Artificially Intelligent Wine Tasting

A report on the performance and utility of an "AI Sommelier"

23 April, 2018

Contents

1 Executive Summary

1.1 Project Goals

This research project investigates the performance and utility of an "AI sommelier" - Delicious AI's flagship, eponymous, artificial intelligence application for the automatic identification and rating of high quality wines. For Delicious AI, the "AI sommelier" is likely to be the first of many artificial intelligence applications focusing on the food and drinks market. Understanding the "AI sommelier's" performance and usability is vital for Delicious AI's long-term vision of creating artificial intelligences with an appreciation for quality food. In

order to develop an AI that can create high-quality novel cuisine, it is important to establish whether an AI can identify quality in the first place.

As part of the initial scope of this project, we have been employed by Delicious AI to investigate the possibility of using machine learning techniques to automatically evaluate the quality of wines in a reliable, accurate, efficient, and scalable manner. In addition to creating an AI sommelier, we determine which features are associated with highly rated wines, evaluating the predictive power of various chemical components as well as wine color. We later explore whether it is possible for an algorithm to help create a wine whose quality exceeds that of those previously observed. Finally, we develop a revolutionary super-human AI sommelier prototype that could be extended to predict wine quality from grapes that are still on the vine, guiding the wine-making process from beginning to end.

In this report we summarise the results of our investigations and provide our recommendations for Delicious AI's "AI sommelier" product. We trust that it will provide Delicious AI with useful information on the performance and utility of an "AI sommelier", and additionally inform the extent to which Delicious AI allocates analytical resources towards developing food and drink AIs.

1.2 Key Findings

We build and test the performance of five different AI Sommelier models: an intercept-only dummy regressor model (DR), a linear regression model (LR), a non-linear support vector machine (SVM), a random forest (RF), and a multi-layer perceptron (MLP). We also include two different methods of categorizing the quality of wines, either treating the 1-10 ratings as a continuous variable (univariate regression) or a categorical variable (deterministic classification). We find that our models perform best when the ratings are treated as a continuous variable and are pleased to report that the RF Sommelier is our highest performer, with a prediction error on average of roughly 0.4 points on a 1-10 scale. The agent with the lowest performance (apart from the dummy) is the MLP Sommelier.

One of the advantages of the RF Sommelier is interpretability. Using the RF Sommelier, we determined that alcohol is the most important predictor of wine quality, where ratings increase with alcohol content. The next most important factor is volatile acidity (i.e. mass concentration of acetic acid), which tends to be lower in good wines. Fixed acidity and wine color (red or white) are the least important factors.

We find that none of the models are able to accurately predict wine quality from wine color alone. This aligns with the low importance that the RF Sommelier places on color and suggest that neither red nor white wines tend to be rated higher. However, many of the AI Sommeliers predict quality more accurately when they are aware of a wine's color, so it remains important to include color in the model, likely because the color interacts with other variables.

A few potential "perfect wine" recipes are presented at the end of the paper. The SVM Sommelier (the second highest performer) and LR Sommelier are well-suited for the task, as they are both capable of predicting values beyond the range of the data used in training, enabling them to envision a "perfect 10" wine. Not all the AI sommeliers could agree on the ratings of the AI-generated recipes. However, we notice that both the LR Sommelier and SVM Sommelier came up with wines that have relatively high levels of free sulfur dioxide and low levels of total sulfur dioxide, suggesting these characteristics improve wine quality.

Finally, we present a "super-human"" RF Sommelier that is able to predict the quality of a wine from a subset of features that can be detected before the fermentation process. These include color, fixed acidity, citric acid, residual sugar, and chlorides. The super-human RF Sommelier has a mean error of about 0.5 points, and manages to outperform the LR Sommelier trained on all the features.

1.3 Conclusions

We conclude that we can successfully develop an AI Sommelier that can accurately predict the quality of wine in accordance with expert ratings. We also find that there are empirical characteristics associated with

wines rated highly by sommeliers, such as increased alcohol content. The AI Sommeliers we recommend in this paper have the advantage of consistency, scalability, and interpretability. Although the predictions are most accurate when using the full set of wine characteristics, including color, the AI Sommeliers are still able to predict quality from just a subset of the variables, enabling us to develop a novel early-detection algorithm that could be used to predict the quality of a wine before the fermentation process even begins. We hope that our AI Sommeliers can help add value to the wine production process and encourage human sommeliers to try the new AI-generated recipes for themselves.

1.4 Limitations

Without more information, we are unable to determine with certainty if the AI Sommeliers can outperform humans' predictive performance on any given metric. It would help to have the individual wine ratings from each human sommelier so that we could have a better idea of human accuracy and the variance across individual wine ratings. We would also benefit from external, objective metrics associated with each wine (e.g., bottles sold per year) to further assess our models' versatility relative to that of humans. In addition, the models were limited to considering the variables included in this particular dataset. There may exist additional variables related to wine quality that would improve the predictive power of the models. Finally, obtaining more data on wines on the high and low ends of the quality spectrum would improve performance.

We are also unable to assert that the "perfect wines" would be rated a 10 by human sommeliers, since it is possible that the predictive models do not hold for extreme values of the features. A human expert would need to assess the quality of the proposed wine recipes. A final limitation is that the AI Sommeliers are only able to assess quality in accordance with expert ratings, which may differ from a winery's target market. By collecting additional data that is specific to a particular demographic (e.g., college students), wineries may re-train the AI Sommeliers to acquire a new taste that is more relevant to their individual goals.

2 Introduction

2.1 Research questions

The specific, scientific, empirically quantifiable questions we propose to answer in this project are:

1. Supposing that certain variables are detectable early in the wine-making process, is it possible to train a model on these variables alone that is sufficiently predictive of the sommelier ratings?
2. Which features are most strongly associated with high-quality ratings?
3. Is it possible to simulate a (reasonable) dataset of wine upon which a learner estimates a rating of 10?
4. Can we use relationships in the raw data between objective qualities and wine ratings to produce a model that is predictive of the expert ratings?

The first question we consider is whether we can use the dataset to create an AI with super-human performance in wine-tasting. However, since we lack data on an external metric by which we can evaluate human performance, we cannot say whether Delicious AI can perform "super-humanly" on such a metric. The ratings we have are not a comparative measure of human performance. Absent a measure of of inter-rater reliablity, for example, we cannot say whether Delicious AI rates wines better than a human.

However, Delicious AI still has other qualities which make it super-human in wine-tasting tasks. For example, Delicious AI has a super-human ability to be consistent in its ratings of wines. While a human sommelier may occasionally evaluate the same bottle of wine differently depending on mood, context and other factors, Delicious AI will reliably give the same rating to a given bottle of wine, with a known margin of error. Second, once a data-gathering system is in place at a winery, Delicious AI will be super-human in its ability to rate wines at scale. An individual sommelier would need days or weeks to systematically taste and evaluate every batch of wine produced by a winery.

Furthermore, each wine would have to be evaluated by multiple sommeliers in order to calculate a mean rating that averages out the potential biases and differences between individual sommeliers. Meanwhile, Delicious AI alone can evaluate all of the batches of wine in a matter of minutes, incorporating the subjective tastes of a variety of human sommeliers. Perhaps most significantly, Delicious AI can periodically evaluate the wines at different stages of the production process, predicting the future quality of a wine before it even exists. One of the main focuses of our investigation will be developing an early-prediction algorithm for wine quality.

Second, we would like to determine which components constitute a good wine. In order to empirically analyze this question, we define a "good" wine as a wine which is rated highly (above 5) by the sommeliers in our dataset. In theory, we could also identify wines considered "good" by individuals of a specific demographic (e.g., young consumers in Eastern Europe), but this would require additional data, so we will instead focus on finding the components of wine that are associated with high ratings from sommeliers. The simplest way to scientifically investigate this question is to analyze correlation plots between wine quality and individual variables from the wine dataset.

Similarly, we can look at the coefficients in a linear regression model to see how much an increase in the value of a particular variable will increase the predicted rating, conditioned on the other variables. These methods will provide new insight into which components are associated with higher ratings. There may be additional components of wine outside the scope of our dataset which serve as important indicators of quality, but we would need additional data to investigate this possibility.

Next, we would like to assess whether we can create a "perfect" wine whose quality exceeds that which we have seen in our data. In order to conceive a grade 10 wine, Delicious AI would have to extrapolate beyond the labels and combinations of predictors seen in the training data, so it is difficult to guarantee that the model would be valid for these new ranges of the variables. We have no reason to believe that the assumptions made by the models will hold for a novel combination of chemicals, nor do we make any claims that such a combination is physically possible or safe for human consumption.

Despite the lack of performance guarantees, we can still use Delicious AI for creative inspiration, identifying

which novel combinations of variables Delicious AI would expect to produce a perfect wine. In this way, Delicious AI can suggest new combinations of features that wine producers may not have otherwise considered.

Fourth, we would like to see how accurately Delicious AI can predict wine ratings from the features alone. A high prediction accuracy will at least confirm that the ratings among sommeliers are not random and correspond to empirically verifiable differences in wine components. For this question, it would be helpful to know the variance of ratings for each wine and how consistently human sommeliers rate the same wine. For example, would a sommelier rate a wine differently if it were presented in a different bottle? We would need to conduct an experiment to answer these questions. A wider, more uniform, distribution of ratings would also help our models differentiate between high and low quality wines. However, we can still use the dataset for insight into whether wines rated relatively highly tend to have a different set of features than wines rated poorly. We can again look at the variable distributions to determine whether the mean of a predictor is different among good and bad wines. Finally, in addition to our empirical research questions, we consider a question raised by a philosopher of ethics - is the human perception of wine quality so subjective that there is no empirically verifiable correlate of good and bad wine? This is a perfectly reasonable question to ask, considering that new sommeliers are trained to match the opinions of existing sommeliers, so human biases could easily perpetuate for generations. This consideration aside, in theory, the blind-tasting of the wines should have mitigated the influence of factors exogenous to the wines themselves.

Of course, Delicious AI is only trained to approximate *expert* opinion, which is not necessarily an objective measure of wine quality and may differ from individual preferences or the preferences of the general wine-drinking public. We deem our effort nonetheless valuable because expert opinion tends to drive market prices, and we can always follow similar methodologies with different data in order to predict individual or demographically relevant ratings. Similarly, with additional data, Delicious AI could predict external metrics such as quantity sold or awards received.

3 Methods

In this section we describe the modelling approaches we undertook in this study. We also outline all key decisions made in our work.

3.1 Overview

The main purpose of our investigation was to explore the feasibility of an "AI sommelier". In order to do this, we conducted a series of predictive benchmarking experiments to compare the performance and utility of various machine learning algorithms.

We used each algorithm to determine:

1. whether, and how well, wine quality can be predicted from chemical composition and colour;
2. whether wine colour adds predictive power above chemical composition and vice versa, in 1.

Our predictive benchmarking experiments were first carried out with wine quality treated as a continuous variable (univariate regression) and then repeated with wine quality treated as a categorical variable (deterministic classification). The nature of the wine quality scale is such that both approaches were deemed reasonable. For more information on the dataset, see Appendix A.1.

3.2 Algorithms

The following five algorithms were analysed in all experiments:

- an intercept-only dummy regressor model (DR)
- a linear regression model (LR)
- a non-linear support vector machine (SVM),

- an ensemble of (at least 10) trees (RF)
- a neural network with two or more middle layers (MLP)

We used the intercept-only dummy regressor (mean) model as a "best guess" performance baseline against which the other algorithms could be compared. The linear/logistic regression model was chosen because it is a simple, fast, and readily intepretable model that would allow us to understand relationships between the quality of a wine and its characteristics. The SVM has previously been identified as the best performing model for classifying wine by quality (Cortez et. al. 2009).

Like the linear regression model, Random Forests are relatively easy to understand and quick to implement. Additionally, they are able to deal with a minimally pre-processed input data, can perform automatic feature selection, and provide information on the importance of model features.

Finally, neural networks are also able to create and combine new features independently, and perform well on classification tasks. Although it is harder to interpret the relationship between features and classification decisions, neural networks have the advantage of being able to 1) work with unlabelled data, so that Delicious AI could develop its own taste; 2) carry out multitask learning, which is useful when Delicious AI expands to other products; and 3) learn embedded representations of data, such as the taste sensor data that may be available in the future.

3.3 Optimisation

In order to achieve the best performance from each algorithm for each task, we optimised model hyper-parameters using randomised search with 3-fold cross-validation. In the randomised search method, a fixed number of parameter settings is sampled from specified distributions, and then optimised by cross-validated search over parameter settings.

Since the randomised search method does not test every single parameter in the search space, it is considerably faster than a full grid search. Using cross-validation allowed us to make the most of our training data, whilst also lowering the risk of overfitting our model to a specific train/validation split. Hyper-parameters were optimised separately for the regression task (to minimise the mean squared error) and for the classification task (to maximise accuracy). For more information about the tuned hyperparameters, see Appendix A.2.

3.4 Performance metrics

Two performance metrics were used to compare the predictive performance of our learners in the univariate regression task - mean squared error and mean absolute error. In addition to these two metrics, an accuracy score was also evaluated for the classification task.

