Predicting wine quality from physicochemical properties

Bjoernar Tuftin

January 8, 2020

Introduction

In this project I analyzed part of the *Wine Quality Data Set* (Cortez et al, 2009a) from the UCI Machine Learning Repository (Dua & Graff, 2019). The complete data set contains measurement data of physicochemical properties collected during certification of a number of red and white wines by the Viticulture Commission of the Vinho Verde region of Portugal, as well as expert assessments of those wines. The dataset stems from a project involving the commission and the University of Minho, Portugal. That project was a case study of a machine learning approach to model wine preferences based on measured physicochemical properties with results published as *'Modeling wine preferences by data mining from physicochemical properties'* and available online as a preprint(Cortez et al, 2009b).

They used three approaches, multiple regression (MR), neural network (NN) and support vector machine (SVM) to select features and train regression models and compared the resulting predictions using Regression Error Characteristics (REC) with varying tolerances, as well as Mean Absolute Deviation. Exploring the data and comparing naive approaches with the results they report, both through comparable measures and visualizations, I found their results unimpressive and wanted to see if improvements could be made if one took into consideration the high difference in prevalence and the fact that a quality assessment is ordinal rather than numeric data.

I chose to compare my own results running a basic regression and a comparable SVM with the results using the `ordinalForest` function from the package of the same name. All modeling and training was run using functions in the `caret` package.

The Ordinal Forest approach compensates for the ordinal nature of the data by creating multiple different ways to score the classes and choosing between them based on the accuracy of the result and it has a parameter for choosing between overall accuracy, accuracy by class, etc., which by default seeks to maximize the accuracy per class. Using regression approaches rather than straight classification does include the order of the ordinal data in the analysis, but under the assumption that a wine of quality 4 is somehow half as good as one in quality 8 and a 6 is twice as good as a 3.

Cortez et al (p. 25, 2009b) appear to report their results as an average over their many bootstraps, but I chose to create a separate test set of 10 % for validation of the generalizability of the models and ran final validation tests on that set after completing training on the remaining data. My results are therefore not directly comparable to theirs.

I compared the first algorithms by running predictions on the training set, and looking at overall accuracy when mapping regression results to one class and when accepting "one off" errors, and per class accuracy allowing "one off" errors. The overall results were similar to those reported by Cortez et al (p. 25, 2009b).

For the Ordinal Forest algorithm though, predicting on the training set gave a near perfect fit. Looking at the mean accuracy reported for the bootstrap samples in training I still observed an improvement over that shown in predicting on the training set for the previous algorithms, but I feared overfitting and changed the number of trees in the final forest to try to mitigate that.

Based on the error characteristics I was using I couldn't be certain it was successful, but it seemed like a plausible approach and I chose a forest size that only reduced the overall accuracy on the bootstraps by an amount withing a standard deviation of the best and then moved to validating the final models by predicting on the reserved test set.

This showed that the Ordinal Forest algorithm still outperformed the other algorithms. My attempts at reducing overfitting appeared to, at best, have had only a minuscule positive effect, but, as expected based

on the results in training, they definitely represent a simplification of the model with insignificant loss of accuracy.

A copy of the .Rmd-file used to generate this report can be found at: https://github.com/btuftin/edx-cyop-wine-quality-prediction

Method and Analysis

Data aquisition and cleaning

The data required no cleaning as it was available as CSV-type file and could be readily imported directly to a data frame. In total the full dataset had 1599 observations of 13 variables, each observation being a different wine.

Data Exploration

The target variable

I first split the data 90/10 into a training set of 1437 observations and a test set with 162. The variables are 11 physicochemical measurements and a quality assessment. According to Cortez et al (pp 6-7, 2009b) the quality assessment is the median of a score given by three experts grading on a scale from 1 to 10. As we can see from a plot of the distribution of scores (fig. 1), none of the red wines in the training sample were given a median score at the extremes of the scale. A plausible reason for this could be that the expert scores were never in perfect agreement on extreme scores. There's also the fact that these are measures from a certification board, and presumably a vintner would not bother with the expense of certification for a really bad wine, but that does not explain the lack of wines in the top categories.

The qualities of 3 and 8 have so low prevalence I checked closer and found that out of the total 1599 observations, there are only 10 of quality 3 and 18 of quality 8. Since the training/test split is done by quality there was only one observation of quality 3 and two of quality 8 in the test set.

Fig. 1 also shows us that the vast majority of wines, over 80%, end up with a classification of 5 or 6, and that more than half of what remains is in classification 7. In table 2, Cortez et al. (p. 24, 2009b) report an accuracy of about 89% for red wines when counting "one offs" as hits, for all three approaches used, which does not seem very good considering predicting all 6s then gives you hits for all 5s, 6s and 7s, which would mean an accuracy of 0.9498956 on the training data.

The per class sensitivity for such an approach would of course be 0 for 3s, 4s, and 8s, so it's not a fair comparison.

The predictors

Since so much of the data is in classes 5 or 6 I decided to start by looking at the distribution for the predictors for those two classes. Fig. 2 is a box and whisker plot of the z-scores for each of the 11 predictors.

We see that there are few large differences between quality 5 and 6. Alcohol and sulphates are the only ones for which at least one of the medians fall outside the interquartile range of the other class, but there is still substantial overlap. Based on this I knew successful prediction would have to rely on quality correlating with interactions between the predictors, if any existed.

I wanted to see if the trends hinted at by comparing 5 and 6 held for the other classes, but since an equivalent plot for all the qualities would be difficult to interpret I instead made one for grouped qualities, defining 3 and 4 as "poor", 5 and 6 as "middling", and 7 and 8 as "good".

We see in figure 3 that comparing these three groupings there are more factors that could plausibly have some predictive value and that alcohol and sulphates at least partially have the same trend as we saw comparing just quality 5 and 6.

But we also see that there is a great deal of overlap even for the variables with the most potential.

