.

Part Two: Predict Wine Quality Scores.

As the trends in wine quality don't seem to be strong with the predictors, methods like decision trees/ random forest are likely to better encapsulate the complex interactions between the predictors that determine quality.

I decided to try three approaches.

1) Fitting a lasso regression (using glmnet for R). I added all pairwise interactions and logs of some variables to the input variables. Lasso works by minimising the penalised negative log likelihood. A penalty term λ needs to be selected. I used cross validation and a grid search to select λ. Rather than select the λ that minimises the mean squared error (λ_{min}), I selected the maximum value within one standard error of (λ_{min}). This imposes are higher penalty on complexity.

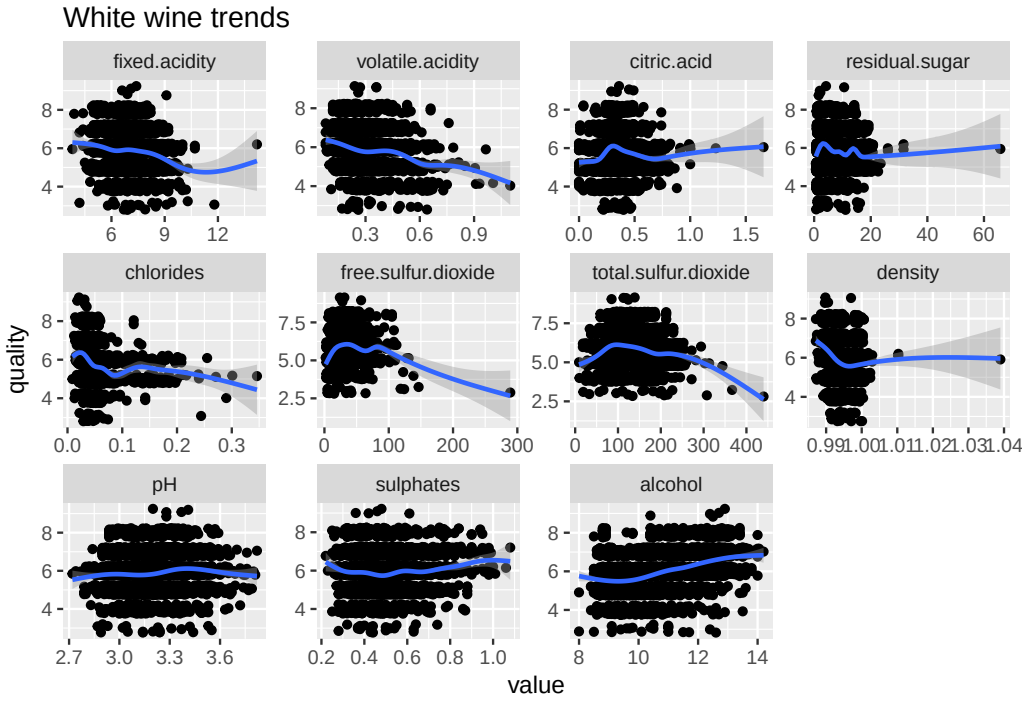


Figure 4: Trends in white wine quality against the predictor variables. Here "jittering" has been used to display the trends better (a small amount of deviation is added to the quality score to make the score less discrete). The smoother (using generalised additive modelling) displays the trend in the true (unjittered) data.

2) A random forest (using randomForest for R). I did not include the pairwise interactions (as with lasso) here as the forest should be good at discovering those itself. This implementation uses Breiman's algorithm.

3) A lasso regression additionally using as variables the rules generated from an ensemble of decision trees. The last method is similar to RuleFit (Friedman and Popescue 2005, http://statweb.stanford.edu/~jhf/ftp/RuleFit.pdf). The idea of RuleFit is to fit decision trees to generate a large number of new potential features, but then select features and fit coefficients via lasso regression. A (very) large number of the coefficients will be estimated as zero.

To compared the approaches I used K - fold cross validation with K = 3, repeated 3 times, and then averaging the resulting performance metrics. These numbers are a bit on the low side, but give a rough indication of predictive accuracy. I used the root mean squared error and mean absolute deviation as performance metrics. I evaluated the performance metrics on both the raw model predictions and the rounded model prediction (e.g. the model may predict a value of 5.3, and we round this to a score of 5). I assumed if a wine expert wanted to give a score of 5.5 they would round it to 6 and so forth.