3.5 Model validation

The standard protocols for carrying out a supervised learning benchmarking experiment were followed in this study. Of the 6,497 wines in the dataset, a random 20% sample (1,299 wines) was held out as a separate test set. The same test set was used for all benchmarking experiments to evaluate the performance and generalisability of our different models on previously unseen data. The test set was not used for any model training or hyper-parameter optimisation purposes. The remaining 80% sample (5,198 wines) was used as a training set upon which each model could be learned and hyper-parameters could be optimised.

The learners were all evaluated on the same test set for both the regression and classification tasks using the previously outlined performance metrics. The mean and 95% bootstrapped confidence intervals for the each performance metric on the test set were calculated for each algorithm and each task. Bootstrapped resamples with replacement were taken from the test set in order to do this. Each resample was the size of the test set (1,299 wines) and the resampling process was repeated for 1,000 iterations. The results of this process can be seen in Section 5.1.

3.5.1 Additional methodological considerations

In line with the project specification provided by Delicious AI, all wines were retained in all benchmarking experiments i.e. outliers were not removed. We used the full range of wine quality values present in the dataset i.e. the target variable was not grouped into broader categories. Both red and white wines were analysed together so that models learned from a single dataset, rather than separate models by colour. For consistency across learners, we did not specify any feature interactions, as different learners determine relevant model features in their own ways. All variable transformations were carried on both training and test sets. The relative importance of different features was evaluated using the RF regression model, whilst the strength of the association between features and ratings was evaluated using the LR model.

4 Exploratory analysis

First, we confirm that there are no missing entries in the dataset. We also notice that there are many more white wines (4898) than red wines (1599). Next, for each variable, we plot a summary of the range, mean, median, and first and third quartiles, shown below. A few variables in particular stand out. Citric acid, for example, has a minimum value of zero. Further investigation shows that there are 151 zero entries for citric acid. This indicates that it is an optional component in wine-making. In addition, we see that different variables take on quite different ranges of values. For instance, while "volatile acidity" ranges from 0.08 to 1.58, "free sulfur dioxide" ranges from 1 to 289. This suggests that we should try models in which we normalize the variables, although it would be easier to interpret unscaled models. We also notice that the ratings range from 3 to 9, indicating that no wines were "perfect" enough to achieve a 10 or awful enough to be rated 1 or 2.

Table 1: Summary of Wine Data

	Range					Moments	
Variable	**Min**	**1st Quantile**	**Median**	**3rd Quantile**	**Max**	**Mean**	**S.D.**
alcohol	8.00	9.50	10.30	11.30	14.90	10.49	1.19
chlorides	0.01	0.04	0.05	0.06	0.61	0.06	0.04
citric acid	0.00	0.25	0.31	0.39	1.66	0.32	0.15
color	0.00	0.00	0.00	0.00	1.00	0.25	0.43
density	0.99	0.99	0.99	1.00	1.04	0.99	0.00
fixed acidity	3.80	6.40	7.00	7.70	15.90	7.22	1.30
free sulfur dioxide	1.00	17.00	29.00	41.00	289.00	30.53	17.75
pH	2.72	3.11	3.21	3.32	4.01	3.22	0.16
quality	3.00	5.00	6.00	6.00	9.00	5.82	0.87
residual sugar	0.60	1.80	3.00	8.10	65.80	5.44	4.76
sulphates	0.22	0.43	0.51	0.60	2.00	0.53	0.15
total sulfur dioxide	6.00	77.00	118.00	156.00	440.00	115.74	56.52
volatile acidity	0.08	0.23	0.29	0.40	1.58	0.34	0.16

We see that quite a few variables are skewed right or have large outliers, as indicated by a higher mean than median and relatively high maximum. These include chlorides, residual sugar, and sulphates. Residual sugar in particular has a median of 3 but a maximum of 65.8. Upon looking at a table of values we see that this is due to just one outlier and the second highest value is 31.6. Figure 1 has histograms representing the distributions of these three variables can be found below.

To correct for the skew, we consider log-transforming the predictors. The box-and-whisker plot, Figure 2, shows how the log-transformation has the potential to even out the distributions of individual variables.

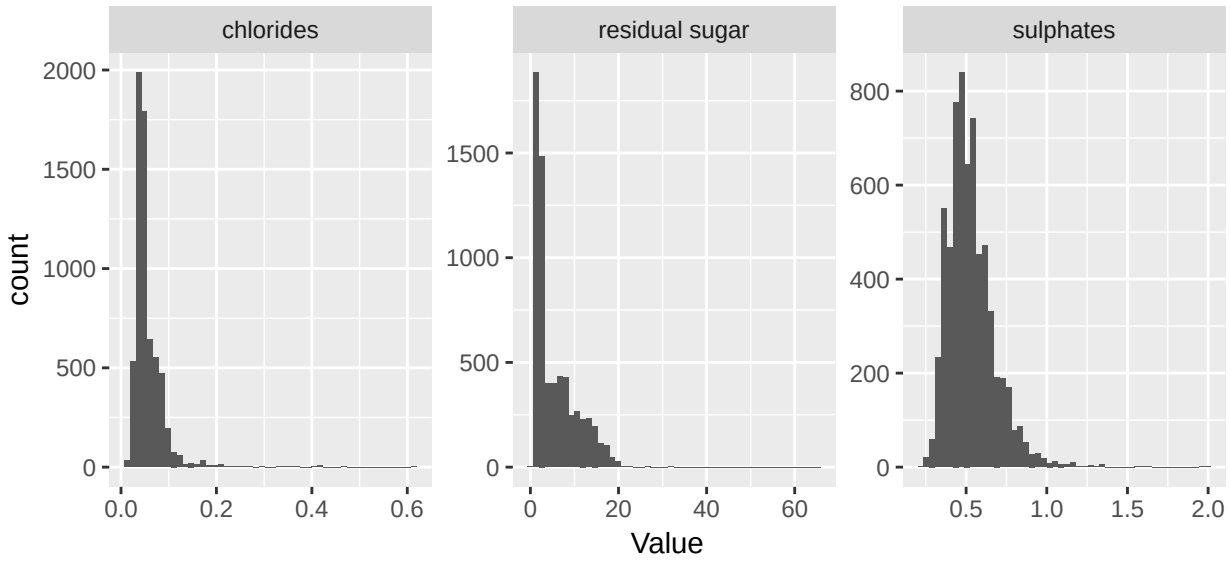


Figure 1: Histograms of Chlorides, Residual Sugar, and Sulphates

Additionally, research has demonstrated that log transformations tend to help stabilize the variance of variables pertaining to concentration data[1].

A series of matrix plots, Figure 4, do not appear to suggest that there are any non-linear relationships in the data. However, we do see a positive linear relationship between "free sulfur dioxide" and "total sulfur dioxide." The correlation coefficient for the two variables is 0.72 and a correlation plot, Figure 3, is shown below. It is not surprising that the variables are related, since total sulfur dioxide is defined as the sum of free and bound sulfur dioxide. There is also slight collinearity between residual sugar and density, which has a correlation coefficient of 0.55. Third, we notice a high negative correlation between alcohol and density of -0.69. It is intuitive that increasing the alcohol percentage would decrease the density, as alcohol is less dense than water and juice.

Next, we consider how wine color affects the variables. There are 4898 white wines and 1599 red wines. White wines tend to have a higher rating. 66% of white wines are rated above 5 and five white wines are rated 9, while only 53% of red wines are rated above 5 and none of them receive a rating of 9. To visualize the differences, we consider the distribution of each variable, partitioned by the discrete integer-valued ratings and wine color, Figure 5. Some of the density plots are based on a small number of samples - only 10 red wines and 20 white wines are rated 3, and only 18 red wines are rated 8. We therefore focus our analysis on the distributions corresponding to ratings between 4 and 7, which both have over 50 examples for each color.

We see from the plots that the color of the wine does indeed interact with the predictors. The plots reveal that red wines tend to have higher fixed acidity, volatile acidity, chlorides, pH and sulphates (see Appendix A.3 for the plots for all variables). Alternatively, white wines have higher citric acid, total sulfur dioxide and a distinct right-skew in the distribution of residual sugar. This confirms a natural intuition that it is more common for white wines to be sweet than red wines and that red wines tend to have more sulphates. We also notice an interesting trend with citric acid. While low-quality red wines tend to have less citric acid than low-quality white wines, the trend appears to reverse for high-quality wines. Red wine also has a wider range of citric acid values. However, it is hard to be certain that these particular trends are significant, since the number of samples is low for wines with especially high and low ratings.

These graphs also indicate that certain variables are distributed differently among high- and low-quality

[1]Igarashi, T. "The Rationale for Using Logarithmic Transformation of Concentration Data in Toxicokinetic Studies." The Journal of Toxicological Sciences., U.S. National Library of Medicine, Feb. 1995, www.ncbi.nlm.nih.gov/pubmed/7595977.