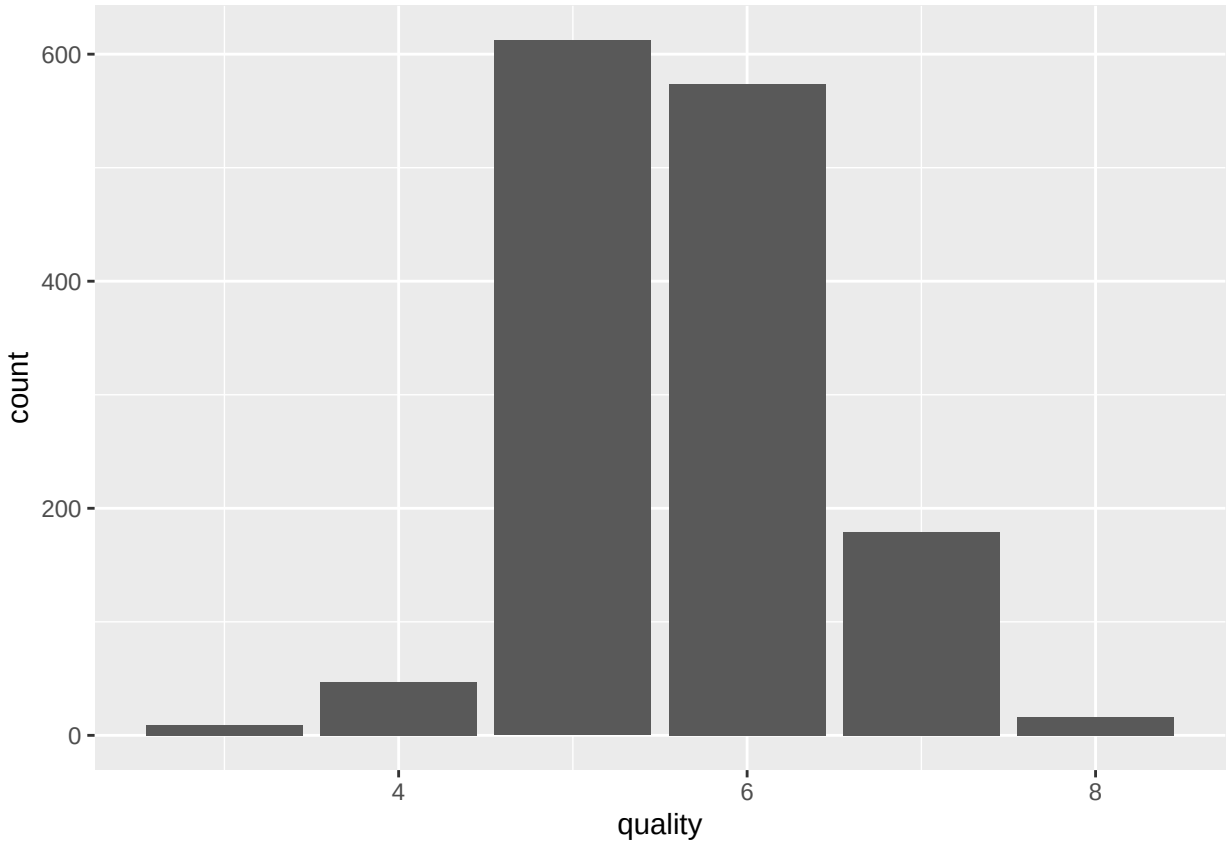

Figure 1: Distribution of quality scores in the training data

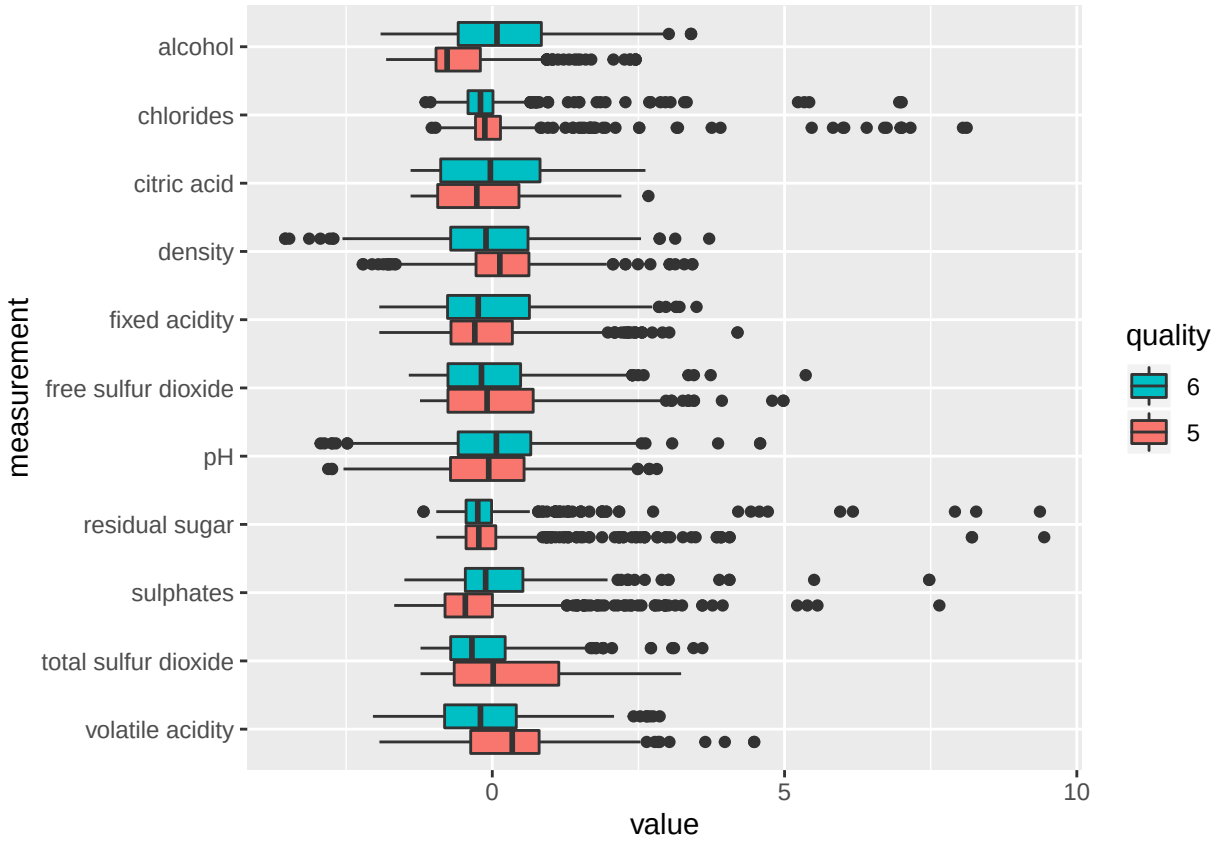

Figure 2: Variation in physicochemical measurements between qualities

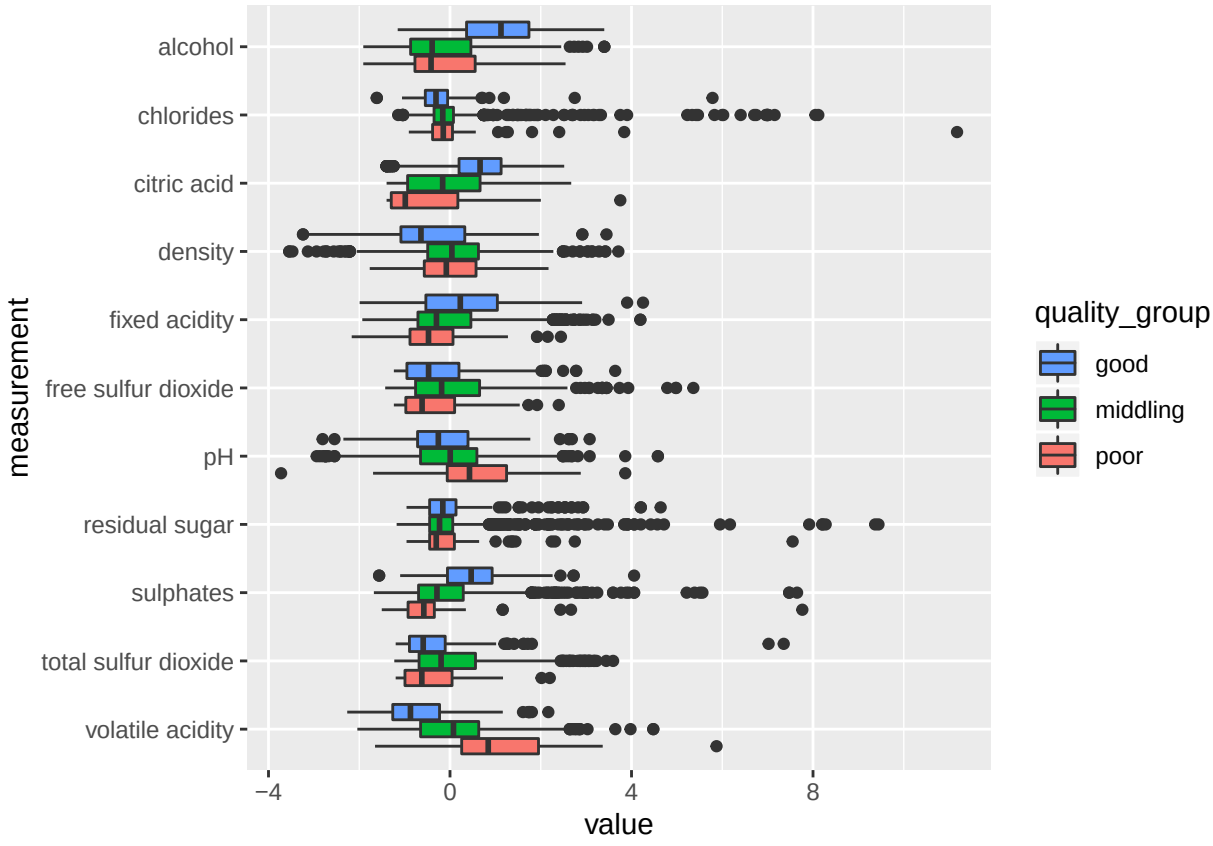

Figure 3: Variation in physicochemical measurements between quality groups

Modeling

Error Metrics

There are many error metrics one could use for training models on this data. I chose to use the default metric in caret for each algorithm for training, but to compare the results I used an Regression Error Characteristic (REC), the percentage of predictions where the observed class falls within the range defined by the prediction and a tolerance T, with tolerances of .5 (T .5) and 1 (T 1) overall. To see how the algorithms faired on the less prevalent classes I also used **T 1**grouped by the actual class of the observation.

T .5 is the same as mapping each prediction onto the closest class, **T 1** counts "one-offs" as hits, which makes sense when trying to predict ordinal data and comparing regression and classification.

Naive approaches

As figure 1 shows, quality 5 and 6, the middling wines, are much more prevalent than the others, with 7 less so but still significantly more prevalent than the others, so a decent model should at least be able to beat a prediction of 6 for all wines. The precise error metrics for using this naive approach to predict the training data are given in tables 1 and 2.

Table 1: Regression Error Characteristic (tolerances 0.5 and 1.0).

model	T=0.5	T=1.0
Naive	0.399	0.95

Table 2: Regression Error Characteristic class (tolerance 1.0)

model	3	4	5	6	7	8
Naive	0	0	1	1	1	0

If I had only used **T 1** this would have looked like a great model, but its ridiculousness is obvious from looking at the accuracy by class.