Now lets have a look at the results, which I have combined into the tables evaluation_metrics_red and evaluation_metrics_white.

```
end_time = proc.time()
end_time- start_time
```

```
##    user  system elapsed
##  791.53    1.50  799.68
```

```
evaluation_metrics_red
```

```
##                             Rule Fit  Naive Lasso RandomForest
## RMSE                        0.6850179 0.7174064   0.5910697
## MAE                         0.5541563 0.4492391   0.4365836
## RMSE (rounded predictions) 0.7720628 0.7174064   0.6362326
## MAE (rounded predictions   0.5153221 0.4492391   0.3452158
```

```
evaluation_metrics_white
```

```
##                             Rule Fit  Naive Lasso RandomForest
## RMSE                        0.7604294 0.7867297   0.6207202
## MAE                         0.5930134 0.5178304   0.4541849
## RMSE (rounded predictions) 0.8446784 0.7867297   0.6690092
## MAE (rounded predictions   0.5824146 0.5178304   0.3757316
```

The table shows that random forest is performing best- on test data it has the lowest mean absolute error and root mean squared error. Whether or not we round the score, to make it a kind of classification of wine quality, does not seem to affect things.

Part 3: Variable Importance

Lets fit the random forest again and look at variable importance. For a particular variable x, we take the average percentage increase in mean squared error (on the test data/ out of bag data, over all trees) when x is randomly permuted. A high increase means the variable is more important. Figure 5 show the relative importance of the predictors. Alcohol and volatile acidity are among most important predictors for both red and white wine. Sulfates are important for red wines while free sulphur dioxide is amongst the most important predictors for white wine.

```
fit_red = randomForest(x=red_1[,1:11], y=red_1[,"quality"], importance=TRUE)
fit_white = randomForest(x=white_1[,1:11], y=white_1[,"quality"], importance=TRUE)
```

```
par(mfrow=c(1,2))
varImpPlot(fit_red,type=1 )
varImpPlot(fit_white,type=1 )
```

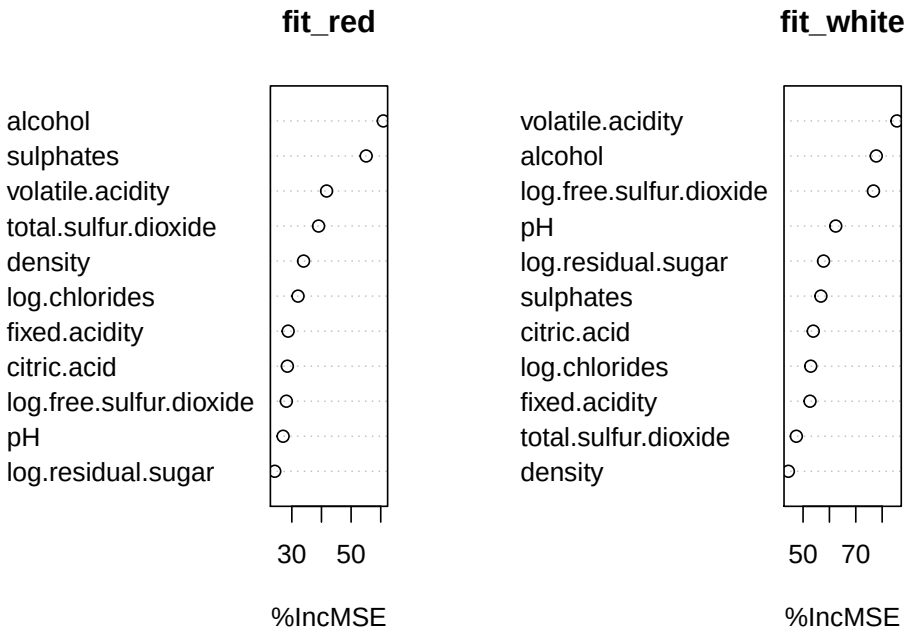


Figure 5: This plot shows the importance of the predictors for red and white wines: alcohol, sulphates and volatile acidity are the most important predictors for red wine whilst volatile acidity, free sulphur dioxide and alcohol are the most important predictors for white wine.

Wrap - Up

I fitted three different types of model. Based on what I read, I thought RuleFit, the combination of decision trees and lasso, was going to perform best. However, my implementation of it did not improve on a naive lasso implementation. Perhaps it is prone to overfitting, or perhaps I need to spend more computational effort on determining the optimal lasso penalty and the decision rules. I chose the lasso penalty with 5 - fold cross validation (20 would be better). I could have generated more decision rules by "growing" more decision trees (with more random splits of the data). Some things I thought of that could also be tried to improve the fit of the lasso include adding quadratic and cubic terms in the predictors.

One thing I did not consider, was the correlation between variables in the variable importance calculations. It's notable that the method I used selected several of the least collinear variables (according to the pairs plot). Could clusters of correlated variables be more important?