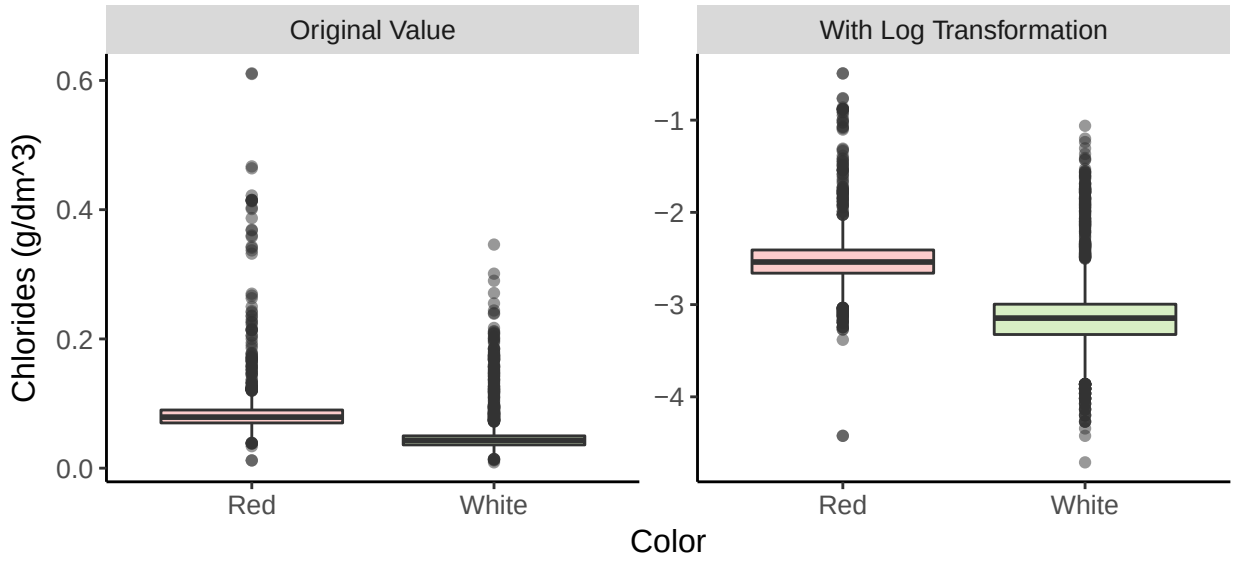


Figure 2: Chlorides with and without log transformation

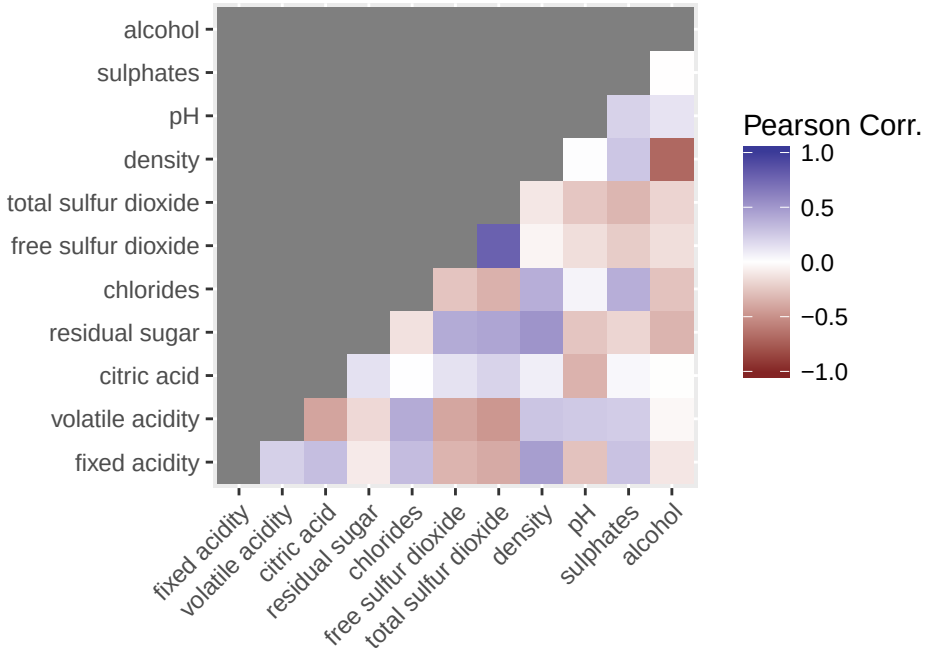


Figure 3: Untransformed Variable Correlations

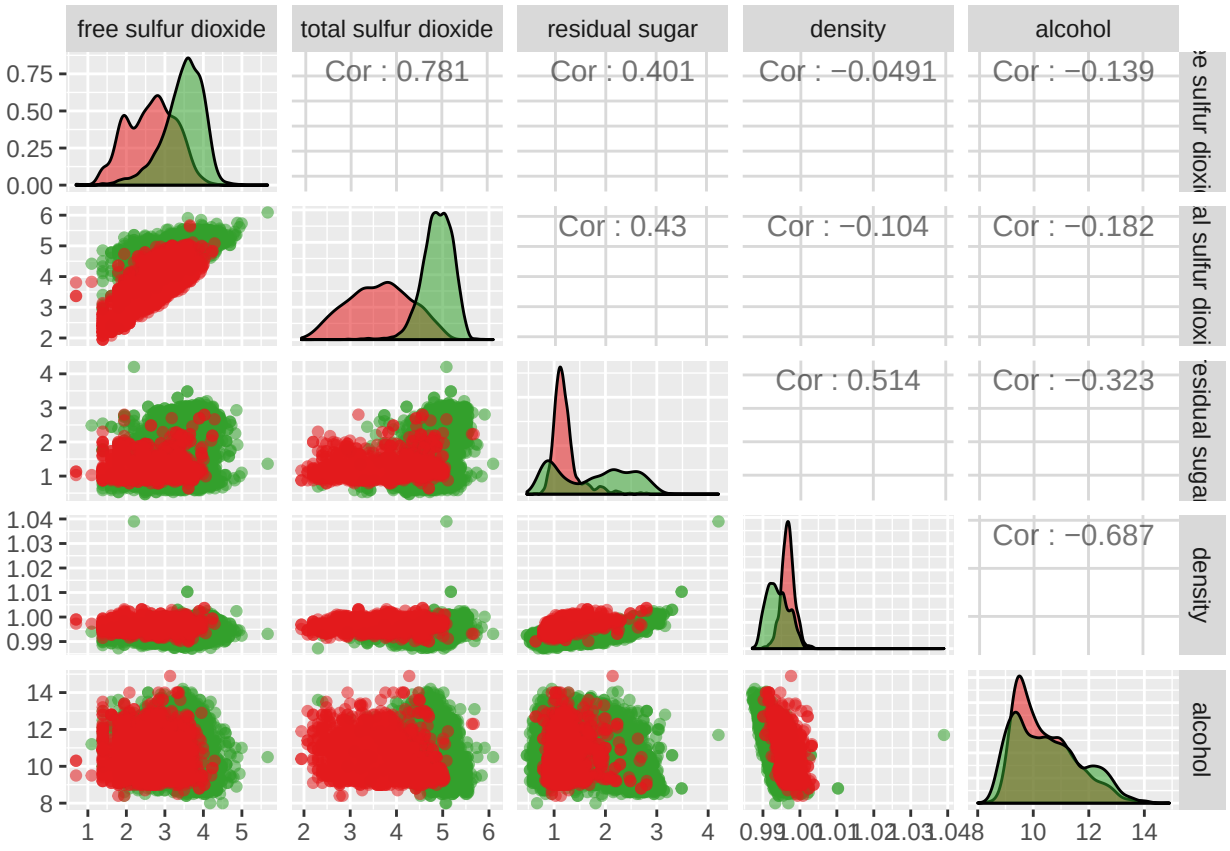


Figure 4: Pairs Plot of Select Variables (Log-Transformed)

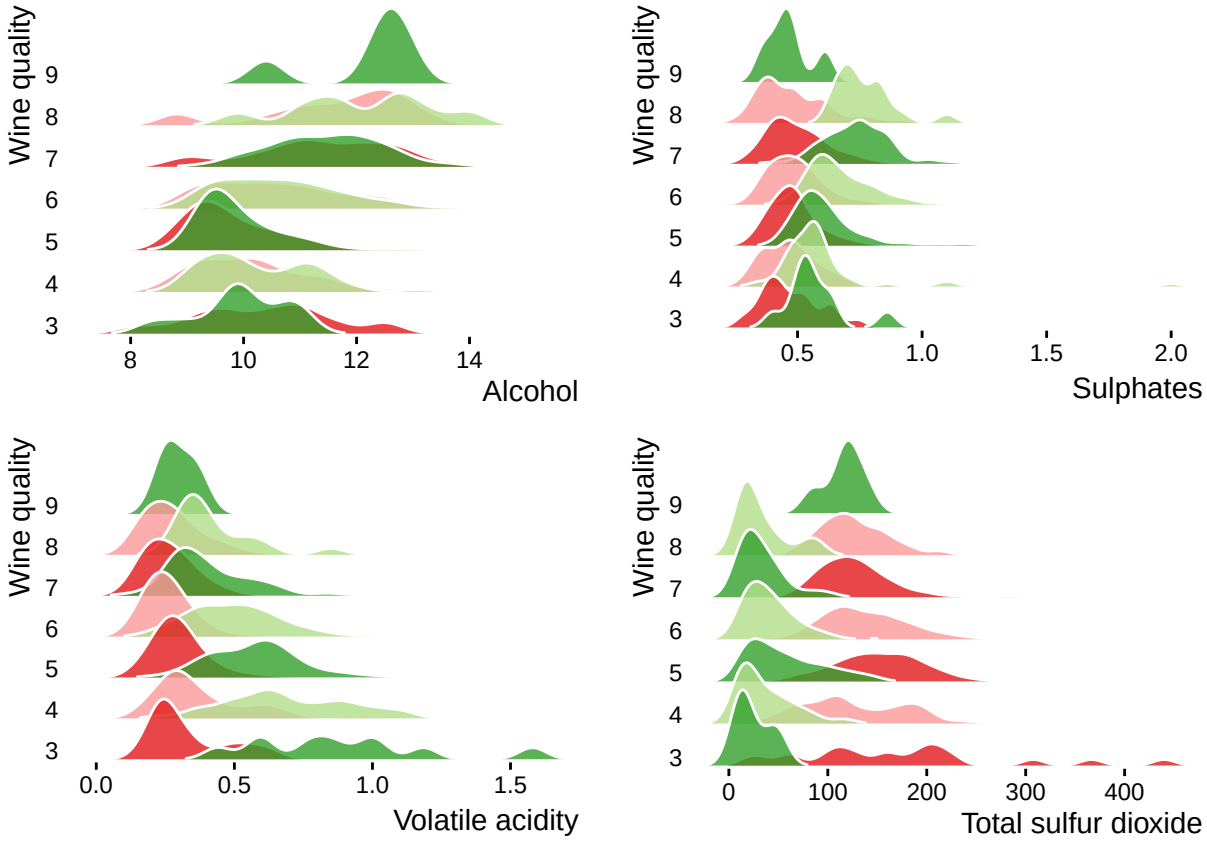


Figure 5: Density Plots of Select Variables by Quality (Log-Transformed)

wines. The white wines seem to have more stable distributions across ratings than the red wines, mostly bunched up into a small range with just a few outliers. For the red wines, very high volatile acidity seems to be an indication that the wine is of poor quality. We also observe that highly rated red wines tend to have more sulphates. Finally, the strongest indicator of wine quality seems to be alcohol, where quality tends to increase with alcohol percentage for both red and white wines alike.

5 Results

5.1 Results of the predictive benchmarking experiment

5.1.1 Regression task

The results of the predictive benchmarking experiment for the regression task can be seen in Table 2. The random forest algorithm is the best performer on both metrics under consideration for the regression task, achieving a mean squared error of 0.344 and a mean absolute error of 0.411. For comparison, the dummy regressor achieved a mean squared error of 0.75 and a mean absolute error of 0.677. On the MSE metric, the SVM is the next best performer, followed by the linear model, the MLP, and finally the dummy regressor.

Table 2: Benchmarking results - regression task with all unscaled variables

Model	Metric	Mean	LB	UB
Dummy	MSE	0.7498	0.6930	0.8134
Dummy	MAE	0.6766	0.6463	0.7056
Linear	MSE	0.5203	0.4709	0.5730
Linear	MAE	0.5538	0.5302	0.5783
SVM	MSE	0.4793	0.4323	0.5274
SVM	MAE	0.5205	0.4970	0.5448
RF	MSE	0.3443	0.3038	0.3899
RF	MAE	0.4111	0.3895	0.4338
MLP	MSE	0.5236	0.4755	0.5755
MLP	MAE	0.5625	0.5387	0.5871

Figure 6 plots these results for each algorithm.

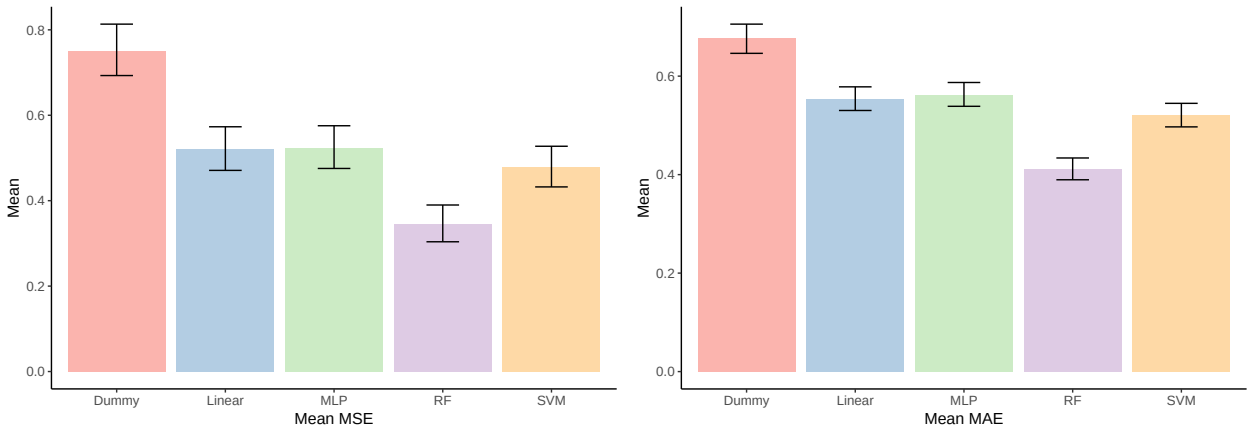


Figure 6: Benchmarking results - regression task with all variables

The performance of the algorithms on the same task but with a reduced set of predictor variables can be seen in Figure 7. The left-hand plot shows model performance (mean squared error) with wine colour as the only predictor. The right-hand plot shows results for physiochemical predictors only. Interestingly, all five algorithms have an almost identical, large, MSE (0.749) for the wine colour only model, suggesting that wine colour alone is not a strong predictor of quality. In contrast, replacing wine colour with the physiochemical characteristics of a wine leads to differential performance across algorithms, and an average MSE of 0.468 for all algorithms excluding the dummy regressor, which had an unchanged performance.

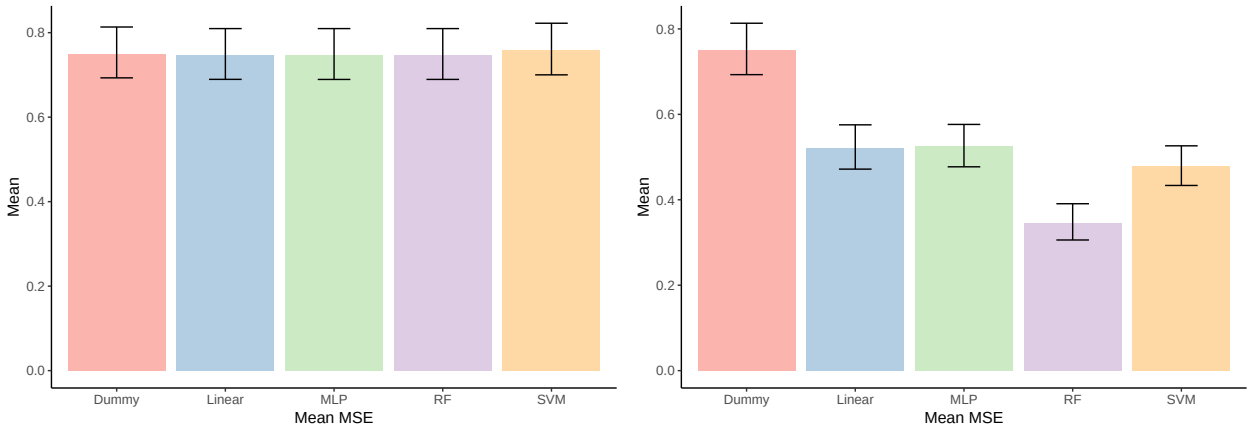


Figure 7: Benchmarking results - regression task with wine colour only (left) and physiochemical variables only (right)

5.1.2 Classification task

The results of the predictive benchmarking experiment for the classification task can be seen in Table 3. As with the regression task, the random forest model was again the best performer, achieving classification accuracy of 0.702 and a mean absolute error 0.334. For comparison, the dummy regressor achieved a classification accuracy of 0.33 and a mean absolute error 0.92. The ranking of algorithms by classification accuracy was the same as for the regression task, with the SVM following the RF, then the linear model, the MLP, and the dummy regressor in fifth position.