Linear Model

The next step up in complexity is a basic linear model $Y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + ... + \beta_1 1 x_{i,11}$ with a term for each predictor. Running a linear model in the training set and predicting on the same set I got the results in tables 3 and 4.

Table 3: Mean Absolute Deviation and Regression Error Characteristic (tolerances 0.5 and 1.0).

model	T=0.5	T=1.0
Naive	0.399	0.950
Linear Regression	0.603	0.891

Table 4: Regression Error Characteristic class (tolerance 1.0)

model	3	4	5	6	7	8
Naive	0	0.000	1.000	1.000	1.000	0
Linear Regression	0	0.149	0.953	0.974	0.737	0

From table 3 we see that although we predict the actual class somewhat better through linear regression, the overall accuracy when counting "one-offs" falls slightly. We see this in the per class predictions as well. Some 5s, 6s and nearly 27% of 7s are now predicted more than one class away. Making decisions about the model based on this is slightly risky, since I ran the prediction on the same data set used to train the algorithm, but the sparseness of the data in the least predicted classes make it undesirable to partition the data more than once.

Figure 4. shows the distribution of predictions per observed class. We can see that range of predictions is a lot narrower than the range of observations.

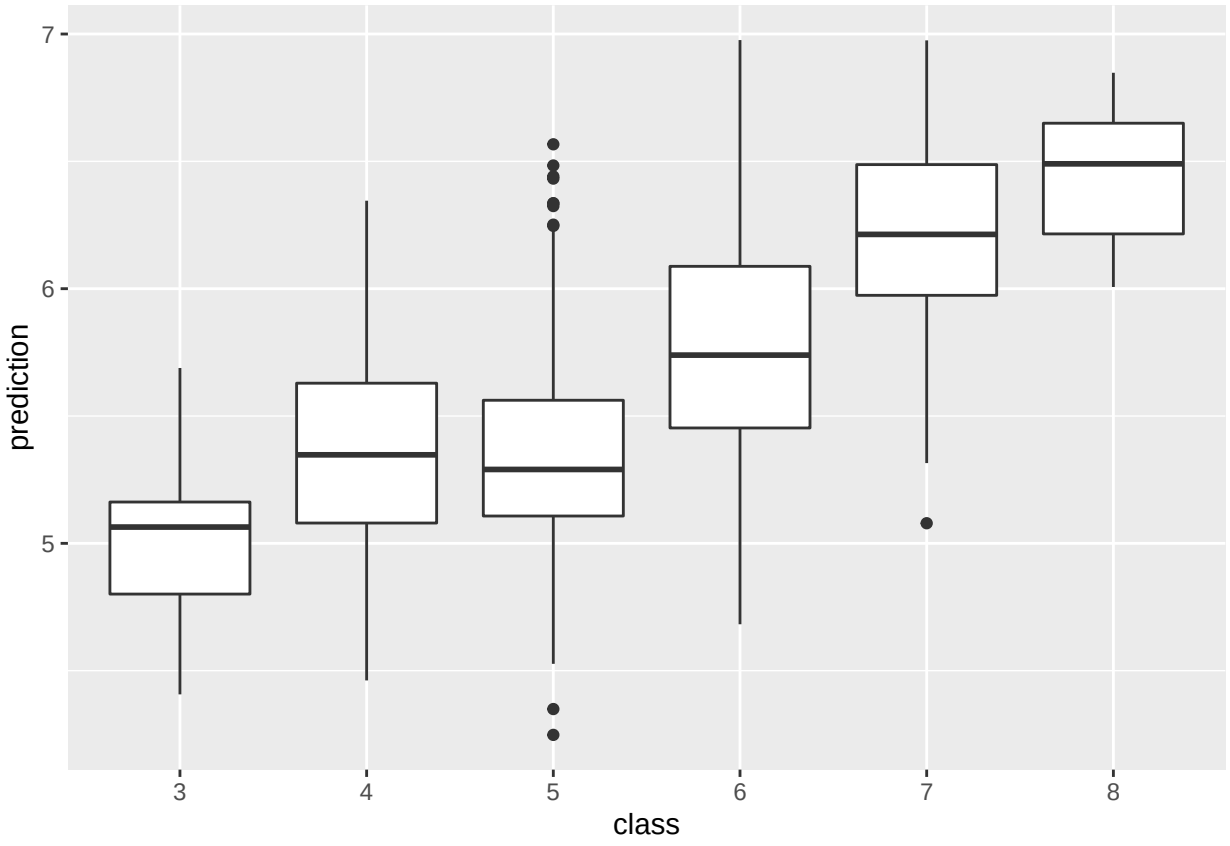

Figure 4: Boxplot of predictions by observed class

Support Vector Machine with Linear kernel

My next step was to compare the linear approach to using a Support Vector Machine (SVM). This has one tunable parameter, cost, representing the size of the soft margin of the hyperplanes, smaller values mean a larger soft margin.

