Wine dataset report

July 29, 2018

1 Executive Summary

This report documents the results of the analysis carried out to analyse the feasibility of the vision statement of our customer, the "food/AI start-up". The main findings and suggestions for further course of action are listed here.

Goals

- Find a suitable predictive-model (AI variety) to predict the wine quality rating based on the given features.
- Find the components of wine that make a good wine (if possible).
- Analyze the feasibility of creating the perfect wine.
- Clarifying what the AI is capable of learning from humans.
- Suggesting further of action based on findings.

Findings & their Limitations

- (Bagging) Ensemble of decision trees seem to perform better than other models for this specific task. This variety of AI works by learning to ask a series of questions with template "Is feature X ¿ C?". Based on the answers to a series of questions, it rates the wine. Many such AIs were trained (to possibly ask different sets of questions) and they were allowed to "vote" to find the best estimate of the rating for the wine. The expected error of this AI is low compared to other varieties. If the customer finds it low enough for the application, then this AI can rate wines in a huge collection of samples (if the numerical data is available) at super-human speeds without getting drunk.
- Good wines (rated 7 and above) seem to have high alcohol content and low density. However, these are just observations of association and are not really the "components" that make a good wine. A causal relationship can be established only from controlled wine-making experiments and not by simply analyzing existing wines.
- The AIs that make up the current state-of-the-art can only imitate humans if they are trained with human responses. They can generalize to unseen wines only if they are similar (in terms of features) to the wines that they are already trained with.
- As they only imitate humans closely based on their training, they cannot find the perfect wine that is better than every other wine ever made. If they try, more often than not, they would fail.

Conclusions

We conclude that AIs can be used to predict the wine quality of the wines that they are trained on. They are best used to predict the wine quality of large samples at high speeds. There is some possibility of finding a repeatable a way to make good wine; however, that requires data from properly designed experiments based on a controlled (or controllable) manufacturing process. The currently available data cannot be used for this purpose as it was not recorded in a controlled way.

Even if we have the data collected in the right way to find a repeatable technique to make good wine (similar to those already seen during training), we cannot create the perfect ideal wine as the AIs are incapable of meaningfully extrapolating beyond their training experience.

It might be far more rewarding to use the data-scientist man power on algorithmic marketing, customer analytics and social media campaigns as AIs are good at prediction in a large scale.

2 Addressing the Questions of Interest

Can we use the sommelier/wine data to create an AI with super-human performance in wine-tasting?

The answer to this question depends on the precise meaning of the term "super-human". As taste is a human sense associated with humanity, the term "super-human" tasting lends itself to numerous creative —but possibly unrealizable— interpretations. In the current context we interpret "super-human performance in wine tasting" in a specific way and rephrase the question as follows.

Rephrased Question: Can a machine be trained to give a numeric rating to a wine-sample, that is same (or close) to what the (median) human expert would give, but at a much higher speed to a much larger volume of samples (assuming all necessary numerical data are already available) in a repeatable way without tiring or getting drunk.

Can this be answered from the data: For the rephrased question, the answer is a "weak" yes, where weak can be quantified by further analysis.

Questions to Address: Can Supervised Learning be used to predict wine rating from the given data (chemical composition and color)? Which model among the numerous ones available is suitable for this purpose? How well (quantified using loss function) does the best model perform for this application?

Which components of a wine make a wine a good wine?

Again "good" is considered to be something that is associated with high numerical ratings given by the experts who rated the wines in the dataset. To be specific we define good-wine as those with ratings 7 or 8 or 9 in the dataset. With this interpretation, we rephrase the question as follows

Rephrased Question: In what aspects and by how much does a highly rated (7 or 8 or 9) wine differ from other wines?

Can this be answered from the data: If the aspects considered by the question are limited to those "features" available in the dataset, then it might be possible to find the features that vary considerably between the highly rated wines and others. However, the conclusion would only be associative (eg. feature X is high in good wine) and not causal (eg. high feature X makes a good wine) as the data doesn't seem to be collected based on an experimental setup that could facilitate making causal relationships.

Questions to Address: For each feature in the dataset, is there sufficient evidence that this feature is different for the highly rated wines? Can model-specific inference be used for this purpose? If so, then which model?

Can we use AI to create the perfect wine (whose quality exceeds all that we have seen)?

"AI" can mean a lot of things; here is it taken to mean a trained supervised learning algorithm. So, we rephrase as

	count	mean	std	min	25%	50%	75%	max
quality	6497	5.81838	0.873255	3	5	6	6	9
fixed acidity	6497	7.21531	1.29643	3.8	6.4	7	7.7	15.9
volatile acidity	6497	0.339666	0.164636	0.08	0.23	0.29	0.4	1.58
citric acid	6497	0.318633	0.145318	0	0.25	0.31	0.39	1.66
residual sugar	6497	5.44324	4.7578	0.6	1.8	3	8.1	65.8
chlorides	6497	0.0560339	0.0350336	0.009	0.038	0.047	0.065	0.611
free sulfur dioxide	6497	30.5253	17.7494	1	17	29	41	289
total sulfur dioxide	6497	115.745	56.5219	6	77	118	156	440
density	6497	0.994697	0.00299867	0.98711	0.99234	0.99489	0.99699	1.03898
pH	6497	3.2185	0.160787	2.72	3.11	3.21	3.32	4.01
sulphates	6497	0.531268	0.148806	0.22	0.43	0.51	0.6	2
alcohol	6497	10.4918	1.19271	8	9.5	10.3	11.3	14.9

Table 1: Numerical summary of given data

Rephrased Question: Can a trained model be used to find the combination of features, which when used to later create a wine with those features (assuming such a wine can be created), would receive a rating from a human expert, which is higher than all the ratings the model was trained with?

Can this be answered from the data: The data the model is trained with constrains the model and hence is useful to answer the question.

Questions to Address: Is the learnt function invertible? How does the training data limit the model?

. . . what would it be that AIs could, or would learn from humans?

Can this be answered from the data: Again, the data the model is trained with constrains the model and hence is useful to answer the question.

Questions to Address: How does the training data limit the model? What kind of learning happens in supervised learning?

3 Exploratory Data Analysis

Numerical Summaries: The numerical summaries for the given data is shown in table 1. It must be noted that some of the values of concentration of citric acid is exactly zero. This might pose some problems during transformations.

Target Distribution Analysis: The mean quality of the wines in the dataset is around 5.8 and the standard deviation is about 0.9. This indicates that there are lot of mediocre wines in the dataset with a dearth of examples for the best and worst wines. In fact, we don't have wines of quality less than 3. The same is also confirmed by the quantiles (table 1) and the histogram (figure 1). This is a potential reason for concern as this might affect the predictive capacity of the model while dealing with the best and worst of wines.