Table 3: Benchmarking results - classification task with all unscaled variables

Model	Metric	Mean	LB	UB
Dummy	MAE	0.9198	0.8746	0.9623
Dummy	Accuracy	0.3300	0.3054	0.3554
Linear	MAE	0.4850	0.4531	0.5169
Linear	Accuracy	0.5697	0.5415	0.5962
SVM	MAE	0.4458	0.4138	0.4785
SVM	Accuracy	0.6051	0.5785	0.6316
RF	MAE	0.3344	0.3054	0.3646
RF	Accuracy	0.7017	0.6769	0.7254
MLP	MAE	0.4972	0.4669	0.5292
MLP	Accuracy	0.5506	0.5238	0.5762

Figure 8 plots these results for each algorithm.

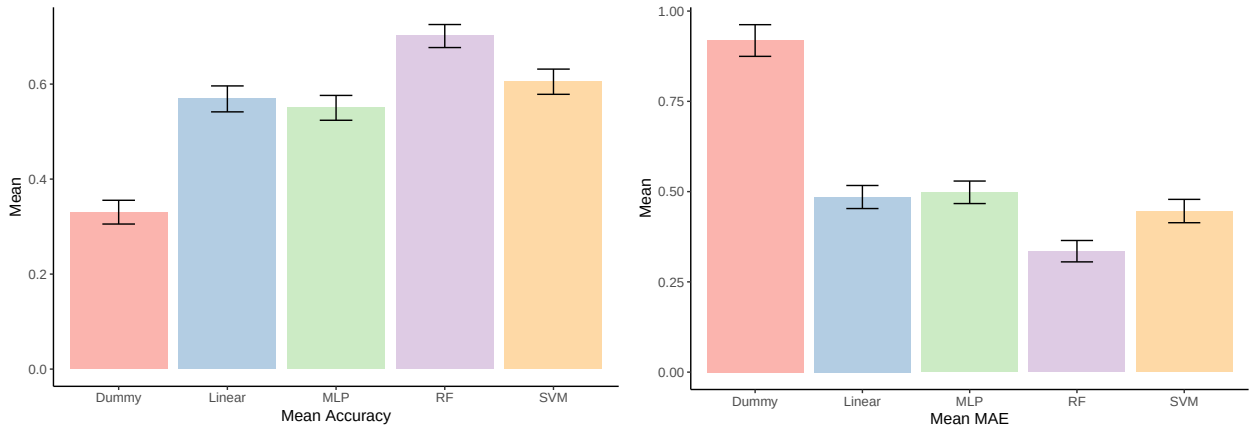


Figure 8: Benchmarking results - classification task with all variables

The performance of the algorithms on the same task but with a reduced set of predictor variables can be seen in Figure 9. The left-hand plot shows model accuracy with wine colour as the only predictor. The right-hand plot shows results for physiochemical predictors only. Interestingly, all algorithms except the dummy regressor achieve identical classification accuracy (0.442) when using wine colour as the only model feature. As with the regression task, replacing wine colour with the physiochemical characteristics of a wine leads to differential performance across algorithms, and an average MSE of 0.599 for all algorithms excluding the dummy regressor, which was again unchanged.

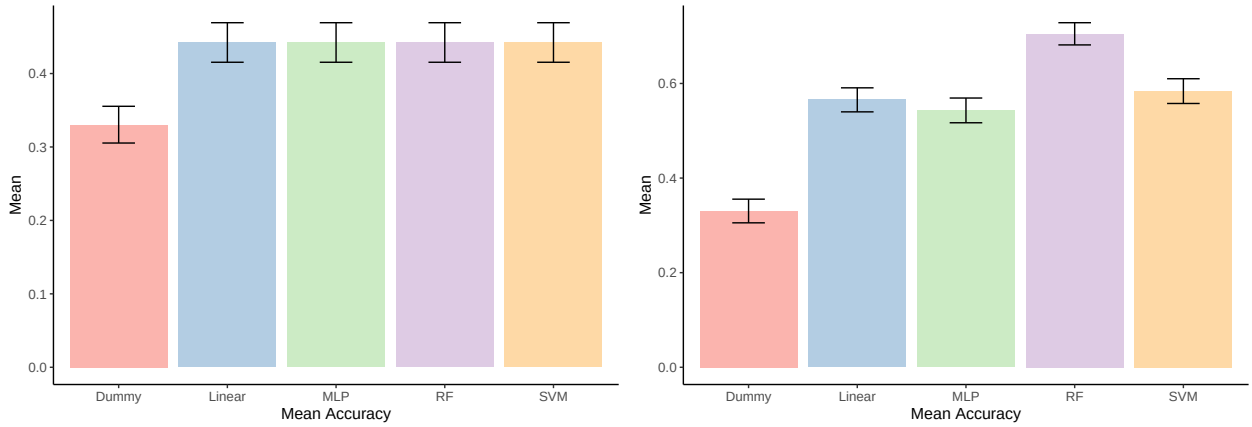


Figure 9: Benchmarking results - classification task with wine colour only (left) and physiochemical variables only (right)

5.1.3 Error Analysis

In figure 10, we consider how our best classifier, the random forest model, performs on the test data. We note that when it does make mistakes, they tend to be in the region around the true value, indicating a strong model. On the other hand, we note that the model is reluctant to predict outside of the most common values. Although this accords with the distribution of the data itself, we believe that a more balanced dataset would enable our model to perform more robustly.

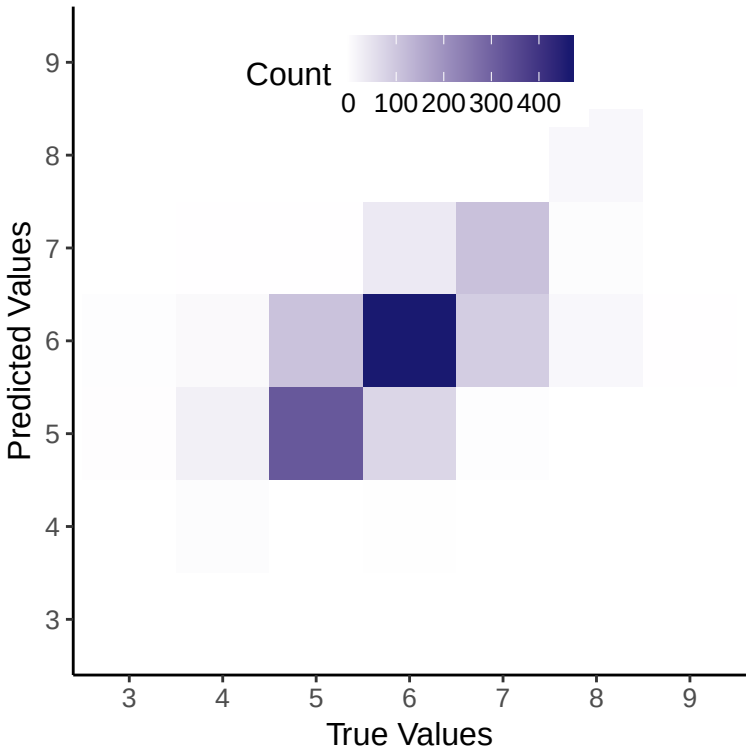


Figure 10: Random Forest classifier confusion matrix

5.2 What makes a wine a good wine

In order to understand what makes a wine a good wine, we ran a simple linear regression model. All physiochemical variables were transformed to the log scale to capture the law of mass action being linear in log-concentration coordinates. Variables were additionally standardised to facilitate the interpretation of coefficients.

Table 4: Linear model - logged and scaled training data

Variable	**Estimate**	**SE**	**Statistic**	**p-value**
Intercept	5.767	0.018	320.397	0.000
Fixed acidity	0.050	0.019	2.663	0.008
Volatile acidity	-0.246	0.015	-16.344	0.000
Citric acid	0.001	0.013	0.111	0.911
Residual sugar	0.179	0.022	8.014	0.000
Chlorides	-0.030	0.013	-2.258	0.024
Free sulfur dioxide	0.164	0.017	9.687	0.000
Total sulfur dioxide	-0.135	0.023	-5.862	0.000
Density	-0.147	0.034	-4.363	0.000
pH	0.036	0.015	2.486	0.013
Sulphates	0.099	0.012	7.929	0.000
Alcohol	0.336	0.020	16.919	0.000
Color	0.229	0.061	3.757	0.000

Table 4 shows how each feature was associated with wine quality. With all other variables held constant, the mean wine quality score was 5.8. All variables were significantly associated with wine quality except for citric acid. In terms of features associated with a good wine, we found relatively larger, positive coefficients from the alcohol volume, colour (red), and residual sugar measures. Notably negative coefficients were present for volatile acidity (the mass concentration of acetic acid, g/dm^3), density, and mass concentration of total sulfur dioxide (mg/dm^3). This suggested that wines with higher alcohol volume, higher residual sugar, and red wines typically had slightly higher quality scores, whilst wines with higher volatile acidity, higher density, and a greater concentration of total sulfur dioxide typically had slightly lower quality scores, holding all else constant. Whether these features completely capture what makes a wine a "good" wine is debatable however, especially in the simple linear context.

Table 5: Relative random forest feature importance in classifying wine quality

Feature	**Mean**	**SD**
Fixed acidity	0.066	0.009
Volatile acidity	0.107	0.022
Citric acid	0.077	0.015
Residual sugar	0.073	0.010
Chlorides	0.087	0.029
Free sulfur dioxide	0.083	0.011
Total sulfur dioxide	0.083	0.015
Density	0.103	0.039
pH	0.071	0.009
Sulphates	0.076	0.011
Alcohol	0.169	0.057
Color	0.005	0.005

Additional evidence comes from looking at the relative importance of features identified by the random forest algorithm. Table 5 shows these relative importances with regards to each feature's use in predicting wine quality. As can be seen, alcohol(vol%) is the most important feature, with a mean importance of 0.169, followed by volatile acidity i.e. mass concentration of acetic acid (0.107) and density (0.103). Fixed acidity (0.066) and colour (0.005) are the least useful features, suggesting wines of both colours and across the range of mass concentrations of tartaric acid can be "good" wines (and "bad" wines).

5.3 The search for the perfect wine

By creating mixtures of chemicals (and color) – limited only to the ranges observed in the test data – we attempt to simulate a wine that will achieve a perfect "10" rating by our models' estimates (emphasizing that such a wine may be impossible to create in the physical world, is likely missing other key components, and is "perfect" only in the eyes of the trained models for which it is optimized).

We use a support vector machine (SVM) and linear regression model to produce two possible "super-wines", since they are both capable of predicting values beyond the range of data used in training. For the linear regression model, we maximize the values of each feature known to correspond to high quality wines, and minimize those associated with low-quality wines. For the SVM, we can randomly sample values for each predictor (within the range of the training data) and feed the artificial wine into the SVM, stopping when we achieve a rating above 10.

Unfortunately, these "superwines" fail to achieve 10 ratings across all models, which is to be expected, since they were optimized for these particular models. However, they do indicate certain trends. As seen in Table 6, they tend to have higher levels of free sulfur dioxide but lower levers of total sulfur dioxide than the average wine in the data, and have increased levels of chlorides and residual sugar.

Table 6: Simulated Wine Characteristics

Variable	Mean	Linear	SVM
fixed acidity	7.22	16.60	9.85
volatile acidity	0.34	1.08	1.25
citric acid	0.32	2.00	1.15
residual sugar	5.44	23.00	19.91
chlorides	0.06	1.01	1.02
free sulfur dioxide	30.53	139.50	89.72
total sulfur dioxide	115.74	7.00	8.38
density	0.99	0.99	1.00
pH	3.22	3.80	3.62
sulphates	0.53	2.95	2.70
alcohol	10.49	14.00	10.29
color	25% Red	Red	White

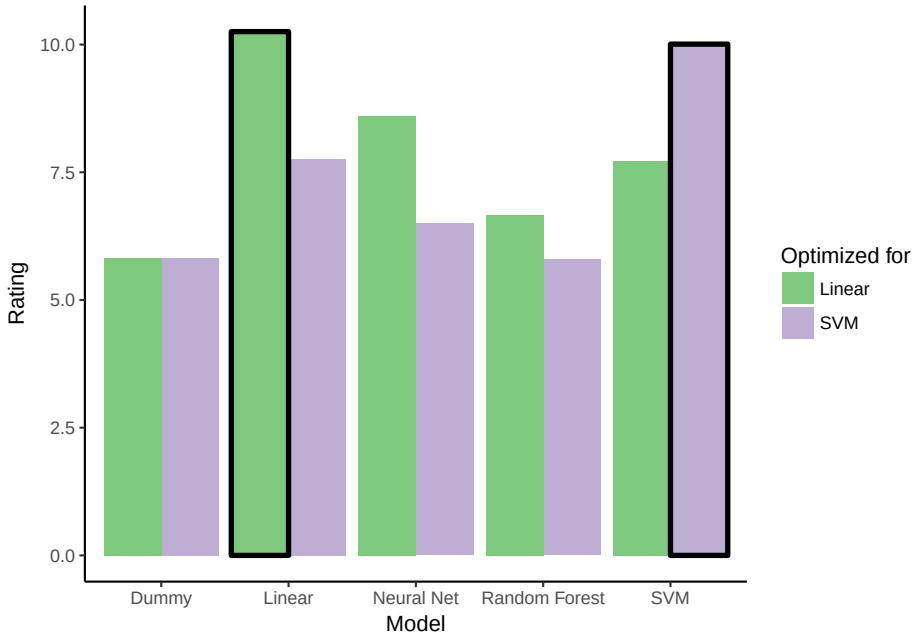


Figure 11: Simulated Wine Performance by Model

5.4 A 'super-human' AI

Although our models are likely to rate wines with greater consistency than sommeliers – and perhaps make more accurate predictions on objective metrics, like provenance – without more information we are unable to determine with certainty if Delicious AI can outperform humans' predictive performance on any given metric. Lacking data on measurable human performance, we cannot speak to a strictly "superhuman" capacity.