From fig. 5 we see that we get the best result with a cost parameter of 0.0065.

Table 5: Regression Error Characteristic (tolerances 0.5 and 1.0).

model	T=0.5	T=1.0
Naive	0.399	0.950
Linear Regression	0.603	0.891
SVM - linear	0.593	0.888

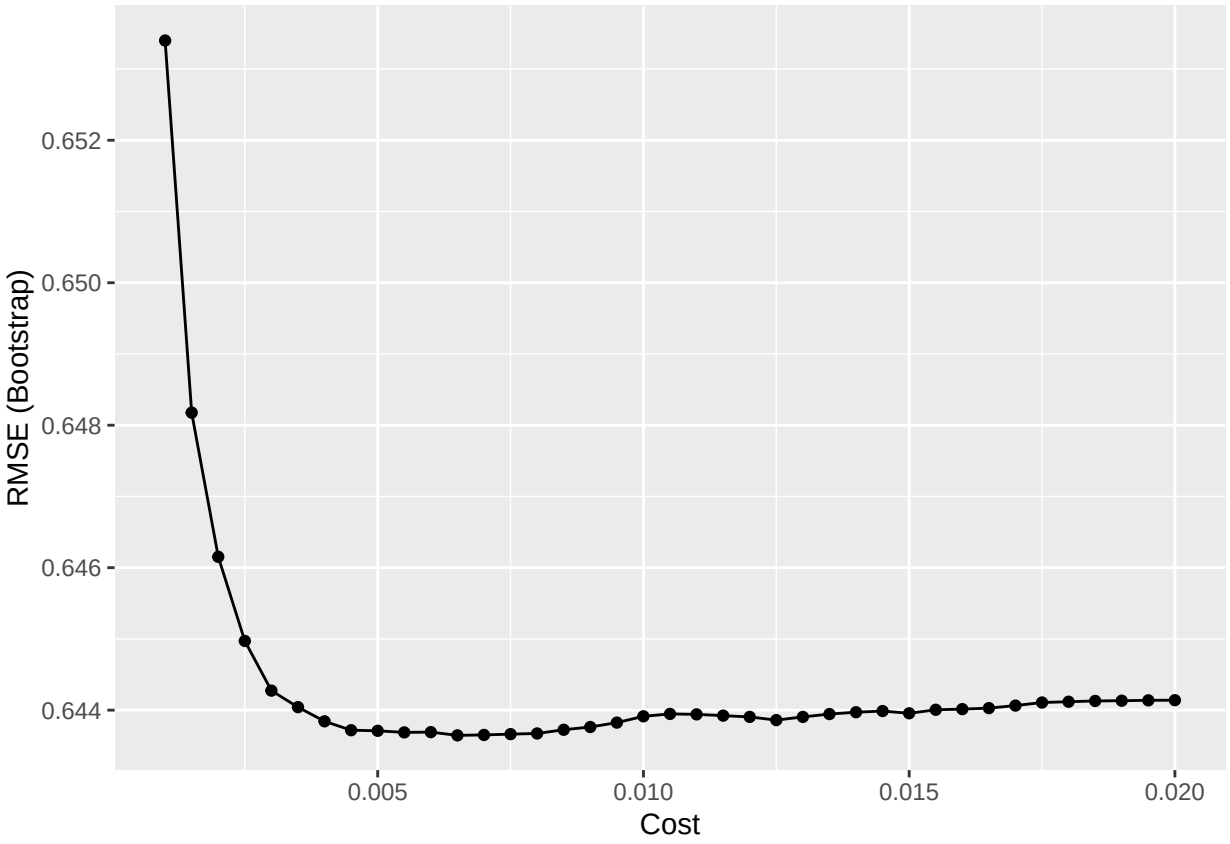

Figure 5: Cost vs. RMSE for Linear SVM

As table 5 shows, our overall accuracy goes down a little compared to plain linear regression.

Table 6: Regression Error Characteristic class (tolerance 1.0)

model	3	4	5	6	7	8
Naive	0	0.000	1.000	1.000	1.000	0
Linear Regression	0	0.149	0.953	0.974	0.737	0
SVM Linear	0	0.234	0.959	0.965	0.693	0

Looking at table 6 and figure 6 we see that the change in overall accuracy comes from being less accurate for the highest classes.

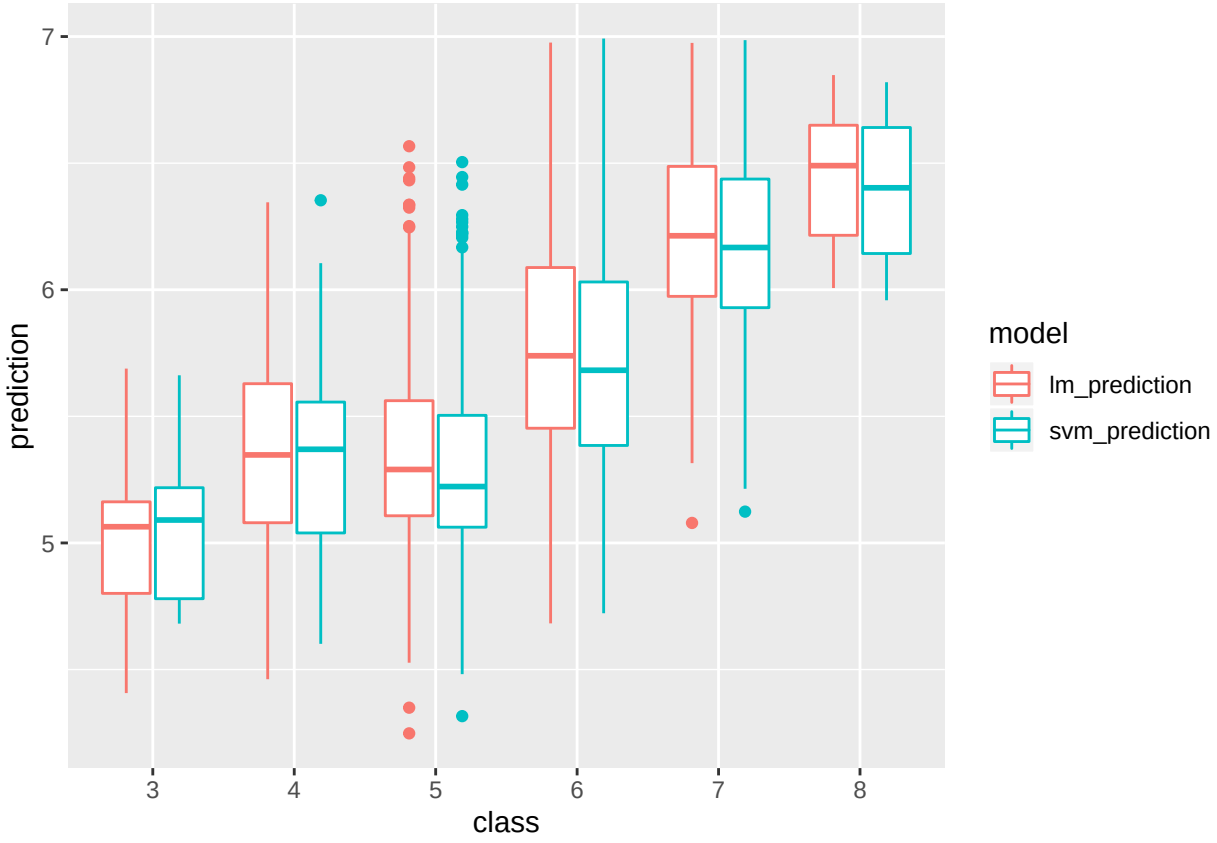

Figure 6: Boxplot of predictions per class - linear model and svm linear

SVM Polynomial

A slightly more flexible approach is to use a Support Vector Machine with a polynomial kernel. It is tuned with the same cost parameter, but with the addition of a degree for the polynomial kernel. It has the potential to be more accurate when the hyper-boundaries between classes are more complicated than the linear approach accounts for.