Difference between Red and White wines: Approximately 25% of the samples in the dataset are red-wine and the remaining 75% are white wine. As seen in the histogram (figure 1), the average quality of red wines (5.63) in the dataset is lower than that of the white wine (5.88). But the hypothesis test results in table 2 indicates that there is not significant difference. Also in the given dataset, there are no red-wines with quality greater than 8. Again there are few examples of the best and worst wines in both the categories.

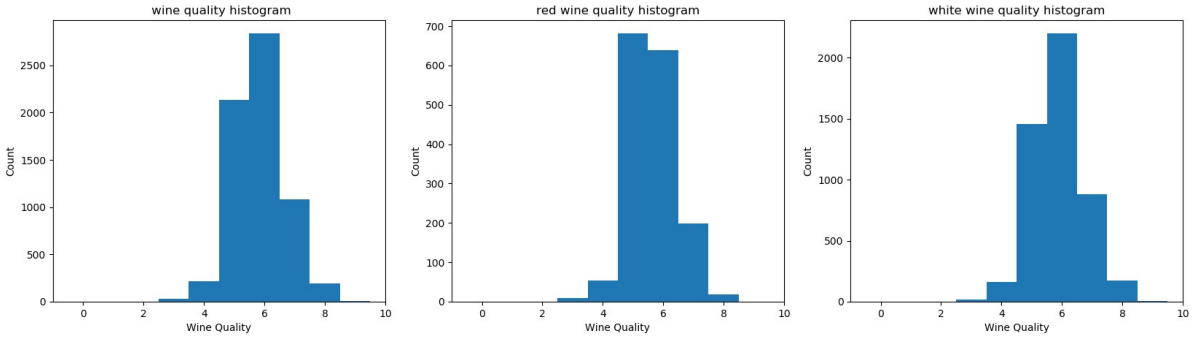

Figure 1: Wine quality histograms

feature	R>W: p value	R>W: Result	R<W: p value	R<W: Result
quality	0.8368	-	0.1632	-
fixed acidity	0.0000	T	1.0000	-
volatile acidity	0.0000	T	1.0000	-
citric acid	0.9688	-	0.0312	T
residual sugar	0.9969	-	0.0031	T
chlorides	0.0000	T	1.0000	-
free sulfur dioxide	1.0000	-	0.0000	T
total sulfur dioxide	1.0000	-	0.0000	T
density	0.0005	T	0.9995	-
pH	0.0017	T	0.9983	-
sulphates	0.0000	T	1.0000	-
alcohol	0.6051	-	0.3949	-

Table 2: Independent Comparison of features between red and white wine. R¿W indicates the p values and the result for the alternate hypothesis that the feature is higher in red wine. R¡W indicates the alternate hypothesis that the feature is lower in red wine."T" indicates that there is sufficient evidence to reject the null hypothesis that the features are equal.

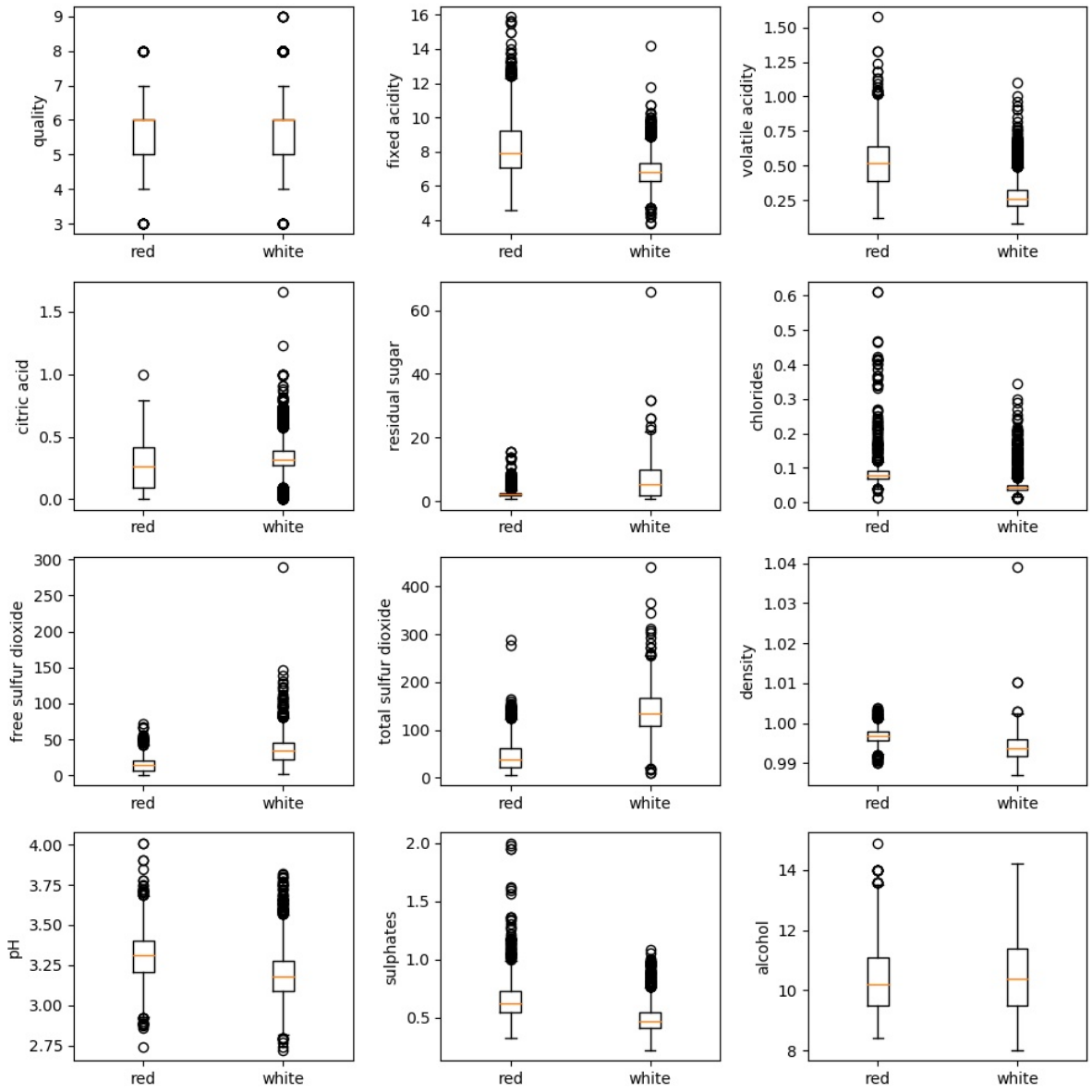

Figure 2: Bar plots comparing red and white wines

Based on the box plot (figure 2) it can be seen that the chemical composition of red and white wine vary by quite a few aspects. We did (unpaired-z) hypothesis tests independently comparing each feature of the two wine colors. The p-values and the results are show in table 2. Looks like there is quite a bit of difference in chemical composition between red and white wine. Note that the results hold only if the independence assumptions holds. If there is interaction between features then the results might not be valid.