However, our models are, on average, able to rate wines to within a close margin of sommelier rankings using only a limited subset of information (e.g., the models cannot assess the aroma directly). Moreover, some of these characteristics are present early in the wine making process, perhaps well before an expert could make a judgement of quality (although here too we would need more information to make such a determination).

Namely, a wine's color, fixed acidity, citric acid, residual sugar, and chlorides are all largely determined by grape characteristics rather than effects of the fermentation process. Limiting ourselves to these components alone, our best model achieves a mean squared error of 0.49, (C.I. of [0.44, 0.54]), which is worse than the best model trained on all variables (0.34, C.I. of [0.3, 0.39]), but nonetheless represents an improvement over the naive baseline, performing on par with the linear model that uses all data. Obtaining such an early estimate could help guide the manufacturing process, alerting winemakers to potential problems with a batch.

Although human winemakers indeed use similar variables to select harvest times, it is within the realm of possibility that Delicious AI could make judgements of resulting wine quality with "super-human" prescience and accuracy.

5.5 Future research

As it stands, Delicious AI has been trained to match the wine experts' taste. It may make more commercial sense, however, to train an AI to match the taste of Delicious AI's customer base, or even the taste of the general public. This could be done by carrying out another wine tasting experiment with ordinary members of the public, perhaps following the protocols of a randomised control trial, to see how perceptions of wine quality differ (both between experts and ordinary people, and between the AIs). Alternatively, other measures of "success" could be used instead of subjective wine quality, such as volume sold, profit made, or wine prizes won. Delicious AI could be trained to predict these outcomes instead of, or in addition to, expert judgement of wine quality.

Another obvious direction for future research is the linking of the human experts' wine quality ratings with the potential outputs from taste sensors developed in the "AI gustometer" project. This would provide the AI sommelier with an alternative "taste" to learn from. If the focus remains on human experts' quality ratings, the taste sensor data could instead be used to augment the current training dataset as additional input variables to a model.

A Appendix A

A.1 Data

We used the Wine Quality Dataset for this project. This systematic data on 1599 red and 4898 white *vinho verde* quality wines from Portugal was acquired specifically to help the AI sommelier acquire its taste for wine. Wine quality was measured using the median sensory preference for a wine from up to three sensory assessors. These sensory preferences were decided following blind sensory assessment on a subjective scale of 0 (disgusting) to 10 (excellent). In the absence of a truly objective measure of wine quality, we used this subjective scale in order to train the AI sommelier towards an "expert's taste" for wine. We used the extensive physiochemical data on each wine as features in our models. Laboratory analysis of the wine provided the following 11 variables (in addition the wine quality measure):

1. fixed acidity = mass concentration of tartaric acid (g/dm^3)
2. volatile acidity = mass concentration of acetic acid (g/dm^3)
3. mass concentration of citric acid (g/dm^3)
4. residual sugar mass concentration (g/dm^3)
5. mass concentration of sodium chloride (g/dm^3)
6. mass concentration of free sulfur dioxide (mg/dm^3)
7. mass concentration of sulfur dioxide total (mg/dm^3)
8. density (g/dm^3)
9. pH value
10. mass concentration of potassium sulphate (mg/dm^3)
11. alcohol content (vol%)

Summary statistics and further descriptive, exploratory analysis can be seen in the next section.

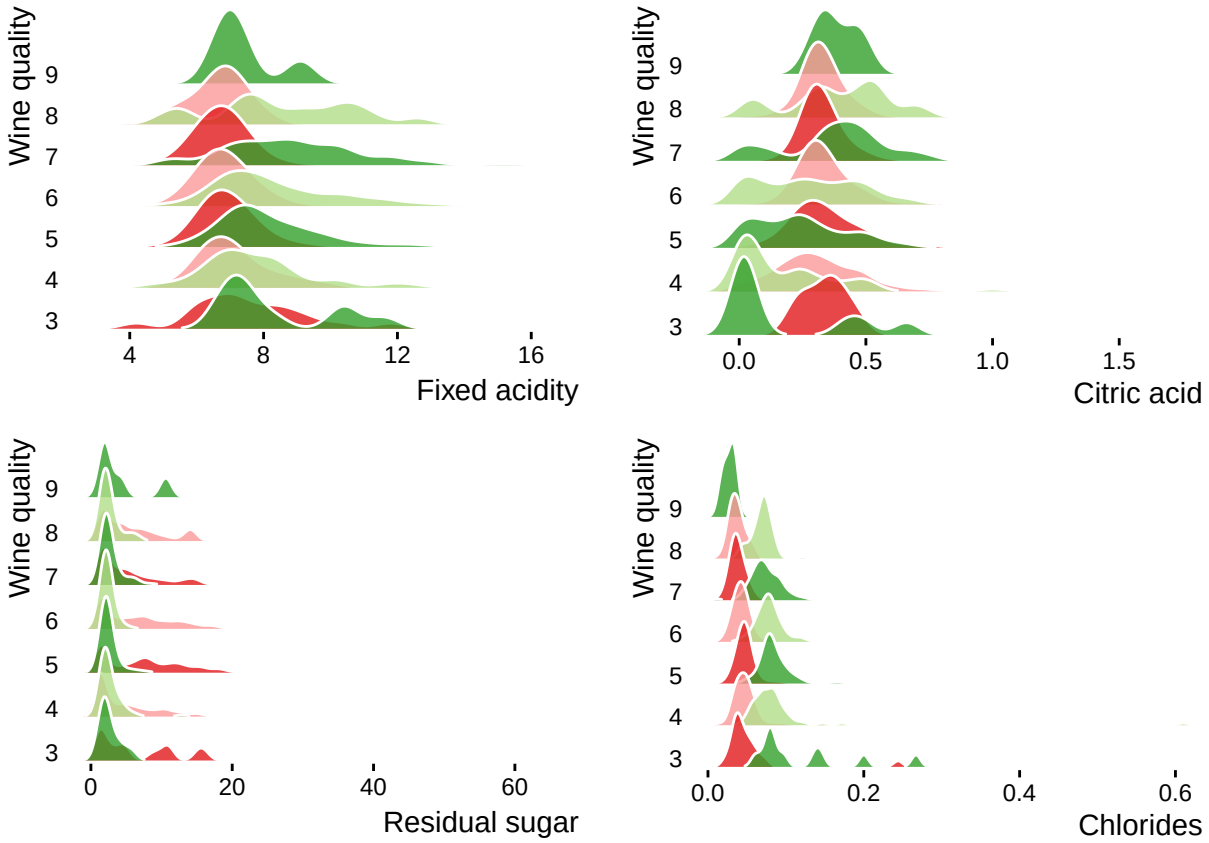
A.2 Tuned Hyper-Parameters

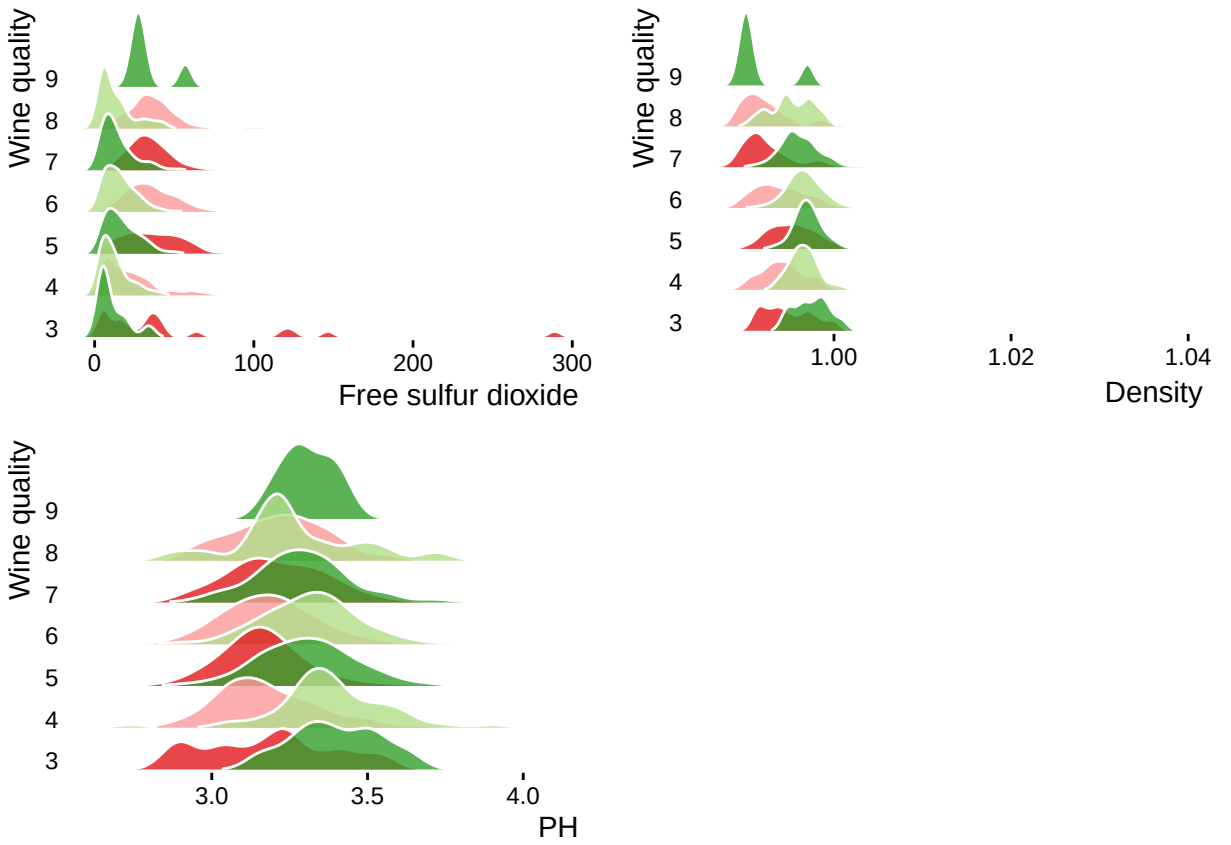
The following hyper-parameters were optimised or pre-selected for each learner and each task (where relevant), with all other parameters set to their defaults. The bounds were determined by reading a variety of online resources.

- Linear/logistic regression:
  - Penalty: L1 or L2
  - C: sampled from an exponential distribution with $\lambda = 0.1$
- SVM:
  - C: sampled from an exponential distribution with $\lambda = 0.1$
  - Gamma: sampled from an exponential distribution with $\lambda = 10$
  - Kernel: Radial Basis Function (pre-selected)
- Random Forest:
  - Number of trees: between 10 and 100
  - Number of features: between 1 and 11
  - Maximum tree depth: None (pre-selected)
- Multilayer Perceptron:
  - Activation functions: ReLU or Sigmoid
  - Hidden layer size/number of neurons in a hidden layer: 32, 64 or 128
  - Learning rate: sampled from a normal distribution with $\mu = 0.001$, $\sigma = 0.0002$
  - Maximum iterations: 500 (pre-selected)