Figure 7 shows that we get the best result with a degree of 2 and a cost of 0.003.

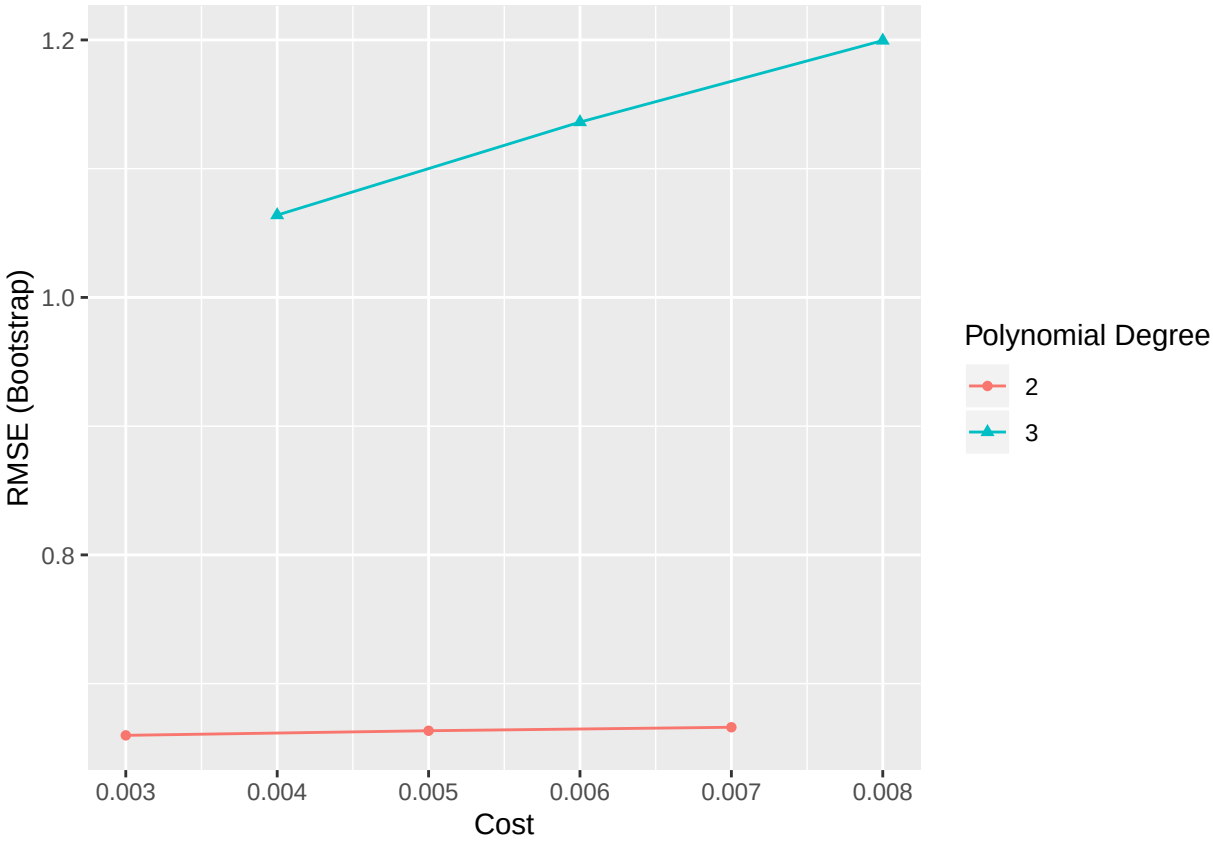

Figure 7: Tuning parameters for SVM polynomial model

Table 7: Mean Absolute Deviation and Regression Error Characteristic (tolerances 0.5 and 1.0).

model	T=0.5	T=1.0
Naive	0.399	0.950
Linear Regression	0.603	0.891
SVM - linear	0.593	0.888
SVM poly	0.633	0.900

Table 8: Regression Error Characteristic class (tolerance 1.0)

model	3	4	5	6	7	8
Naive	0	0.000	1.000	1.000	1.000	0.000
Linear Regression	0	0.149	0.953	0.974	0.737	0.000
SVM Linear	0	0.234	0.959	0.965	0.693	0.000
SVM Poly	0	0.191	0.969	0.974	0.732	0.062

Using this model to predict the training data gave tables 7 and 8 showing a significant improvement over the previous models, but as can be seen from table 8, at a cost for the low prevalence quality 4.

Ordinal Random Forest

The final algorithm I wanted to try is an Ordinal Random Forest (ORF). According to documentation for the package `ordinalForest` the core idea is to assume there is some underlying continuous variable mapping to a known ordinal variable such as the quality in the data set in this report. This means the algorithm not only takes into consideration that predicting a 4 for class 5 is better than 3, but also that, unlike mapping the ordinal variable to a numerical and doing regression, it doesn't assume a quality 6 wine is twice as good as a quality 3 wine, but that there *is* some numerical relationship between them, perhaps it's 20% better, or 150% better. This is likely to be closer to reality than running a regression.

ORF seeks to approximate such an underlying reality by: * generating multiple random mappings from ordinal to continuous variable, called score sets, * creating a random forest for each, with a limited number of trees, * picking the best performing score sets and averaging them * creating a larger random forest with this optimized score set

In caret the algorithm can be trained on the number of score sets to use initially (*nsets*), the number of trees used to evaluate the score sets (*ntreeperdiv*) and the size of the final random forest (*ntreefinal*).

There is some discrepancy between the documentation for the `ordinalForest` package and `caret`. The default tune grid in caret has ntreeperdiv 50, 100 and 150, ntreefinal 200, 400, 600 and nsets 50, 100, 150. While the 'ordinalForest package has defaults 100 (ntreeperdiv), 5000 (ntreefinal) and 1000 (nsets).

Because the algorithm is quite time intensive and more so with large values I chose to run with the defaults in caret initially and then narrowed it in somewhat to get the results shown in fig 8.

Using the best tune from this figure, I got the results shown in table 9, but the near perfect classification on predicting the algorithm's input shows that this was not a valid comparison to the previous algorithms. However, since this algorithm trains on class accuracy we can compare the accuracy of the best tune (abt. 0.668) with the average bootstrap of 0.631 which was our best *T 0.5* so far.