Further, the red and white wines (indicated in red and blue color respectively) show obvious clusters in the scatter plots (figure 3). The silhouette score with color labels is 0.42 and silhouette score with random labels is close to 0 as expected. The high and positive silhouette score indicates color based clustering. So, wines of different color cluster together and seem to be different in terms of chemical composition.

Transforming the Data: The law of mass action states that the rate of chemical reaction is proportional to the product of masses of the participating elements. So, if we are fitting

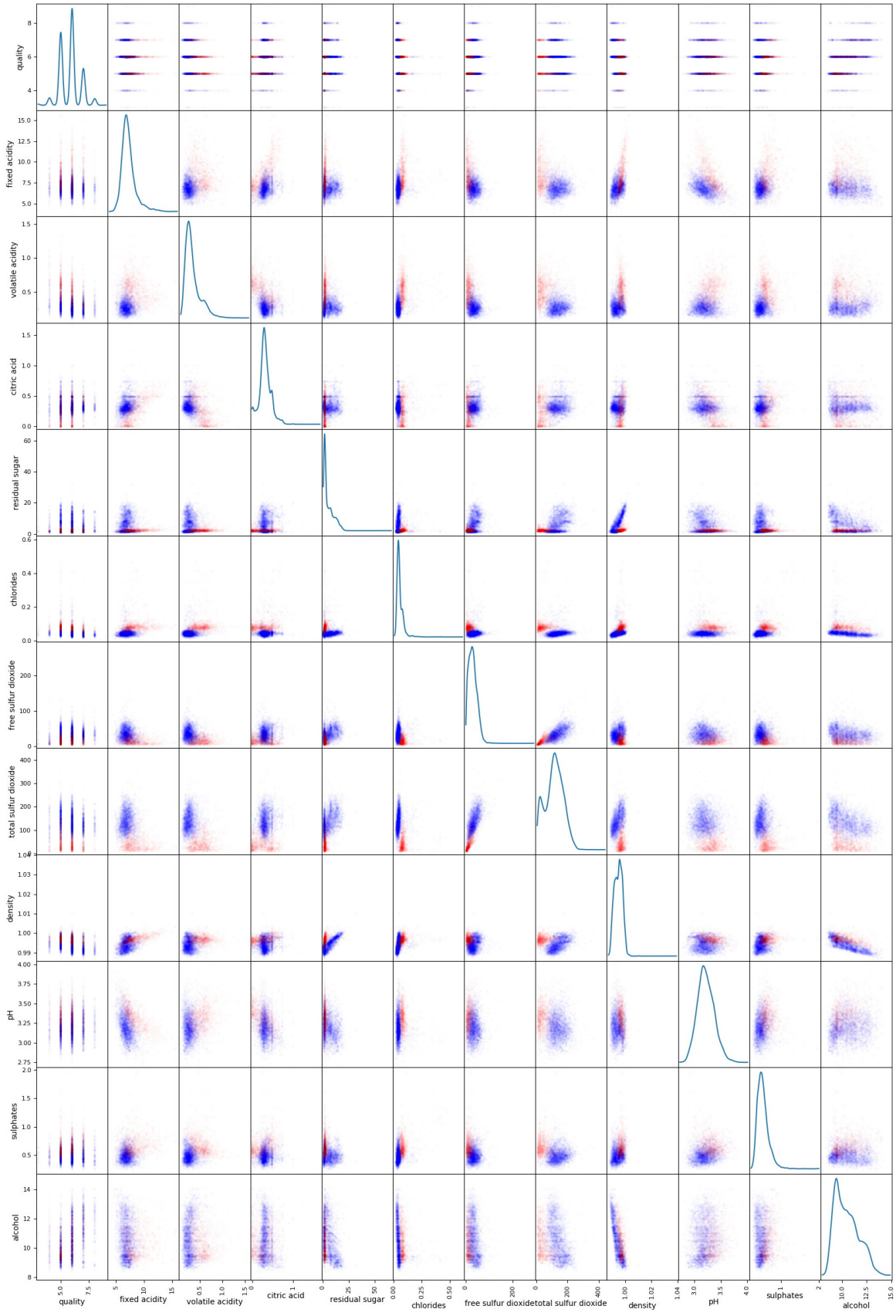

Figure 3: Matrix scatter plot of all features. Red wine samples are indicated using red dots and the white wine samples are indicated using blue dots

	count	mean	std	min	25%	50%	75%	max
quality	6497	5.81838	0.873255	3	5	6	6	9
fixed acidity	6497	1.962	0.164255	1.335	1.8563	1.94591	2.04122	2.76632
volatile acidity	6497	-1.18029	0.439046	-2.52573	-1.46968	-1.23787	-0.916291	0.457425
citric acid	6497	-1.10282	0.520564	-2.99573	-1.20397	-1.02165	-0.820981	0.536493
residual sugar	6497	1.3257	0.863555	-0.510826	0.587787	1.09861	2.09186	4.18662
chlorides	6497	-2.99062	0.431162	-4.71053	-3.27017	-3.05761	-2.73337	-0.492658
free sulfur dioxide	6497	3.21613	0.698635	0	2.83321	3.3673	3.71357	5.66643
total sulfur dioxide	6497	4.56383	0.714447	1.79176	4.34381	4.77068	5.04986	6.08677
density	6497	0.994697	0.00299867	0.98711	0.99234	0.99489	0.99699	1.03898
pH	6497	3.2185	0.160787	2.72	3.11	3.21	3.32	4.01
sulphates	6497	-0.666818	0.257057	-1.51413	-0.84397	-0.673345	-0.510826	0.693147
alcohol	6497	10.4918	1.19271	8	9.5	10.3	11.3	14.9

Table 3: Numerical summary of log-transformed data

a linear model that models a quantity that is again linearly related to the rate of chemical reaction, then the log transformation would immensely help. However, there does not seem to be any conclusive relations between the rate of chemical reaction (that might happen inside the wine when exposed to atmosphere or during tasting or during some other phase - which itself is unknown) and perceived (or idealized) quality of wine. Further, most of the models that we are comparing (except for the linear parsimonious baseline) are capable of accommodating non-linear relations. So, there doesn't really seem to be any obvious transformation that has to be done.