A.3 Remaining Ridge Plots

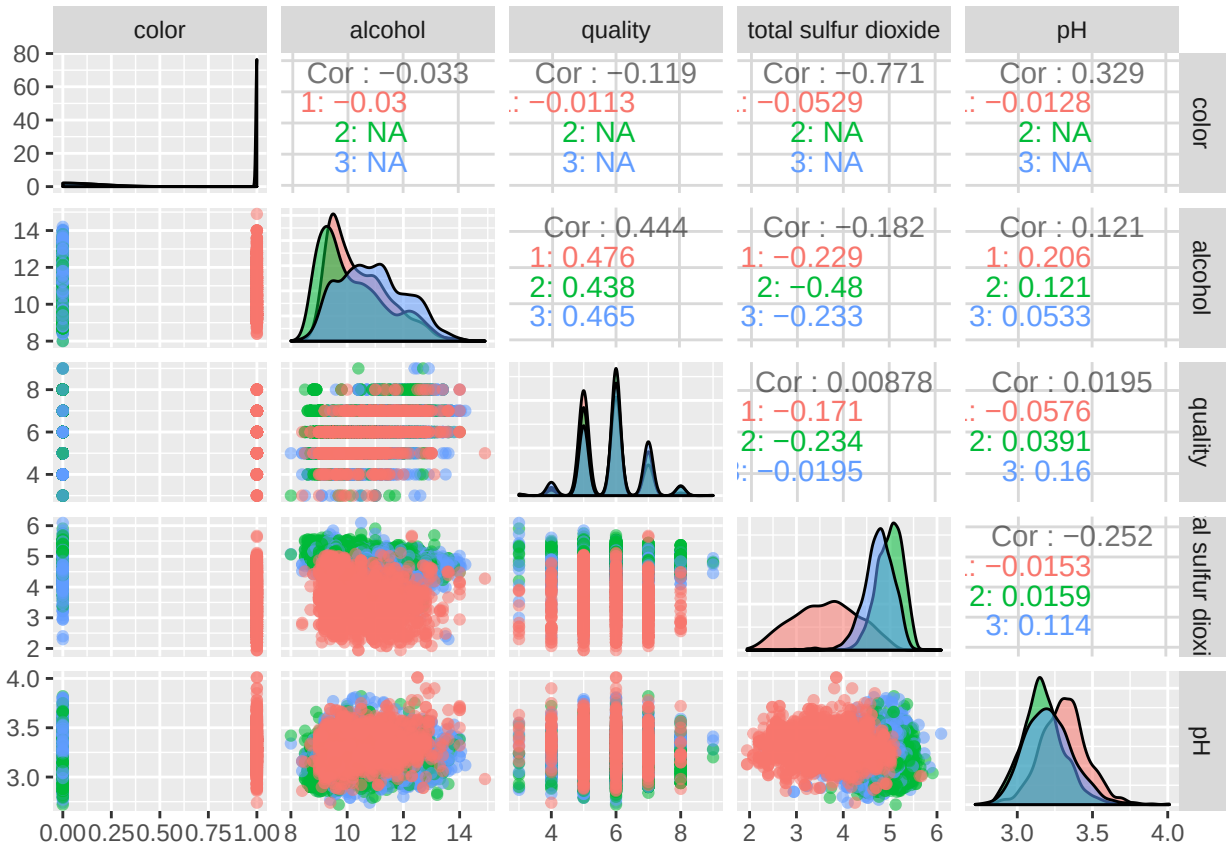
Here we include ridge plots of the remaining variables for completeness.

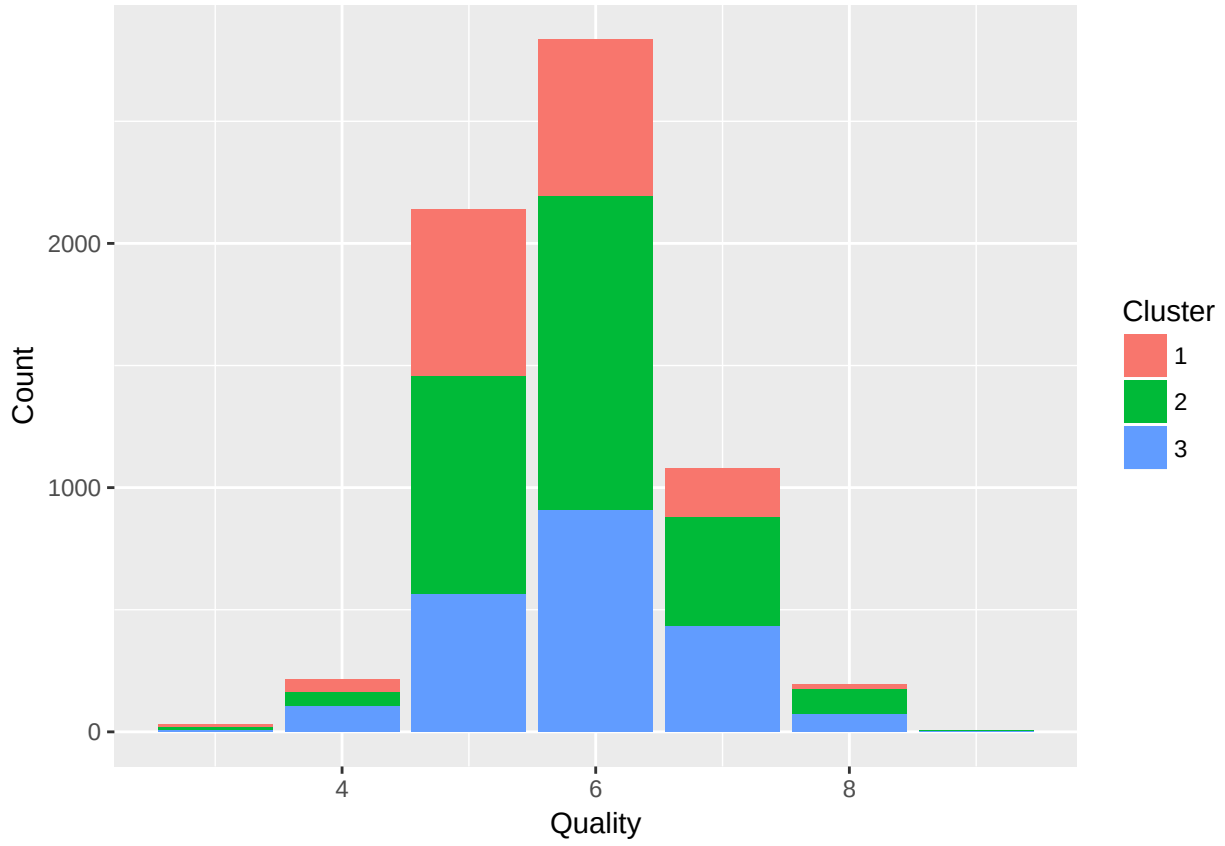

Wine quality
9
8
7
6
5
4
3
4
8
12
16
Fixed acidity
Wine quality
9
8
7
6
5
4
3
0.0
0.5
1.0
1.5
Citric acid
Wine quality
9
8
7
6
5
4
3
0
20
40
60
Residual sugar
Wine quality
9
8
7
6
5
4
3
0.0
0.2
0.4
0.6
Chlorides



A.4 Clustering

We also attempted to cluster the data with a Gaussian Mixture Model and observed that the clusters do not divide up quality in any meaningful way.


color
alcohol
quality
total sulfur dioxide
pH
Cor : −0.033
1: −0.03
2: NA
3: NA
Cor : −0.119
: −0.0113
2: NA
3: NA
Cor : −0.771
: −0.0529
2: NA
3: NA
Cor : 0.329
: −0.0128
2: NA
3: NA
Cor : 0.444
1: 0.476
2: 0.438
3: 0.465
Cor : −0.182
1: −0.229
2: −0.48
3: −0.233
Cor : 0.121
1: 0.206
2: 0.121
3: 0.0533
Cor : 0.00878
1: −0.171
2: −0.234
: −0.0195
Cor : 0.0195
: −0.0576
2: 0.0391
3: 0.16
Cor : −0.252
: −0.0153
2: 0.0159
3: 0.114
color
alcohol
quality
al sulfur dioxi
pH



B Appendix B

B.1 Code

Note that the code is a mixture of R and Python. Python is noted when used.

B.1.1 Setup

Knitr Setup

```
knitr::opts_chunk$set(echo = FALSE, #cache = TRUE, cache.lazy = FALSE,
                      fig.pos = "H")
```

Load R packages

```
# use pacman for package management
if(!require("pacman")) {
    install.packages("pacman")
}

# load the tidyverse, joyplots (now in ggridges) for plots,
# magrittr for pipes, summarytools for descriptives
pacman::p_load(
  tidyverse, ggridges, magrittr, summarytools,
```

```
  GGally, mclust, feather, reticulate, purrr, here, stringr,
  kableExtra, broom, knitr)
```

Load data

```
# read in the csv files
wine <-
  bind_rows(
    # read in white, encode as 0
    read_delim('./input/winequality-white.csv', delim=';', guess_max=10000) %>%
      mutate(color=0L),
    # read in red, encode as 1
    read_delim('./input/winequality-red.csv', delim=';', guess_max=10000) %>%
      mutate(color=1L)
  )
```

In order to avoid fighting with factors, we make a function to move from hard-coded numbers to colors.

```
recode_color_ <- function(x) dplyr::recode(x, `1` = 'Red', `0` = 'White')
recode_color <- function(x){
  # used to avoid factors, which can be capricious

  switch(
    is.data.frame(x) + 1,
    # if not a dataframe, directly recode
    recode_color_(x),
    # if a dataframe, overwrite color column (presumed to exist)
    dplyr::mutate(x, color=recode_color_(color))
  )
}
```

B.1.2 Data exploration

Create table of summary statistics.

```
q025 <- function(x) quantile(x, 0.25)
q075 <- function(x) quantile(x, 0.75)

wine_summary <-
  wine %>%

  # summary statistics by color
  # perhaps a little cleaner to look overall?
  # group_by(color) %>%
  summarize_all(
    funs(min, q025, median,  mean, q075, max, sd)
  ) %>%

  # transform data
  # gather(... = -color) %>%
  gather %>%
  separate(key, into = c('Variable', 'statistic'), sep = '_') %>%
  spread(key = statistic, value = value)

# order variables with highest variation (after rescaling)
```

```
variable_var <-
  wine %>%
  #select(#-color,
         #-quality) %>%
  # min-max scale the data
  mutate_all(funs(. / (max(.) - min(.)))) %>%
  # take variances
  summarize_all(var) %>%
  # reshape and sort
  gather %>%
  arrange(-value)

# make beautiful
wine_summary %>%
  # recode_color %>%
  select(
    Variable,
    # Color=color,
    Min = min,
    `1st Quantile` = q025,
    Median = median,
    `3rd Quantile` = q075,
    Max = max,
    Mean = mean,
    `S.D.`=sd
  ) %>%
  arrange(Variable) %>%
  kable(format = "latex",
        booktabs = T,
        digits = 2,
        caption = "Summary of Wine Data") %>%
  kableExtra::row_spec(0, bold = TRUE) %>%
  kableExtra::kable_styling(latex_options = "hold_position") %>%
  kableExtra::add_header_above(c(" " = 1, "Range" = 5, "Moments" = 2))
```

Create histograms.

```
# plot histograms (unscaled)
wine %>%
  select(chlorides, `residual sugar`, sulphates) %>%
  #mutate_at(vars(-alcohol, -pH, -quality), log) %>%
  gather(key='Variable', value='Value') %>%
  ggplot(aes(x = Value)) +
  geom_histogram(bins = 50) +
  facet_wrap(~Variable, scale='free')
```

Create boxplots to compare log transformed data with original concentration values.

```
wine %>%
  # Transform data
  recode_color %>%
  select(`Original Value` = chlorides, Color=color) %>%
  mutate(`With Log Transformation` = log(`Original Value`)) %>%
  gather(key = 'key', value = 'Chlorides (g/dm^3)',
         `Original Value`, `With Log Transformation`) %>%
```

```
  # Plot
  ggplot(aes(x = Color, y = `Chlorides (g/dm^3)`, fill=Color)) +
  geom_boxplot(alpha=0.5) +
  scale_fill_manual(values=c("Red"="#FB9A99", "White" = "#B2DF8A"),
                    guide=FALSE) +
  facet_wrap(~key, scales='free_y')  +
  theme(text = element_text(size = 12),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        axis.line = element_line(color = 'black'))
```

Plot a correlation heatmap of the variables.

```
wine_cor <-
  wine %>%
  select(-quality, -color) %>%
  mutate_all(log1p) %>%
  # get correlations
  cor

wine_cor[upper.tri(wine_cor, diag = TRUE)] <- NA

wine_cor %>%
  reshape2::melt() %>%
  # plot
  ggplot(aes(x=Var1, y=Var2, fill=value)) +
  geom_tile() +
  scale_fill_gradient2(name = 'Pearson Corr.',
                       limit=c(-1, 1)) +
  # geom_text(aes(label=round(value, 2))) +
  theme(axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1)) +
  coord_fixed()
```

Log-transform the data.

```
# convert all concentration variables to the log scale
origwine <- wine

concvars <- c(
  "fixed acidity",
  "volatile acidity",
  "citric acid",
  "residual sugar",
  "chlorides",
  "free sulfur dioxide",
  "total sulfur dioxide",
  "sulphates"
)

wine <-
    origwine %>%
    mutate_at(.vars = vars(concvars),
```

```
            .funs = log1p)
    # scale the data (can turn off later)
    #mutate_at(vars(-color, -quality), funs(scale(.) %>% as.numeric))
```

Make a pairs-plot of the log-transformed data.

```
# pairs plot of the n variables with the greatest variance
n <- 5
p <-
  wine %>%
  recode_color %>%
  ggpairs(
    mapping=aes(color=color, alpha=0.4),
    columns=c('free sulfur dioxide',
              'total sulfur dioxide',
              'residual sugar',
              'density',
              'alcohol')
    # if we want n variables with greatest variance, change to
    # variable_var$key[1:n]
  )
for(i in 1:p$nrow) {
  for(j in 1:p$ncol){
    p[i,j] <- p[i,j] +
      scale_color_manual(values = c("Red" = "#E31A1C", "White" = "#33A02C")) +
      scale_fill_manual(values = c("Red" = "#E31A1C", "White" = "#33A02C"))
  }
}

p
```