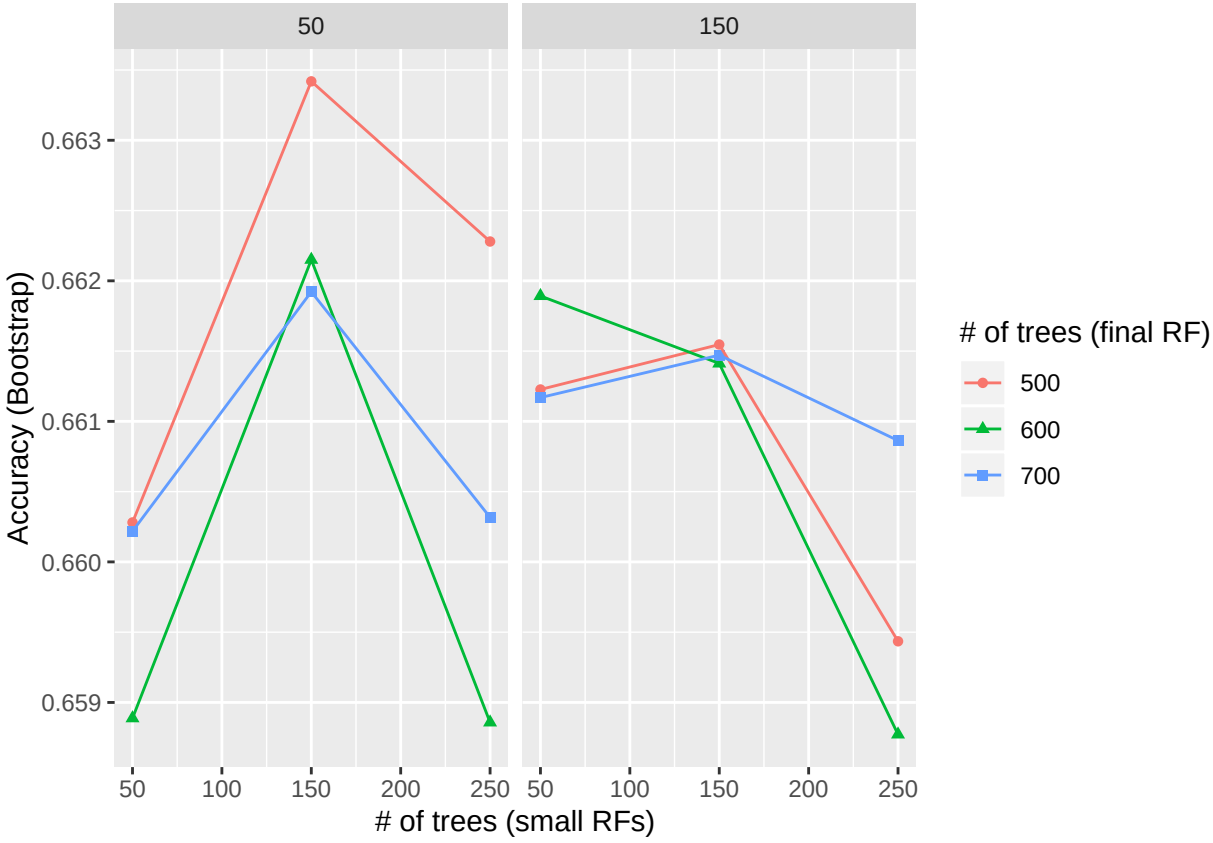

Figure 8: Tuning Ordinal Random Forest

Table 9: Mean Absolute Deviation and Regression Error Characteristic (tolerances 0.5 and 1.0).

model	T=0.5	T=1.0
Naive	0.399	0.950
Linear Regression	0.603	0.891
SVM - linear	0.593	0.888
SVM poly	0.633	0.900
ORF	0.998	1.000

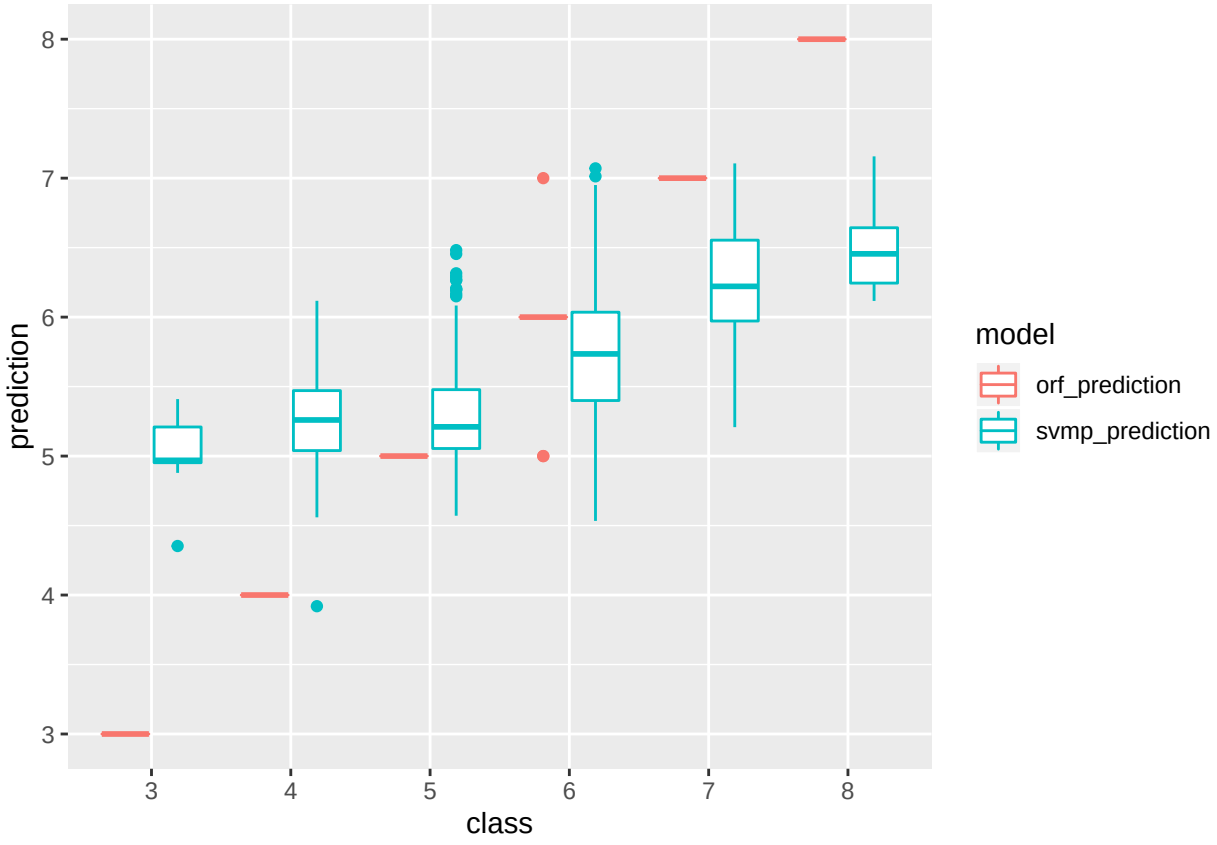

Figure 9: Prediction on the training set by SVM-poly and Ordinal Random Forest

The big disparity between the accuracy we get on the bootstrap samples, and what we get when predicting the whole training set by itself indicates the algorithm is overfitting, which we also see in fig. 9. I looked into different possible ways to deal with this, such as setting samples per node higher, but I thought that would be problematic for the tiny class size for quality 3. So instead I tried reducing the size of the final forest, since it appeared from the results of training I did that smaller final forests were not much worse, and for some values of the other parameters actually better, than larger. I chose to tune again only on final trees and set both nsets and ntreeperdiv fixed at 50, since those gave relatively good results for the lowest number of trees in the previous training grid, and I tried final forests between 200 and 500. The training result is shown in fig. 10.