However, to give the parsimonious baseline models a good chance against the state of the art models, we decided to perform the log transformation on the various concentrations (only features with unit g/dm^3 or mg/dm^3). The concentration of citric acid happens to be 0 in some cases. To avoid negative infinity during log transformation, we added a shift to these concentration values by 0.005 g/dm^3. This shift was (subjectively) decided to be half the least count (0.01). The scatter plot for the transformed data is also shown in the figure 4. As expected the crowding of points has visibly reduced.

The numerical summaries for the log-transformed data is in table 3.

Collinearity, Non-Linearity and Clustering in Scatter Plots: As already mentioned, the red and white wines cluster together and forms obvious clusters in the scatter plots. There is not much relation between the features except for a couple of exceptions. There is a roughly linear relation between total sulphur-di-oxide and free sulphur-di-oxide ($\rho = 0.78$). There is also a slightly negatively correlated linear relation between alchohol content and density ($\rho = -0.68$). All correlation co-efficient values are shown in table 4.

4 Predictive Benchmarking Experiments

The predictive experiments were carried out to quantitatively compare predictive performance of different models on the given dataset. The models in table 5 were considered for the comparison. For each model variety three variants were considered as follows

- Trained with all features in the dataset
- Trained with only chemical composition

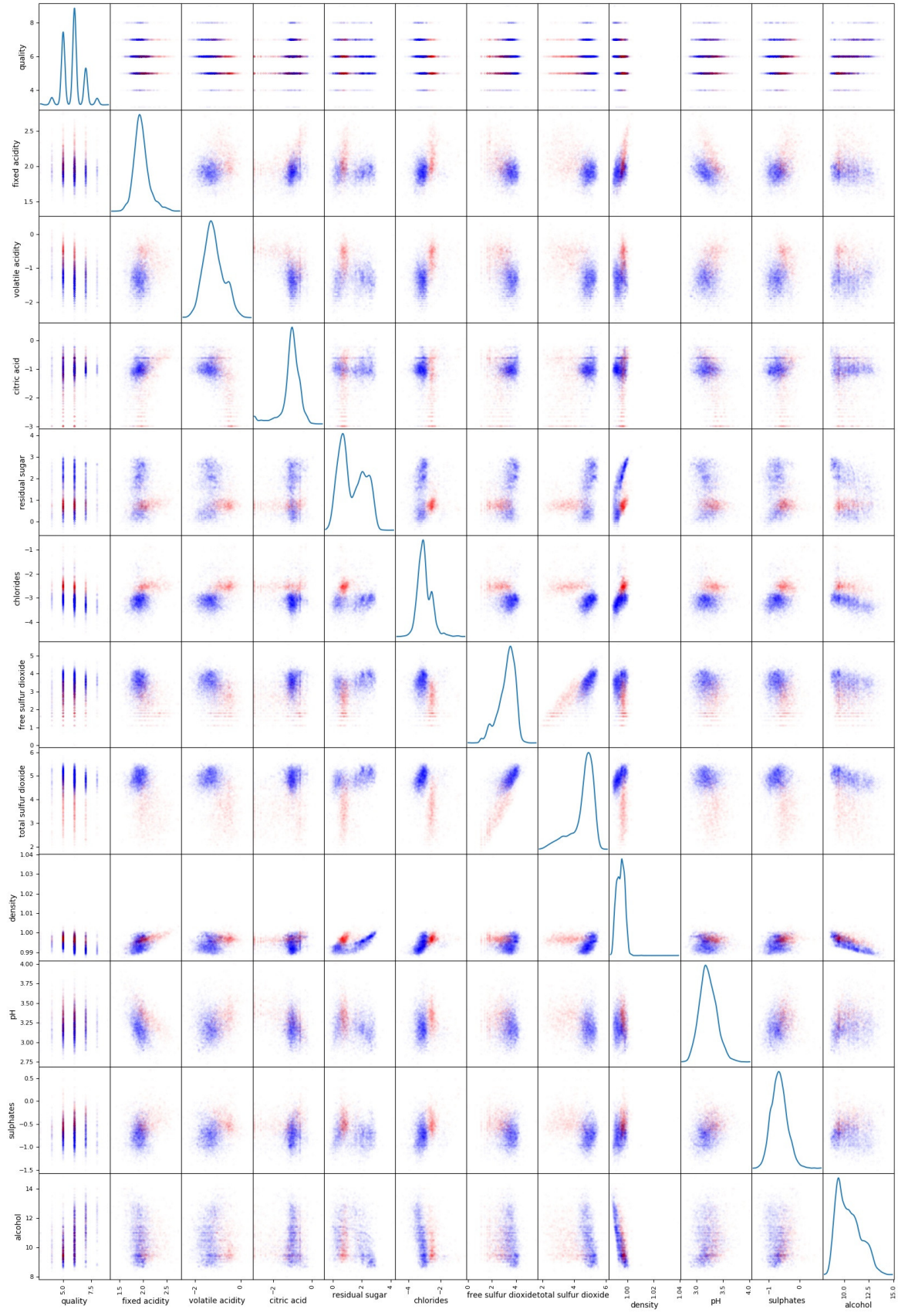

Figure 4: Matrix scatter plot of all features (after log-transformation). Red wine samples are indicated using red dots and the white wine samples are indicated using blue dots

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
fixed acidity	1.000	0.220	0.240	-0.077	0.379	-0.332	-0.376	0.466	-0.274	0.277	-0.117
volatile acidity	0.220	1.000	-0.453	-0.120	0.438	-0.380	-0.437	0.267	0.219	0.239	-0.021
citric acid	0.240	-0.453	1.000	0.162	-0.136	0.192	0.300	0.017	-0.372	-0.030	0.009
residual sugar	-0.077	-0.120	0.162	1.000	-0.102	0.385	0.412	0.511	-0.246	-0.168	-0.309
chlorides	0.379	0.438	-0.136	-0.102	1.000	-0.293	-0.400	0.521	0.123	0.391	-0.371
free sulfur dioxide	-0.332	-0.380	0.192	0.385	-0.293	1.000	0.784	-0.053	-0.146	-0.223	-0.136
total sulfur dioxide	-0.376	-0.437	0.300	0.412	-0.400	0.784	1.000	-0.106	-0.251	-0.320	-0.180
density	0.466	0.267	0.017	0.511	0.521	-0.053	-0.106	1.000	0.012	0.278	-0.687
pH	-0.274	0.219	-0.372	-0.246	0.123	-0.146	-0.251	0.012	1.000	0.231	0.121
sulphates	0.277	0.239	-0.030	-0.168	0.391	-0.223	-0.320	0.278	0.231	1.000	-0.024
alcohol	-0.117	-0.021	0.009	-0.309	-0.371	-0.136	-0.180	-0.687	0.121	-0.024	1.000