Generate ridge plots of a few select variables.

```
# ridge plot function
ridgeplot <- function(winevar) {
    origwine %>%
    recode_color %>%
    ggplot(aes(y = factor(quality))) +
    geom_density_ridges(aes_q(x = as.name(winevar),
                              fill = paste(origwine$quality, origwine$color)),
                        alpha = .8,
                        color = "white",
                        rel_min_height = 0.01) +
    labs(x = Hmisc::capitalize(gsub(x = winevar,
                                    pattern = "\\.",
                                    replacement = " ")),
         y = "Wine quality") +
    scale_y_discrete(expand = c(0.01, 0)) +
    scale_x_continuous(expand = c(0.01, 0)) +
    scale_fill_cyclical(breaks = c("3 red", "3 white"),
                        labels = c(`3 red` = "Red", `3 white` = "White"),
                        values = c("3 red" = "#E31A1C",
                                   "3 white" = "#33A02C",
                                   "4 red" = "#FB9A99",
                                   "4 white" = "#B2DF8A",
```

```
                                    "5 red" = "#E31A1C",
                                    "5 white" = "#33A02C",
                                    "6 red" = "#FB9A99",
                                    "6 white" = "#B2DF8A",
                                    "7 red" = "#E31A1C",
                                    "7 white" = "#33A02C",
                                    "8 red" = "#FB9A99",
                                    "8 white" = "#B2DF8A",
                                    "9 white" = "#33A02C"),
                        name = "Wine colour",
                        guide = "legend") +
    theme_ridges(grid = FALSE, font_size = 11)
}

# make ridgeplots
ridgeplotlist <- lapply(c('alcohol', 'sulphates', 'volatile acidity', 'total sulfur dioxide'), ridgeplo
ridgeplots_grid <- ggpubr::ggarrange(plotlist = ridgeplotlist, ncol = 2, nrow = 2)
ridgeplots_grid
```

Save the data for training.

```
# save the data in binary format for python interoperability
write_feather(wine, './intermediate/wine_logged_scaled.feather')
origwine %>%
  mutate_at(.vars = vars(concvars),
            .funs = log1p) %>%
  write_feather('./intermediate/wine_logged_unscaled.feather')
```

B.1.3 Model Training and Evaluation

We move to python to learn our models.

```
import pickle
import feather
import scipy
import numpy as np
import pandas as pd
from sklearn.dummy import DummyRegressor, DummyClassifier
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.svm import SVR, SVC
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.neural_network import MLPRegressor, MLPClassifier
from sklearn.model_selection import (
  train_test_split, RandomizedSearchCV, KFold, StratifiedKFold)
from sklearn.metrics import (
  mean_squared_error, mean_absolute_error, f1_score, make_scorer)
from sklearn.utils import resample
TRAIN_MODELS = False # set to true to train our models
```

Setup functions to train models.

```
def gen_data(data_fpath, test_size=0.2, random_state=1234,
             features_to_drop=[], features_to_keep=[]):
  """
  Generate train and test data, optionally dropping features
```

```
  """
  # read in data
  wine = feather.read_dataframe(data_fpath)
  # split into x and y data
  x = wine.drop(['quality'] + features_to_drop, axis=1)
  if features_to_keep:
      x = x[features_to_keep]
  y = wine.quality
  # split into train and test
  x_train, x_test, y_train, y_test = train_test_split(
      x, y, test_size=test_size, random_state=random_state)
  return x_train, x_test, y_train, y_test
def config_cv(learner, scoring=None, params=None,
              n_iter=300, folds=3, n_jobs=8, **kwargs):
    """
    Pre-configure the RandomizedSearchCV
    """
    if isinstance(learner, (LogisticRegression, SVC, RandomForestClassifier)):
      params = {**params, 'class_weight': ['balanced', None]}
    if isinstance(learner, LinearRegression):
      params = {}
    if not params:
      n_iter = 1

    learner = RandomizedSearchCV(
      learner,
      params,
      scoring=scoring,
      n_iter=n_iter,
      cv=folds,
      n_jobs=n_jobs,
      **kwargs
    )
    return learner
def train(model_dir, x_train, y_train):
  """
  Train models
  """
  obs, features = x_train.shape
  # global configurations
  mse_scorer = make_scorer(mean_squared_error, greater_is_better=False)
  mae_scorer = make_scorer(mean_absolute_error, greater_is_better=False)
  f1_scorer = make_scorer(f1_score, labels=[3,4,5,6,7,8,9], average='micro')
  # build configurations for each of the learners
  params = [
    ('dummy', {}),
    ('linear', {
        'penalty': ['l1', 'l2'],
        'C': scipy.stats.expon(scale=100)
      }
    ),
    ('svm', {
        'C': scipy.stats.expon(scale=100),
```

```
        'gamma': scipy.stats.expon(scale=.1),
        'kernel': ['rbf']
      }
    ),
    ('rf', {
        'n_estimators': scipy.stats.randint(low=10, high=100),
        'max_features': scipy.stats.randint(low=2, high=features) if features > 1 else [1],
        'max_depth': [None] # consider making this random as well
      }
    ),
    ('mlp', {
        'activation': ['relu', 'logistic'],
        'hidden_layer_sizes': [(32, 32), (64, 64), (128, 128)],
        'learning_rate_init': scipy.stats.norm(loc=0.001, scale=0.0002),
        'max_iter': [500],
      }
    )
]
# create regressors
regressors = [
  DummyRegressor(),
  LinearRegression(),
  SVR(),
  RandomForestRegressor(),
  MLPRegressor()
]
scorers_reg = {'mse': mse_scorer, 'mae': mae_scorer}
regressors = {
  name: config_cv(learner, scorers_reg, p, refit='mse')
  for learner, (name, p) in zip(regressors, params)
}
# create classifiers
classifiers = [
  DummyClassifier(),
  LogisticRegression(),
  SVC(),
  RandomForestClassifier(),
  MLPClassifier()
 ]
scorers_clf = {'mse': mse_scorer, 'mae': mae_scorer, 'f1': f1_scorer}
classifiers = {
  name: config_cv(learner, scorers_clf, p, refit='f1')
  for learner, (name, p) in zip(classifiers, params)
}
# perform training
np.random.seed(12345)
for name, reg in regressors.items():
    print('On {} regressor'.format(name))
    reg.fit(x_train, y_train)
    with open('./intermediate/{}/reg_{}.pkl'.format(model_dir, name), 'wb') as o:
        pickle.dump(reg, o)
np.random.seed(12345)
for name, clf in classifiers. items():
```

```
      print('On {} classifier'.format(name))
      clf.fit(x_train, y_train)
      with open('./intermediate/{}/clf_{}.pkl'.format(model_dir, name), 'wb') as o:
          pickle.dump(clf, o)
```

Setup functions to evaluate models.

```
def load_models(model_dir):
  """
  Load saved models
  """
  model_names = ['dummy', 'linear', 'svm', 'rf', 'mlp']
  regressors, classifiers = {}, {}
  for name in model_names:
    with open('./intermediate/{}/reg_{}.pkl'.format(model_dir, name), 'rb') as i:
      regressors[name] = pickle.load(i)
    with open('./intermediate/{}/clf_{}.pkl'.format(model_dir, name), 'rb') as i:
      classifiers[name] = pickle.load(i)
  return regressors, classifiers
def test_err_bootstrap(model, metric, x_test, y_test,
                       n_iter=1000, alpha=0.05, seed=1234):
  """
  Calculate mean and 100*(1-alpha)% central confidence intervals for a metric
  on a bootstrapped dataset
  """
  np.random.seed(seed)
  results = []
  for _ in range(n_iter):
    # by default, samples the same number in array, with replacement
    x_test_bs, y_test_bs = resample(x_test, y_test)
    y_pred_bs = model.predict(x_test_bs)
    results.append(metric(y_test_bs, y_pred_bs))

  a = 100 * alpha / 2
  lb, ub = np.percentile(results, (a, 100 - a))
  mean = np.mean(results)
  return mean, lb, ub
def evaluate_models(model_dir, x_test, y_test, **kwargs):
  """
  Evaluate the models and save the results
  """
  all_models = load_models(model_dir)
  results = []
  metrics = {
    'reg': [
      ('mse', mean_squared_error),
      ('mae', mean_absolute_error),
    ],
    'clf': [
      ('mse', mean_squared_error),
      ('mae', mean_absolute_error),
      ('f1_score',
        lambda y_true, y_pred: f1_score(
          y_true, y_pred, labels=[3,4,5,6,7,8,9], average='micro'
```

```
        )
      )
    ]
  }
  # iterate through models
  for model_type, model_set in zip(('reg', 'clf'), all_models):
    for model_name, model in model_set.items():
      for metric_name, metric in metrics[model_type]:
        print('On {} model {}, calculating {}'.format(
          model_type, model_name, metric_name), end='\r', flush=True)

        mean, lb, ub = test_err_bootstrap(model, metric, x_test, y_test, **kwargs)
        results.append({
          'type': model_type,
          'model': model_name,
          'metric': metric_name,
          'mean': mean,
          'lb': lb,
          'ub': ub
        })
  results = pd.DataFrame(results,
    columns=['type', 'model', 'metric', 'mean', 'lb', 'ub'])
  return results
```

Load the data and split into train and test sets.

```
data_fpath = './intermediate/wine_logged_unscaled.feather'

## Data Collection
# all functions below maintain same train-test split
# all data
x_train, x_test, y_train, y_test = gen_data(data_fpath)
# color only
x_train_col, x_test_col, y_train_col, y_test_col = gen_data(
  data_fpath, features_to_keep=['color'])
# chemicals only
x_train_chm, x_test_chm, y_train_chm, y_test_chm = gen_data(
  data_fpath, features_to_drop=['color'])
# "early" varaibles that enable early detection
x_train_sup, x_test_sup, y_train_sup, y_test_sup = gen_data(
  data_fpath, features_to_keep=[
  'color', 'fixed acidity', 'citric acid', 'residual sugar', 'chlorides'
])
```

Train the models.

```
# train on all variables
train('unscaled', x_train, y_train)
# train on colors only
train('unscaled-color', x_train_col, y_train_col)
# train on chemicals, no color
train('unscaled-chemical', x_train_chm, y_train_chm)
# train on early-detection variables
train('unscaled-superhuman', x_train_sup, y_train_sup)
```

Calculating errors on bootstrapped test data.

```
## Evaluation
results_all = evaluate_models('unscaled', x_test, y_test)
results_col = evaluate_models('unscaled-color', x_test_col, y_test_col)
results_chm = evaluate_models('unscaled-chemical', x_test_chm, y_test_chm)
results_sup = evaluate_models('unscaled-superhuman', x_test_sup, y_test_sup)
#save
results_all.to_csv('./output/results_unscaled_all.csv', index=False)
results_col.to_csv('./output/results_color_all.csv', index=False)
results_chm.to_csv('./output/results_chemical_all.csv', index=False)
results_sup.to_csv('./output/results_superhuman_all.csv', index=False)
```

Save the random forest feature importances.

```
def rf_feature_importances(model, labels):
  """
  Return feature importances for a random forest model
  """
  rf = model.best_estimator_
  # get mean importances and standard deviation
  importances_mean = rf.feature_importances_
  importances_sd = np.std(
    [tree.feature_importances_ for tree in rf.estimators_], axis=0)

  # label and return
  return pd.DataFrame(
    {'feature': labels, 'mean': importances_mean, 'sd': importances_sd})

## Model-specific results
regressors, classifiers = load_models('unscaled')
# get rf importances
rf_importances = rf_feature_importances(regressors['rf'], x_train.columns)
rf_importances.to_csv('./output/rf_importances_all.csv', index=False)
```