The best tunes training with those parameters were only marginally worse than the previous best, despite having fewer trees in the final forest. There was a lot of variation though, with a forest of only 250 having nearly as good results as one with 400 trees, so I plotted them again with errors bars representing one standard deviation, as shown in fig. 11

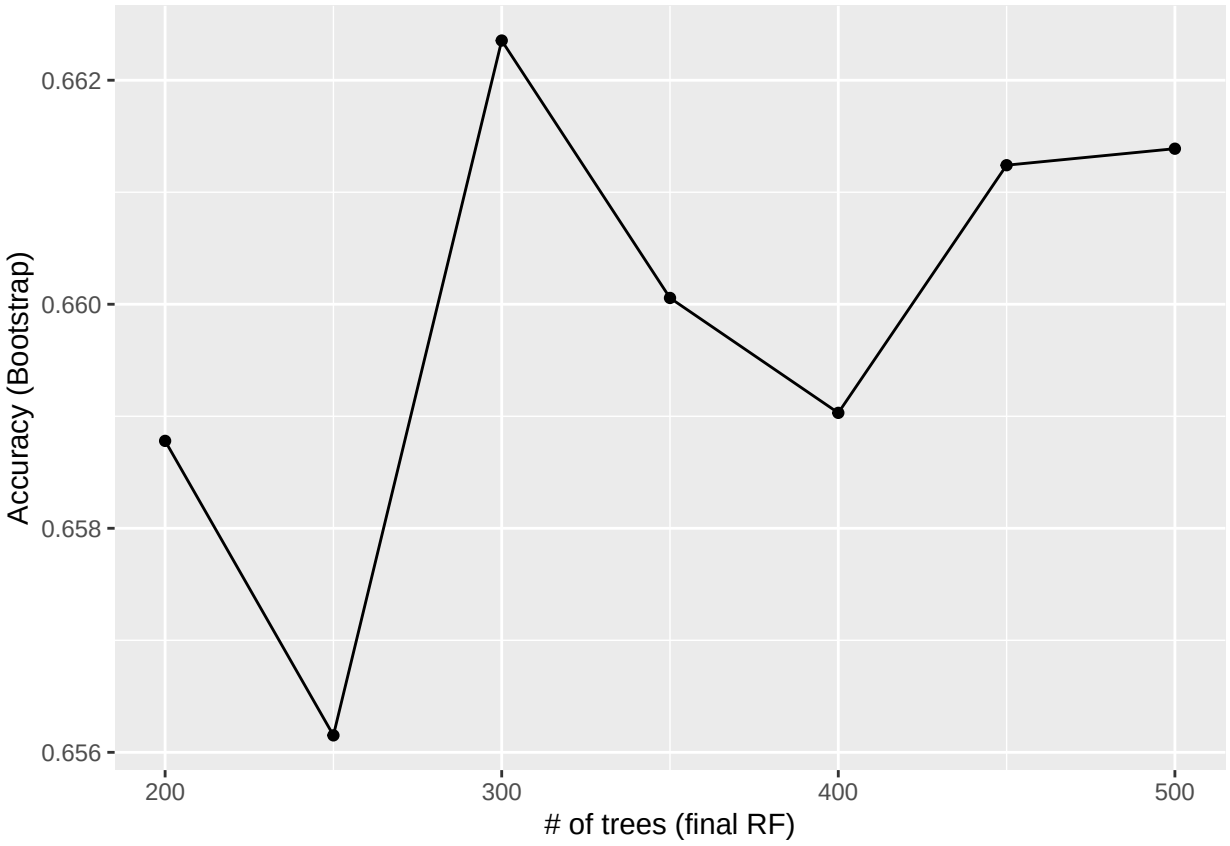

Figure 10: Training Ordinal Random forest with smaller forests

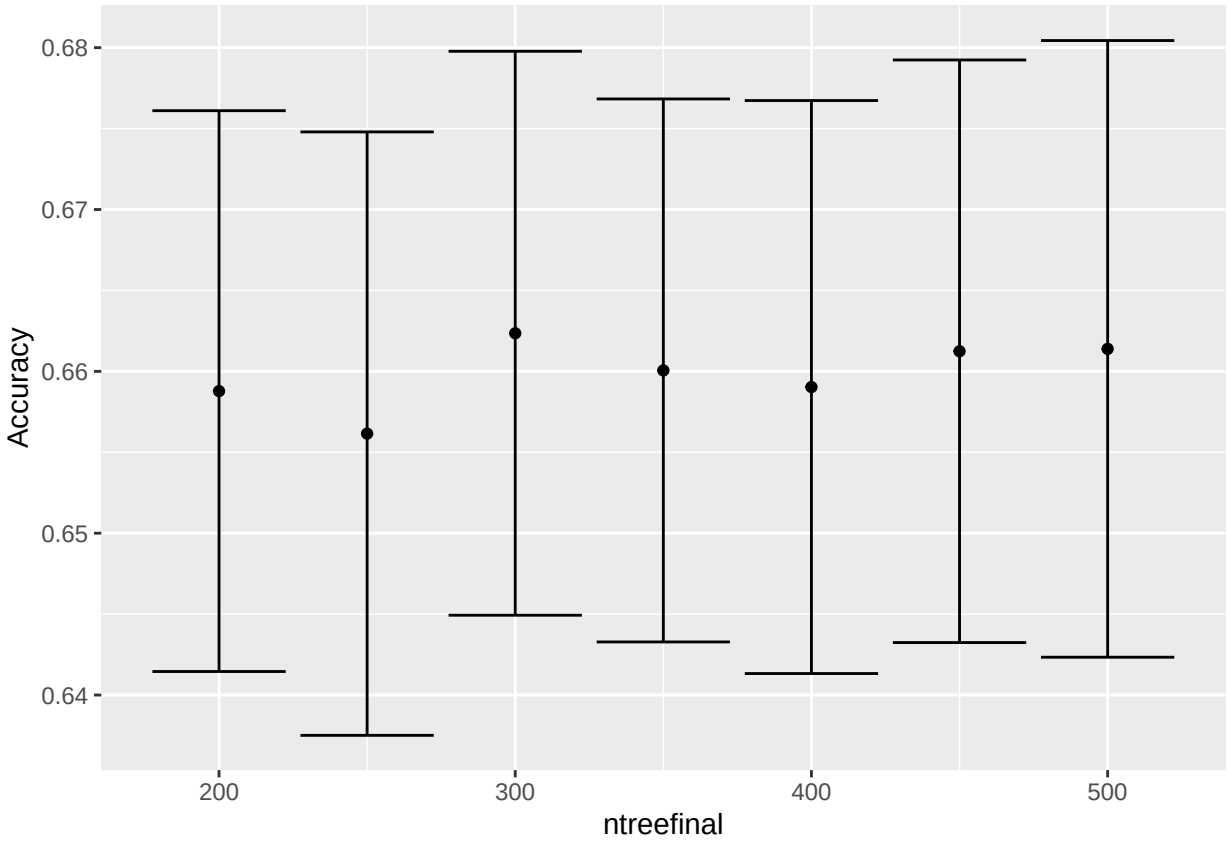

Figure 11: Training results with error bars for smaller forests

This showed that even for the smallest forests in this tune range, results were statistically equivalent to the larger. So I ran train one final time for even smaller forests (fig 12.).