Table 4: Pearson Correlation coefficient between features (after log-transformation)

Regressors	Comments / Parameters and values considered for tuning
DummyRegressor	Predicts mean, no tuning
OLS Linear Regression	tuning: intercept = {yes, no}
Support Vector Regression	RBF Kernel, tuning: penality C = {0.1, 1.0, 10.0}, lower the C more the regularization, learning rate = {0.01, 0.001}
MLP Regressor	tuning: hidden_layer_size = {2 layers: 32 neurons in each, 3 layers: 16 neurons in each}
Ensenble Regressor	Bagging of 20 Decision Tree Regressors

Classifiers	Comments / Parameters and values considered for tuning
DummyClassifier	Predicts randomly as per training class distribution, no tuning
Logistic Regression	tuning: intercept = {yes, no}, penality C = {0.1, 1.0, 10.0}, lower the C more the regularization
Support Vector Classification	RBF Kernel, tuning: penality C = {0.1, 1.0, 10.0}, lower the C more the regularization
MLP Classifier	tuning: hidden_layer_size = {2 layers: 32 neurons in each, 3 layers: 16 neurons in each}, learning rate = {0.01, 0.001}
Ensemble Classifier	Bagging of 20 Decision Tree Classifiers

Table 5: Parameters and their values considered for tuning. Top table lists the Regressors and the bottom table lists Classifiers

- Trained with only the color of the wine for prediction

The total list of models considered along with their abbreviations is listed in table 6. Also, we decided to compare the regressors and classifiers separately as we see them as different paradigms in the way they treat a mistake and they have different loss functions. Also, all models were trained with log-transformed data as explained in section. 3

Goal

The comparisons were made so that the existence of the best model (among those compared) serves as a proof that the wine quality can be predicted from chemical composition and color of wine. Further the performance metric of this best model quantifies how well the wine quality can be predicted.

From the same setup, the question asking whether wine color adds predictive power above chemical composition (and vice versa) can be answered by comparing the prediction metric of appropriately trained models.

Note that the goal is to answer the questions only in the context of the models selected for comparison and is in no way the absolute truth. To be as comprehensive as possible and to have good baselines for comparison, along with state of the art models, we also considered parsimonious simple models and dummy baselines.

Tuning Strategy

Some of the models considered for comparison have some hyper parameters to be tuned. As a part of nested re-sampling for tuning and validation, we did cross validated grid search on each training split of the folds for tuning (and also refit to the whole training set using the best parameters). We did this using the `GridSearchCV` wrapper (with no. of folds $K = 3$) and validation is effectively for the whole tuning strategy along with the model. The different hyper-parameter values considered for tuning are listed in table 5. To reduce computational cost many other parameters (of relatively lesser subjective significance) were used as per `sklearn`'s defaults.

Comparison: Metric and Error Bar Calculation

The squared error was used as the loss function for regressors and the 0/1 loss was used as the loss function for classifiers.

$$L_{SE}(Y, \hat{Y}) = (Y - \hat{Y})^2$$

$$L_{0/1}(Y, \hat{Y}) = \mathbb{I}(Y \neq \hat{Y})$$

Note that the afore mentioned losses are per point losses (quantifying the difference between test point Y and the prediction $f_{\mathcal{T}}(X) = \hat{Y}$), which were then aggregated by mean over the test test ν. This mean value ($\hat{\epsilon}(f|\mathcal{T})$) serves as an unbiased consistent estimate of the generalization error ($\epsilon(f|\mathcal{T})$). The variance of this estimate is $V(\hat{\epsilon}(f|\mathcal{T}))$.

$$\hat{\epsilon}(f|\mathcal{T}) = \frac{1}{\#\nu} \sum L$$

$$V(\hat{\epsilon}(f|\mathcal{T})) = \frac{1}{\#\nu(\#\nu - 1)} \sum_{\nu} (L - \hat{\epsilon}(f|\mathcal{T}))^2$$

Estimator	Description
DR	Dummy Regressor (predicts mean) with all variables
DR CHM	Dummy Regressor (predicts mean) with all variables except wine color
DR COL	Dummy Regressor (predicts mean) with only wine color data
LR	Linear Regressor all variables
LR CHM	Linear Regressor with all variables except wine color
LR COL	Linear Regressor with only wine color data
SVR	Support Vector Regressor (RBF Kernel) with all variables
SVR CHM	Support Vector Regressor (RBF Kernel) with all variables except wine color
SVR COL	Support Vector Regressor (RBF Kernel) with only wine color data
NNR	Multi Layer perceptron Regressor with all data
NNR CHM	Multi Layer perceptron Regressor with all data except wine color
NNR COL	Multi Layer perceptron Regressor with only wine color data
ER	Bagging Ensemble of 20 decision tree Regressors with all data
ER CHM	Bagging Ensemble of 20 decision tree Regressors with all data except wine color
ER COL	Bagging Ensemble of 20 decision tree Regressors with only wine color data
DC	Dummy Classifier (random as per (training) class distribution) with all variables
DC CHM	Dummy Classifier (random as per (training) class distribution) with all variables except wine color
DC COL	Dummy Classifier (random as per (training) class distribution) with only wine color data
LC	Linear Classifier all variables
LC CHM	Linear Classifier with all variables except wine color
LC COL	Linear Classifier with only wine color data
SVC	Support Vector Classifier (RBF Kernel) with all variables
SVC CHM	Support Vector Classifier (RBF Kernel) with all variables except wine color
SVC COL	Support Vector Classifier (RBF Kernel) with only wine color data
NNC	Multi Layer perceptron Classifier with all data
NNC CHM	Multi Layer perceptron Classifier with all data except wine color
NNC COL	Multi Layer perceptron Classifier with only wine color data
EC	Bagging Ensemble of 20 decision tree Classifiers with all data
EC CHM	Bagging Ensemble of 20 decision tree Classifiers with all data except wine color
EC COL	Bagging Ensemble of 20 decision tree Classifiers with only wine color data