B.1.4 Benchmarking

Build funtions to report benchmarking results.

```
# Function to make formatted benchmark results table
benchmarktable <- function(file, task, measure) {

df <- read.csv(file = here("output", file))
capvars <- paste(str_split(str_replace(file, ".csv", ""), "_", simplify = TRUE)[3:2], collapse = " ")

df %>%
  filter(type == task) %>%
  filter(metric %in% measure) %>%
  select(-type) %>%
  mutate(metric = case_when(
    metric == "f1_score" ~ "Accuracy",
    TRUE ~ str_to_upper(metric)
    )) %>%
  mutate(model = case_when(
    model %in% c("dummy","linear") ~ str_to_title(model),
```

```
    TRUE ~ str_to_upper(model)
    )) %>%
  rename_at(., vars(starts_with("m")), str_to_title) %>%
  rename_at(., vars(ends_with("b")), str_to_upper) %>%
  kable(digits = 4,
        booktabs = T,
        format = "latex",
        caption = paste("Benchmarking", "results -", ifelse(task == "reg", "regression", "classificatio
  kableExtra::row_spec(0, bold = TRUE) %>%
  kableExtra::kable_styling(latex_options = "hold_position")

}

benchmarkplot <- function(file, task, measure) {

  df <- read.csv(file = here("output", file))
  capvars <- paste(str_split(str_replace(file, ".csv", ""), "_", simplify = TRUE)[3:2], collapse = " ")
  fmeasure <- ifelse(measure == "f1_score", "Accuracy", str_to_upper(measure))

  df %>%
    filter(type == task) %>%
    filter(metric %in% measure) %>%
    select(-type) %>%
    mutate(metric = case_when(
      metric == "f1_score" ~ str_replace(str_to_title(metric), "_", " "),
      TRUE ~ str_to_upper(metric)
    )) %>%
    mutate(model = case_when(
      model %in% c("dummy","linear") ~ str_to_title(model),
      TRUE ~ str_to_upper(model)
    )) %>%
    rename_at(., vars(starts_with("m")), str_to_title) %>%
    rename_at(., vars(ends_with("b")), str_to_upper) %>%

    ggplot(aes(Model, Mean, fill = Model)) +
    geom_bar(stat='identity') +
    scale_fill_brewer(type = 'qual', palette = 4) +
    geom_errorbar(aes(ymin = LB, ymax = UB),
                  position = position_dodge(), width=0.3) +
    theme(legend.position = "none") +
    xlab(paste("Mean", fmeasure)) +
    theme(text = element_text(size = 12),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          panel.background = element_blank(),
          axis.line = element_line(color = 'black'))
}


# a function to get the stat I want
get_stat <- function(file, task, learner, measure, digits = 3) {

  df <- read.csv(file = here::here("output", file))
```

```
  y <- subset(x = df, subset = type == task & metric == measure & model == learner, select = "mean")
  y <- round(y, digits)
  return(y)
}

# a function to get the average stat across learners
get_avg_stat <- function(file, task, measure, learners = c("rf", "linear", "mlp", "svm")) {

  y <- sapply(X = learners, function(x) get_stat(file = file, task = task, learner = x, measure = measu
  y <- mean(unlist(y))
  y <- round(y, 3)
  return(y)

}
# Get the parameters into lists for pmap

files <- c("results_unscaled_all.csv", "results_chemical_all.csv", "results_color_all.csv")
tasks <- c("reg", "clf")
measures <- c("mse", "mae", "f1_score")

benchdata <- list(files = files, tasks = tasks, measures = measures)

# Make the benchmarking tables
benchtablelist <-
  benchdata %>%
    purrr::cross_df() %>%
    filter(!(tasks == "reg" & measures == "f1_score")) %>%
    pmap(., ~ benchmarktable(..1, ..2, ..3))

# Make the benchmarking plots
benchplotlist <-
  benchdata %>%
  purrr::cross_df() %>%
  filter(!(tasks == "reg" & measures == "f1_score")) %>%
  pmap(., ~ benchmarkplot(..1, ..2, ..3))

# Note: remake the relevant plots/tables where needed, as plots/tables referenced from the lists above
```

Report on model performance.

```
benchmarktable(file = "results_unscaled_all.csv",
               task = "reg",
               measure = c("mse", "mae"))
```

```
benchmarkplot(file = "results_unscaled_all.csv",
              task = "reg",
              measure = "mse")

benchmarkplot(file = "results_unscaled_all.csv",
              task = "reg",
              measure = "mae")
```

```
benchmarkplot(file = "results_color_all.csv",
              task = "reg",
              measure = "mse")
```

```
benchmarkplot(file = "results_chemical_all.csv",
              task = "reg",
              measure = "mse")
```

```
benchmarktable(file = "results_unscaled_all.csv",
               task = "clf",
               measure = c("f1_score", "mae"))
```

```
benchmarkplot(file = "results_unscaled_all.csv",
              task = "clf",
              measure = "f1_score")

benchmarkplot(file = "results_unscaled_all.csv",
              task = "clf",
              measure = "mae")
```

```
benchmarkplot(file = "results_color_all.csv",
              task = "clf",
              measure = "f1_score")

benchmarkplot(file = "results_chemical_all.csv",
              task = "clf",
              measure = "f1_score")
```

Analyze the RF classification error (first running in python, and analyzing in R).

```
pd.DataFrame({
  'pred': classifiers['rf'].predict(x_test),
  'true': y_test
}).to_csv('./output/rf_clf_predictions.csv', index=False)
```

Make a confusion matrix.

```
rf_pred <- read_csv('./output/rf_clf_predictions.csv')
rf_pred %>%
  # transform data
  group_by(true, pred) %>%
  summarize(Count = n()) %>%
  complete(true=3:9, pred=3:9, fill=list(0)) %>%
  ungroup %>%
  mutate(Count = recode(Count, .missing=0L),
         true = factor(true),
         pred = factor(pred)) %>%

  # plot
  ggplot(aes(x = true, y = pred)) +
  geom_tile(aes(fill=Count)) +
  scale_fill_continuous(low='white', high="#191970") +
  xlab('True Values') +
  ylab('Predicted Values') +
  theme(text = element_text(size = 12),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        axis.line = element_line(color = 'black'),
```

```
        legend.direction = 'horizontal',
        legend.position = c(0.5,0.9))
```

B.1.5 Linear Regression Interpretation

Fit the linear model and interpret coefficients.

```
# This script loads the scaled and logged dataset, performs the same train-test split as in learners.py

# load packages
library(pacman)
p_load(tidyverse, here, broom, knitr, reticulate, feather)

# use python 3
pypath = "/Applications/anaconda/bin/python3.6"
condapath = "/Applications/anaconda/bin/conda"
conda_py3env = "py-env"

#use_python(python = pypath)
use_condaenv(condaenv = conda_py3env , conda = condapath)

# set seed
py_set_seed(1234)

# run the data generating code
py_run_file("gen_data_train_test.py")

# now back to R - the train and test objects exist in R using py$

# create the training set wine object by combining the x_train and y_train objects
wine_lr <- cbind(py$x_train, py$y_train)
wine_lr$quality <- wine_lr$`py$y_train`
wine_lr$`py$y_train` <- NULL

# write out a feather file to make it easier to load
feather::write_feather(x = wine_lr, path = here::here("intermediate", "wine_logged_scaled_train.feather
```

Make the table of linear regression coefficients.

```
# load packages
library(pacman)
p_load(tidyverse, here, broom, knitr, reticulate, feather)

# read in data
wine_lr <- feather::read_feather(path = here::here("intermediate", "wine_logged_scaled_train.feather"))

# run a linear model on mean-centered data and output to kable
res <- wine_lr %>%
  do(tidy(lm(quality ~ ., data = .)))

res %>%
    mutate(term = stringr::str_replace_all(term, "`", "")) %>%
    mutate(term = case_when(
      term == "(Intercept)" ~ "Intercept",
```

```
      term != "pH" ~ Hmisc::capitalize(term),
      TRUE ~ "pH"
    )) %>%
    rename(Variable = term,
           Estimate = estimate,
           SE = std.error,
           Statistic = statistic,
           `p-value` = p.value) %>%
    kable(caption = "Linear model - logged and scaled training data",
          format = "latex",
          booktabs = T,
          digits = 3) %>%
  kableExtra::row_spec(0, bold = TRUE) %>%
  kableExtra::kable_styling(latex_options = "hold_position")
```

B.1.6 Random forest interpretation

Make table of RF feature importances.

```
rf_impvar <- read.csv(here("output", "rf_importances_all.csv"))

rf_impvar %>%
  mutate(feature = as.character(feature)) %>%
  mutate(feature = case_when(
    feature == "pH" ~ "pH",
    TRUE ~ Hmisc::capitalize(feature)
  )) %>%
  kable(format = "latex",
        booktabs = T,
        digits = 3,
        col.names = c("Feature", "Mean", "SD"),
        caption = "Relative random forest feature importance in classifying wine quality") %>%
  kableExtra::row_spec(0, bold = TRUE) %>%
  kableExtra::kable_styling(latex_options = "hold_position")
```

Make function to get statistics of RF feature importances.

```
# a function to get key stats from the RF importance table

rf_get_stat <- function(feat) {

  rf_impvar <- read.csv(here::here("output", "rf_importances_all.csv"))

  # Pipes don't seem to work in inline r code in bookdown::pdf_document2 style

  # rf_impvar %>%
  #   filter(feature == feat) %>%
  #   select(mean) %>%
  #   round(3) %>%
  #   as_vector()

  y <- subset(x = rf_impvar, subset = feature == feat, select = "mean")
  y <- round(y, 3)
  return(y)
```

```
}
```

B.1.7 "Superwine" and "Superhuman"

Moving back to python, we use our models to evaluate a simulated wine.

```
# Superwine creation
lin = regressors['linear'].best_estimator_
svm = regressors['svm'].best_estimator_
# Obtain bounds of training data
feature_bounds = [(x_test[f].min(), x_test[f].max()) for f in x_train]
# Simulate a wine using the linear regression coefficients
simwine_lin = np.array([b[c > 0] for b, c in zip(feature_bounds, lin.coef_)])
simwine_lin = simwine_lin.reshape(1, -1)
# Simulate a wine using by sampling until we get a wine "better than any one
# we have seen", based on the SVM
pred = 0
while pred < 10.0:
  simwine_svm = np.array([np.random.uniform(*b) for b in feature_bounds])
  simwine_svm = simwine_svm.reshape(1, -1)
  # restrict the last feature, the color, to be red or white
  simwine_svm[:, -1] = np.random.choice((0 , 1))
  pred = svm.predict(simwine_svm)
# Save the two simulations, noting their performance on the models
simwines = pd.DataFrame(
  {'linear': simwine_lin[0], 'svm': simwine_svm[0]}, index=x_train.columns)
simwines.to_csv('./output/simwine_data.csv')
simwine_preds = pd.DataFrame({
  name: model.predict(np.vstack([simwine_lin, simwine_svm]))
  for name, model in regressors.items()
}, index = ['linear', 'svm'])
simwine_preds.to_csv('./output/simwine_preds.csv')
```

Load in these results to R.

```
simwine_preds <- read_csv('./output/simwine_preds.csv')
simwines <- read_csv('./output/simwine_data.csv')
```

Make a table of the superwine features.

```
simwines %>%
  mutate(avg = origwine %>% select(-quality) %>% summarize_all(mean) %>% t,

         linear = if_else(X1 %in% concvars, exp(linear), linear),
         svm = if_else(X1 %in% concvars, exp(svm), svm),

         linear = sprintf('%.2f', linear) %>% recode('1.00'='Red'),
         svm = sprintf('%.2f', svm) %>% recode('0.00'='White'),
         avg = sprintf('%.2f', avg) %>% recode('0.25'='25% Red')
         ) %>%
  select(X1, avg, linear, svm) %>%
  kable(format = "latex",
        booktabs = T,
        digits = 3,
        align = 'r',
```

```
        col.names = c('Variable', 'Mean','Linear', 'SVM'),
        caption = "Simulated Wine Characteristics") %>%
  kableExtra::row_spec(0, bold = TRUE) %>%
  kableExtra::kable_styling(latex_options = "hold_position")
```

Plot predictions of models on the superwines.

```
simwine_preds %>%
  gather(key='Model', value='Rating', -X1) %>%
  rename(`Optimized for` = X1) %>%
  mutate(
    Model = recode(
      Model,
      dummy = 'Dummy',
      linear = 'Linear',
      svm = 'SVM',
      rf = 'Random Forest',
      mlp = 'Neural Net'
    ),
    `Optimized for` = recode(
      `Optimized for`,
      linear = 'Linear',
      svm = 'SVM'
    ),
    highlight = `Optimized for` == `Model`
  ) %>%
  ggplot() +
  geom_bar(aes(x = Model, y = Rating, fill = `Optimized for`),
           stat='identity', position = position_dodge()) +
  scale_fill_brewer(type='qual', palette = 1) +
  geom_bar(aes(x = Model, y = Rating, fill = `Optimized for`, color = highlight),
           alpha=0, stat='identity', position = position_dodge(), size=1.25) +
  scale_color_manual(values=c(NA, "black"), guide = FALSE) +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        axis.line = element_line(color = 'black'))
```

Pull out the results of the early-detection model.

```
sh_rf_mse <-
  read_csv('./output/results_superhuman_all.csv') %>%
  filter(type == 'reg', model == 'rf', metric == 'mse') %>%
  summarize_at(vars('mean', 'lb', 'ub'), round, 2)

all_rf_mse <-
  read_csv('./output/results_unscaled_all.csv') %>%
  filter(type == 'reg', model == 'rf', metric == 'mse') %>%
  summarize_at(vars('mean', 'lb', 'ub'), round, 2)
```

B.1.8 Clustering code

Make clusters of the data

```
mclust_results <-
  select(wine, -quality) %>%
  Mclust(1:15)
saveRDS(mclust_results, './intermediate/mclust_results.rds')
```

```
mclust_results <- readRDS('./intermediate/mclust_results.rds')

wine$Cluster <- factor(mclust_results$classification)

# do we observe clearly-delineated clusters?
wine %>%
  ggpairs(
      mapping = aes(color = Cluster, alpha = 0.4),
      columns = c(variable_var$key[1:n])
  )

# do they divide up quality in any meaningful way?
# not especially! perhaps this isn't the most salient feature
wine %>%
  ggplot(aes(x = quality)) +
  geom_bar(aes(fill = Cluster)) +
  ylab('Count') +
  xlab('Quality')

# remove cluster variable
wine$cluster <- NULL
```