Code Appendices

The code can be examined in the file Functions.R. However I have also appended the more important functions here for convenience

```
RuleFit
```

```
## function (train, test, y_var, ...)
## {
##     Feature_matrix_red = Create_features_from_decision_forest(train,
##         K = 5, N = 2, test)
##     X_test = cbind(Feature_matrix_red$all_new_features_test,
##         test[, -y_var])
##     X_train = cbind(Feature_matrix_red$all_new_features, train[,
##         -y_var])
##     ft.lasso.red = fit_lasso(X = X_train, y = train[, y_var])
##     test_predictions = predict(ft.lasso.red, newx = X_test)
##     test_predictions
## }
## <bytecode: 0x00000000279e00c8>
```

```
random_Forest
```

```
## function (train, test, x_vars, y_var)
## {
##     fit1 = randomForest(x = train[, x_vars], y = train[, y_var],
##         xtest = test[, x_vars], ytest = test[, y_var])
##     pred = fit1$test$predicted
##     pred
## }
```

```
## <bytecode: 0x00000000268c4110>
```

```
lasso_fit_and_predict
```

```
## function (train, test, x_vars, y_var)
## {
##     mod = fit_lasso(train[, x_vars], train[, y_var])
##     pred = predict(mod, newx = test[, x_vars])
##     round(pred)
## }
## <bytecode: 0x0000000026f7db00>
```

```
Create_features_from_decision_forest
```

```
## function (dat, K, N, dat_test)
## {
##     n = nrow(dat)
##     dat = data.frame(dat)
##     all_new_features = matrix(ncol = 0, nrow = nrow(dat))
##     all_new_features_test = matrix(ncol = 0, nrow = nrow(dat_test))
##     for (j in 1:N) {
##         flds <- createFolds(1:n, k = K)
##         for (k in 1:K) {
##             train = dat[-flds[[k]], ]
##             fit1 = rpart(quality ~ ., data = train)
##             splits = fit1$splits
##             new_features = matrix(nrow = nrow(dat), ncol = nrow(splits))
##             new_features_test = matrix(nrow = nrow(dat_test),
##                 ncol = nrow(splits))
##             for (i in 1:nrow(splits)) {
##                 new_features[, i] = dat[, rownames(splits)[i]] >=
##                   splits[i, "index"]
##                 new_features_test[, i] = dat_test[, rownames(splits)[i]] >=
##                   splits[i, "index"]
##             }
##             colnames(new_features) = paste0(rownames(splits),
##                 ">=", round(splits[, "index"], 2))
##             colnames(new_features_test) = paste0(rownames(splits),
##                 ">=", round(splits[, "index"], 2))
##             all_new_features = cbind(all_new_features, new_features)
##             all_new_features_test = cbind(all_new_features_test,
##                 new_features_test)
##         }
##     }
##     list(all_new_features = all_new_features, all_new_features_test = all_new_features_test)
## }
## <bytecode: 0x0000000024e14b48>
```

```
cv.error
```

```
## function (dat, x_vars, y_var, K, N, fit_and_predict)
## {
##     n = nrow(dat)
##     sum_of_squared_diffs = matrix(nrow = N, ncol = K)
##     sum_of_abs_diffs = matrix(nrow = N, ncol = K)
##     sum_of_abs_diffs_rounded = matrix(nrow = N, ncol = K)
##     sum_of_squared_diffs_rounded = matrix(nrow = N, ncol = K)
##     for (j in 1:N) {
##         flds <- createFolds(1:n, k = K)
##         for (k in 1:K) {
##             train = dat[-flds[[k]], ]
##             test = dat[flds[[k]], ]
##             predictions = fit_and_predict(train = train, test = test,
##                 x_vars = x_vars, y_var = y_var)
##             predictions_rounded = round(predictions)
##             sum_of_squared_diffs[j, k] = sum((predictions - test[,
##                 y_var])^2)
##             sum_of_abs_diffs[j, k] = sum(abs(predictions - test[,
##                 y_var]))
##             sum_of_squared_diffs_rounded[j, k] = sum((predictions_rounded -
##                 test[, y_var])^2)
##             sum_of_abs_diffs_rounded[j, k] = sum(abs(predictions_rounded -
##                 test[, y_var]))
##         }
##     }
##     average_RMSE = mean(sqrt(apply(sum_of_squared_diffs, 1, sum)/n))
##     average_MAE = mean(apply(sum_of_abs_diffs, 1, sum)/n)
##     average_RMSE_rounded = mean(sqrt(apply(sum_of_squared_diffs_rounded,
##         1, sum)/n))
##     average_MAE_rounded = mean(apply(sum_of_abs_diffs_rounded,
##         1, sum)/n)
##     list(average_RMSE = average_RMSE, average_MAE = average_MAE,
##         average_RMSE_rounded = average_RMSE_rounded, average_MAE_rounded = average_MAE_rounde
## }
## <bytecode: 0x0000000024a07438>
```