Table 6: Legend for Comparison Tables 7, 8, 9, 10

Regressor	Mean(SE)	Variance(MSE)	95% Confidence Interval
DR	0.762765	0.000999	(0.7008, 0.8247)
DR CHM	0.762765	0.000999	(0.7008, 0.8247)
DR COL	0.762765	0.000999	(0.7008, 0.8247)
LR	0.528562	0.000651	(0.4786, 0.5786)
LR CHM	0.530022	0.000654	(0.4799, 0.5802)
LR COL	0.752257	0.000979	(0.6909, 0.8136)
SVR	0.483202	0.000544	(0.4375, 0.5289)
SVR CHM	0.483762	0.000549	(0.4378, 0.5297)
SVR COL	0.769119	0.000983	(0.7077, 0.8306)
NNR	0.531123	0.000639	(0.4816, 0.5807)
NNR CHM	0.539284	0.000672	(0.4885, 0.5901)
NNR COL	0.752864	0.000984	(0.6914, 0.8143)
ER	0.385379	0.000507	(0.3412, 0.4295)
ER CHM	0.381353	0.000507	(0.3372, 0.4255)
ER COL	0.752070	0.000978	(0.6908, 0.8134)

Table 7: Comparison of mean squared error of Regressors

Note that both the above estimates are conditioned on the training set $\mathcal{T}$. The variation in the trained functional is not considered. This is an unfortunate limitation. However, we did cross validation with $K = 5$ and aggregated both the estimates by mean. Though the 5 individual estimates are correlated, by variance reduction lemma, we get a low-variance estimate ($\bar{x}$) of the (conditioned) generalization error (μ) and the low-variance estimate ($\hat{\sigma}$) of variance of the estimate of the (conditioned) generalization error. Observing that

$$\frac{\bar{x} - \mu}{\sqrt{\hat{\sigma}}} \sim N(0, 1)$$

we calculated the confidence intervals. These are reported in the comparison table for regressors (table 7) and that for classifiers (table 9)

Comparison: Hypothesis Testing

To compare the confidence intervals, we resorted to a simple unpaired z-test. Because of the high degrees of freedom owing to the sufficiently large test set (more than 50 points), we decided not to use a t-distribution and approximated with a standard normal distribution. With $\mu_{m1} - \mu_{m2}$ as the difference in the generalization errors of the models, and estimate of this difference $\bar{x}_{m1} - \bar{x}_{m2}$, and the estimate of the corresponding variance $\hat{\sigma}_{m1} + \hat{\sigma}_{m2}$, we have,

$$\frac{(\bar{x}_{m1} - \bar{x}_{m2}) - (\mu_{m1} - \mu_{m2})}{\sqrt{\hat{\sigma}_{m1} + \hat{\sigma}_{m2}}} \sim N(0, 1)$$

We calculated the p-value from the distribution for the hypothesis ($H0$ is the null hypothesis and $H1$ is the alternate hypothesis).

$$H0 : \mu_{m1} - \mu_{m2} = 0$$
$$H1 : \mu_{m1} - \mu_{m2} < 0$$

The tests were performed with $\alpha = 0.05$ and the results are reported in table 8 for regressors and in table 10 for the classifiers.

	DR	DR CHM	DR COL	LR	LR CHM	LR COL	SVR	SVR CHM	SVR COL	NNR	NNR CHM	NNR COL	ER	ER CHM	ER COL
DR	0.5	0.5	0.5	1	1	0.59	1	1	0.44	1	1	0.59	1	1	0.6
DR CHM	0.5	0.5	0.5	1	1	0.59	1	1	0.44	1	1	0.59	1	1	0.6
DR COL	0.5	0.5	0.5	1	1	0.59	1	1	0.44	1	1	0.59	1	1	0.6
LR	0	0	0	0.5	0.48	0	0.91	0.9	0	0.47	0.38	0	1	1	0
LR CHM	0	0	0	0.52	0.5	0	0.91	0.91	0	0.49	0.4	0	1	1	0
LR COL	0.41	0.41	0.41	1	1	0.5	1	1	0.35	1	1	0.49	1	1	0.5
SVR	0	0	0	0.09	0.09	0	0.5	0.49	0	0.08	0.05	0	1	1	0
SVR CHM	0	0	0	0.1	0.09	0	0.51	0.5	0	0.08	0.06	0	1	1	0
SVR COL	0.56	0.56	0.56	1	1	0.65	1	1	0.5	1	1	0.64	1	1	0.65
NNR	0	0	0	0.53	0.51	0	0.92	0.92	0	0.5	0.41	0	1	1	0
NNR CHM	0	0	0	0.62	0.6	0	0.95	0.94	0	0.59	0.5	0	1	1	0
NNR COL	0.41	0.41	0.41	1	1	0.51	1	1	0.36	1	1	0.5	1	1	0.51
ER	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0.55	0
ER CHM	0	0	0	0	0	0	0	0	0	0	0	0	0.45	0.5	0
ER COL	0.4	0.4	0.4	1	1	0.5	1	1	0.35	1	1	0.49	1	1	0.5

	DR	DR CHM	DR COL	LR	LR CHM	LR COL	SVR	SVR CHM	SVR COL	NNR	NNR CHM	NNR COL	ER	ER CHM	ER COL
DR	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DR CHM	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DR COL	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
LR	T	T	T	-	-	T	-	-	T	-	-	T	-	-	T
LR CHM	T	T	T	-	-	T	-	-	T	-	-	T	-	-	T
LR COL	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SVR	T	T	T	-	-	T	-	-	T	-	-	T	-	-	T
SVR CHM	T	T	T	-	-	T	-	-	T	-	-	T	-	-	T
SVR COL	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
NNR	T	T	T	-	-	T	-	-	T	-	-	T	-	-	T
NNR CHM	T	T	T	-	-	T	-	-	T	-	-	T	-	-	T
NNR COL	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ER	T	T	T	T	T	T	T	T	T	T	T	T	-	-	T
ER CHM	T	T	T	T	T	T	T	T	T	T	T	T	-	-	T
ER COL	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 8: Testing Alternate Hypothesis that the regressor in the row index is better than the regressor in column index. First table gives the p-values and the second table gives the result of the test ($\alpha = 0.05$). "T" indicates cases where there is enough evidence to reject the null hypotheiss that the test loss of both regressors are the same