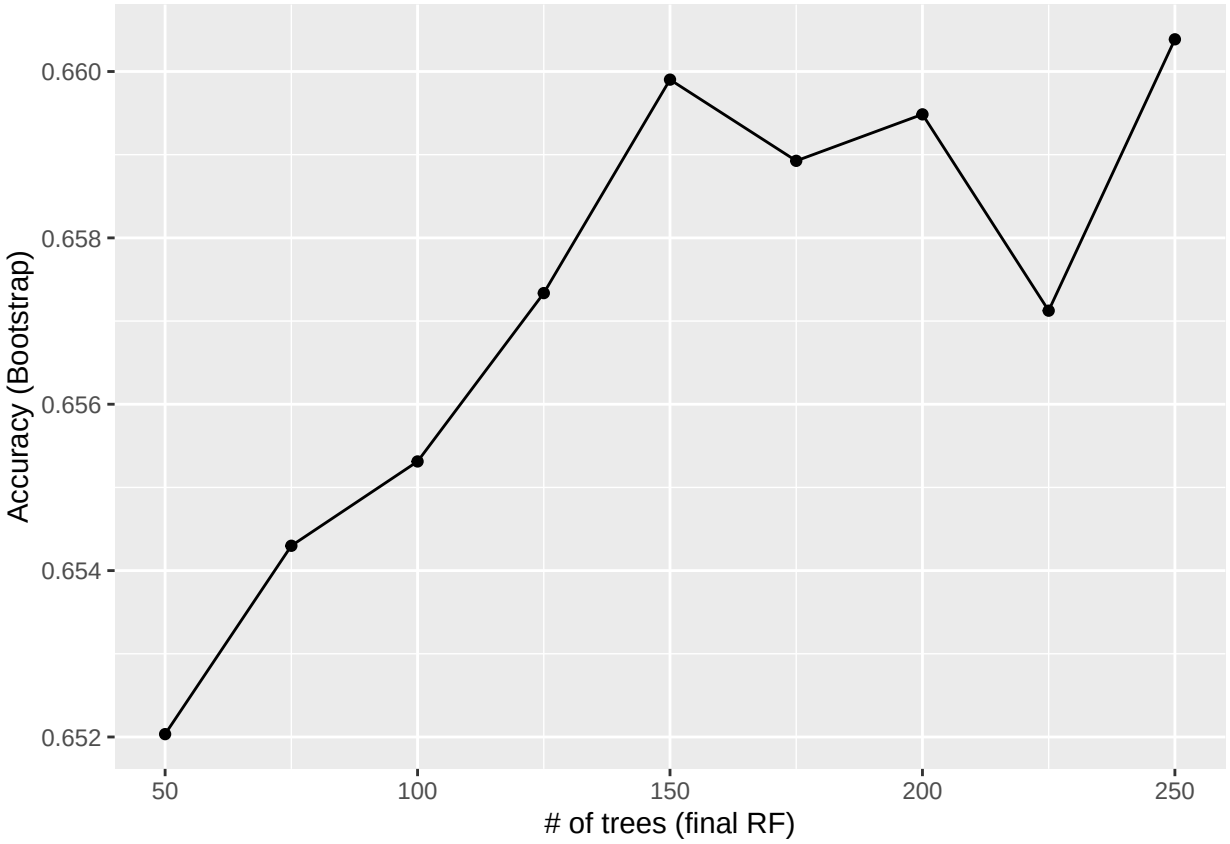

Figure 12: Training Ordinal Random forest with tiny forests

Table 10: Mean Absolute Deviation and Regression Error Characteristic (tolerances 0.5 and 1.0).

model	T=0.5	T=1.0
Naive	0.399	0.950
Linear Regression	0.603	0.891
SVM - linear	0.593	0.888
SVM poly	0.633	0.900
ORF	0.998	1.000
ORF - smaller	0.997	1.000
ORF - tiny	0.997	1.000

Using the best tunes with the smaller forests still gave near perfect predictions on the training set (table 10), but the tuning plot showed a more consistent increase in accuracy towards the larger forests in that range. And the maximum accuracy was almost as good as for a much larger forest.

Results

Although the results for Ordinal Forests predicting on the training set did appear to show overfitting I still believed it had the potential to be an improvement over the simpler models, because it showed the ability to predict 3s and 8s, where the others had predictions almost entirely confined to the interval 4.5 - 7. So I chose to not experiment with more algorithms but to run my final tests with the reserved test data.

The outcomes for all the algorithms predicting the test data is shown in tables 11 and 12.

Table 11: Regression Error Characteristic (tolerances 0.5 and 1.0).

model	T=0.5	T=1.0
Linear Regression	0.537	0.877
SVM - linear	0.531	0.864
SVM - polynomial	0.549	0.877
ORF	0.673	0.963
ORF - smaller	0.685	0.963
ORF - tiny	0.698	0.957

Table 12: Regression Error Characteristic class (tolerance 1.0)

model	3	4	5	6	7	8
Linear Regression	0	0.333	0.928	0.922	0.85	0.0
SVM Linear	0	0.333	0.928	0.906	0.80	0.0
SVM polynomial	0	0.333	0.942	0.906	0.85	0.0
ORF	1	0.500	0.971	1.000	1.00	0.5
ORF - smaller	1	0.500	0.971	1.000	1.00	0.5
ORF - tiny	0	0.667	0.971	0.984	1.00	0.5

As predicted by the results from training, the ORF models are more accurate overall when accepting one off errors, also on the test data. And as I expected due to them predicting more of the extreme values, they are also more accurate for every class. The smallest forrests performed very similarly or even better than the larger, lending support to my assumption that reducing the size of the forest could decrease overfitting, but it bears repeating though that in the test set there is only one wine of quality 3 and two of quality 8.

Conclusion

In this report I have shown the results of trying to predict Wine Quality scores from the Viticulture Commission of the Vinho Verde region using physicochemical properties of the wines measured during the certification process. I've shown that using SVM with a linear or quadratic kernel offers minimal improvement over linear regression and has problems with this specific dataset due to the qualities being ordinal data and due to the high prevalence of middling qualities. I have further shown that at an algorithm specifically designed to make predictions on ordinal data, the Ordinal Forest approach, outperforms these regression approaches.

Although the final algorithms appear to perform well based on the error metric I set out to judge them by, it is possible further improvement could have been made by training based on them, instead of by overall accuracy. Determining if that is the case would have required a much closer look at the Ordinal Forest algorithm.

References

Dua, D. and Graff, C. (2019). UCI Machine Learning Repository http://archive.ics.uci.edu/ml . Irvine, CA: University of California, School of Information and Computer Science.

P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis (2009). *Wine Quality Data Set* UCI Machine Learning Repository, Dataset, 25 Nov 2019, https://archive.ics.uci.edu/ml/datasets/Wine+Quality .

P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis (2009). 'Modeling wine preferences by data mining from physicochemical properties [Preprint]'. To be published in *Decision Support Systems*. Available at: http://repositorium.sdum.uminho.pt/bitstream/1822/10029/1/wine5.pdf (Accessed: 25 Nov 2019).