Classifier	Mean(0/1 loss)	Var(Mean(0/1 loss))	95% Confidence Interval
DC	0.666002	0.000170	(0.6404, 0.6916)
DC CHM	0.667998	0.000170	(0.6424, 0.6936)
DC COL	0.671543	0.000171	(0.6459, 0.6971)
LC	0.458059	0.000191	(0.4310, 0.4852)
LC CHM	0.456980	0.000191	(0.4299, 0.4841)
LC COL	0.563643	0.000189	(0.5367, 0.5906)
SVC	0.438971	0.000190	(0.4120, 0.4660)
SVC CHM	0.438664	0.000190	(0.4117, 0.4656)
SVC COL	0.563643	0.000189	(0.5367, 0.5906)
NNC	0.467599	0.000192	(0.4405, 0.4947)
NNC CHM	0.452516	0.000191	(0.4254, 0.4796)
NNC COL	0.563643	0.000189	(0.5367, 0.5906)
EC	0.338464	0.000172	(0.3127, 0.3642)
EC CHM	0.332154	0.000171	(0.3065, 0.3578)
EC COL	0.563643	0.000189	(0.5367, 0.5906)

Table 9: Comparison of mean(0/1 loss) of Classifiers

	DC	DC CHM	DC COL	LC	LC CHM	LC COL	SVC	SVC CHM	SVC COL	NNC	NNC CHM	NNC COL	EC	EC CHM	EC COL
DC	0.5	0.46	0.38	1	1	1	1	1	1	1	1	1	1	1	1
DC CHM	0.54	0.5	0.42	1	1	1	1	1	1	1	1	1	1	1	1
DC COL	0.62	0.58	0.5	1	1	1	1	1	1	1	1	1	1	1	1
LC	0	0	0	0.5	0.52	0	0.84	0.84	0	0.31	0.61	0	1	1	0
LC CHM	0	0	0	0.48	0.5	0	0.82	0.83	0	0.29	0.59	0	1	1	0
LC COL	0	0	0	1	1	0.5	1	1	0.5	1	1	0.5	1	1	0.5
SVC	0	0	0	0.16	0.18	0	0.5	0.51	0	0.07	0.24	0	1	1	0
SVC CHM	0	0	0	0.16	0.17	0	0.49	0.5	0	0.07	0.24	0	1	1	0
SVC COL	0	0	0	1	1	0.5	1	1	0.5	1	1	0.5	1	1	0.5
NNC	0	0	0	0.69	0.71	0	0.93	0.93	0	0.5	0.78	0	1	1	0
NNC CHM	0	0	0	0.39	0.41	0	0.76	0.76	0	0.22	0.5	0	1	1	0
NNC COL	0	0	0	1	1	0.5	1	1	0.5	1	1	0.5	1	1	0.5
EC	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0.63	0
EC CHM	0	0	0	0	0	0	0	0	0	0	0	0	0.37	0.5	0
EC COL	0	0	0	1	1	0.5	1	1	0.5	1	1	0.5	1	1	0.5

	DC	DC CHM	DC COL	LC	LC CHM	LC COL	SVC	SVC CHM	SVC COL	NNC	NNC CHM	NNC COL	EC	EC CHM	EC COL
DC	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DC CHM	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DC COL	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
LC	T	T	T	-	-	T	-	-	T	-	-	T	-	-	T
LC CHM	T	T	T	-	-	T	-	-	T	-	-	T	-	-	T
LC COL	T	T	T	-	-	-	-	-	-	-	-	-	-	-	-
SVC	T	T	T	-	-	T	-	-	T	-	-	T	-	-	T
SVC CHM	T	T	T	-	-	T	-	-	T	-	-	T	-	-	T
SVC COL	T	T	T	-	-	-	-	-	-	-	-	-	-	-	-
NNC	T	T	T	-	-	T	-	-	T	-	-	T	-	-	T
NNC CHM	T	T	T	-	-	T	-	-	T	-	-	T	-	-	T
NNC COL	T	T	T	-	-	-	-	-	-	-	-	-	-	-	-
EC	T	T	T	T	T	T	T	T	T	T	T	T	-	-	T
EC CHM	T	T	T	T	T	T	T	T	T	T	T	T	-	-	T
EC COL	T	T	T	-	-	-	-	-	-	-	-	-	-	-	-

Table 10: Testing Alternate Hypothesis that the classifier in the row index is better than the classifier in column index. First table gives the p-values and the second table gives the result of the test ($\alpha = 0.05$). "T" indicates cases where there is enough evidence to reject the null hypothesis that the test loss of both classifiers are the same

Results

Looking at the confidence intervals, it seems that the bagging ensemble of decision tree regressors (the ones trained on or only on chemical composition) perform better than every other regressors and the bagging ensemble of decision tree classifiers (again trained on or only on chemical composition) are better than every other classifier. Here by "chemical composition" we mean every feature in the dataset except wine color.

Quantitatively, there was less than 5% chance that ensemble regressor (classifier) performs similar to the other regressors (classifiers). So we can safely reject the null hypothesis and conclude that the ensemble estimators might actually be better than the other estimators (among those considered).

In a similar manner, we can see that all regressors (classifiers) —when trained with chemical composition— perform better than the dummy regressor (classifier).

From the results of the hypothesis test, we can state that all regressors (except dummy) that are trained with all variables are better than the all regressors trained with only the color of the wine. Infact, those trained with only chemical composition are also always better than those trained with only color of the wine. The previous statements are true for more than 95% of the time. Similarly, all classifiers trained with only chemical composition or both chemical composition and color are always better than those trained with only color. Again this statement holds true for more that 95% of the time.

So every estimator with chemical composition added to the list of features perform better than without. So, we can safely conclude that chemical composition adds predictive power above wine color.

However, every estimator trained with all features seem to be performing similar to those trained without the wine-color feature. The p-values are quite high (refer table 8 and 10) indicating high risk in rejecting the null hypothesis. So, we conclude that we don't have sufficient evidence to reject the null hypothesis which claims that wine color does not add predictive power over chemical composition.

To conclude,

- The bagging ensemble regressor can predict wine-quality with generalization (squared-error) loss ≈ 0.38. The bagging ensemble classifier can predict wine-quality with generalization (0/1) loss ≈ 0.33 [Note: These 2 losses are not comparable]. That is how well wine-quality can be predicted from chemical composition and color. If this loss is acceptable for the target application, then and only then we can conclude that "wine quality can be predicted from chemical composition and color".
- There is sufficient evidence to conclude that chemical composition adds predictive power above wine color.
- Wine color doesn't seem to add predictive power above chemical composition.

5 Answering the Questions of Interest

Here we would be answering the rephrased questions from Task A1. Specifically we would be answering the "questions to address" listed for each question in Task A1.

Can we use the sommelier/wine data to create an AI with super-human performance in wine tasting?

It is theoretically possible to use the wine data to train a supervised-learning model to predict the quality of a wine based on its features (the same features in the wine data set) provided that this new wine comes from the same distribution as that of the wine in the training data (wine dataset). This trained model can be used to predict the wine-quality, as would have been rated by the median expert - as that's how the model would be trained. With a reasonably fast computer, assuming all feature values (chemical composition and color) of the wine samples (to be rated) are already available, the model can be used to predict the wine quality ("taste the wine" - if you prefer) at super-human speeds without getting drunk.

However, the trained model might not be as good as the experts that the model is trying to imitate (because it indeed is an imitation). Based on the comparisons in Task A4, the best regressor has a generalization error (squared loss) of 0.38 and the best classifier has a generalization error (0/1 loss) of 0.33. If these errors are suitable for the application, then we can conclude that we can create an AI with super human (in the previously defined sense) performance in wine-tasting (rating imitation).

Which components of a wine make a wine a good wine?

A model specific inference using a trained model might answer this question –when the components in the question are limited to the features available in the training set– at some quantified level of accuracy. However, we observed that the white-box models we had in our model comparison set-up did not perform as well as the best among the models considered. So we resorted to model agnostic (naive) hypothesis test with the (controversial) assumption that the components of the wine are independent of each other in their contribution to the quality of the wine.

We did (unpaired-z) hypothesis tests independently comparing the wines with rating 7 or more with the other wines. The results are available in table 11. With the usual caveats in interpreting hypothesis tests, we can see that there is no significant difference between the good and other wines for most features. The interaction between features might lead to differences; however, such differences might not be visible in our test because of our independence assumption.

So, based on the results of the test, there is sufficient evidence to conclude that the (independent) components characteristics of a wine that are associated with a good wine are

- High alcohol content
- Low density

Important: Note that this is a conclusion of association and not a conclusion of causal linkage. They don't really make a wine a good wine; they are simply observed in good wines.

Can we use AI to create the perfect wine (whose quality exceeds all that we have seen)

No.

Roughly, Supervised Learning works by finding the parameters of the models that fits the training data well. By the very nature of this training, they are capable of generalizing to the test data only if the test data comes from the same distribution as that of the training data. That is so because the model is trained by minimizing the estimate of generalization error, which is

feature	G>O: p value	G>O: Result	G<O: p value	G<O: Result
quality	0.0000	T	1.0000	-
fixed acidity	0.6739	-	0.3261	-
volatile acidity	0.9055	-	0.0945	-
citric acid	0.3193	-	0.6807	-
residual sugar	0.7128	-	0.2872	-
chlorides	0.9165	-	0.0835	-
free sulfur dioxide	0.4485	-	0.5515	-
total sulfur dioxide	0.6727	-	0.3273	-
density	0.9955	-	0.0045	T
pH	0.3999	-	0.6001	-
sulphates	0.3770	-	0.6230	-
alcohol	0.0000	T	1.0000	-

Table 11: Independent Comparison of features between good (7,8,9 rating) and other wine. G¿O indicates the p values and the result for the alternate hypothesis that the feature is higher in good wine. G¡O indicates the alternate hypothesis that the feature is lower in good wine."T" indicates that there is sufficient evidence to reject the null hypothesis that the features are equal.

defined in a distribution specific manner. By this limitation, the trained model is theoretically incapable (no guarantee) of even predicting the perfect wine (not in training data) as being better than (high rating) all the other wines it has previously seen.

Viewing the same problem in another (Bayesian) way; if we try to find the solution (combination of features) using (for eg.) optimization routines, the solution space could be multimodal and there could not be a single perfect wine. Assuming there is a global optimum and that we used a "magic" optimizer (along with a model that has a functional form) that could find this optimum, we could end up having feature values that are well out of the range observed in the training set. In the Bayesian analysis of learning, while looking at the predictive distribution, we can observe that the variance of the prediction is very high as we move far away form the location of training points in the feature space. This again makes our conclusion questionable. One way to interpret the variance is that the model is uncertain how to imitate the expert for feature values in the unseen region of the feature space.

Again the "invertibility" of our model is uncertain as the wine data (training data) doesn't seem to be collected based on a properly designed experimental setup that could facilitate causal conclusions. Without causal relationships, we can't even recreate a wine in the training set as the actual cause for wine quality might be unknown (or simply not recorded).

... what would it be that AIs could, or would learn from humans?

As touched upon in the previous answers, the model simply learns to imitate the mean expert for the specific task of wine-rating. It doesn't really develop a taste for wine. In a numerical sense, it can make novel predictions for wine from unseen feature space; but, most probably, it's just a bad prediction of how a human expert might rate such a wine.

To conclude, they learn to imitate the humans (in this case as the training data is related to human response) in an extremely narrow sense, where the narrowness is determined by the interplay between the location of the training data and the complexity of the model considered.

A task_a2.py: Loading the Dataset

```
'''
This file contains code for importing data
'''
# necessary modules
import os
import pandas as pd


def get_data(folder, colors=("white", "red")):
    '''
    locates the following files ( by default) in the folder taken as argument

    winequality-white.csv
    winequality-red.csv

    The data on quality of red-wine and white-wine are aggregated
    into a super data set. This is returned as a pandas dataframe

    Optionally colors can also be given as an argument
    '''
    # initialize pandas df to hold the dataset
    wine_df = pd.DataFrame()

    # read data for each color and append
    for color in colors:
        # get the file paths
        file_path = os.path.join(folder,
                                 "winequality-"+color+".csv")

        # import the CSVs as pandas dataframe
        temp_df = pd.read_csv(file_path, sep=";")

        # add the color variable
        temp_df.loc[:, "color"] = color

        # append to the existing datatrame
        wine_df = wine_df.append(temp_df,
                                 ignore_index=True,
                                 verify_integrity=True)

    # rearrange to make quality the first column
    wine_df = wine_df[["quality"] +
                      [col for col in wine_df if col != "quality"]]

    # return the aggregated dataframe
    return wine_df


if __name__ == "__main__":
    '''
    run testing code if ran as a script
    '''
    WINE_DF = get_data("winequality")

    # check shape
    if WINE_DF.shape == (6497, 13):
        print("The shape of the DF is as expected")
    else:
        print("Shape test FAILED")

    # check red count
    if WINE_DF.color[WINE_DF.color == "red"].count() == 1599:
        print("Red wine count is as expected")
    else:
        print("Red count test FAILED")

    # check white count
    if WINE_DF.color[WINE_DF.color == "white"].count() == 4898:
        print("White wine count is as expected")
```

```
    else:
        print("White count test FAILED")
```

B task_a3.py: Exploratory Data Analysis

```
'''
This file was actually converted from a notebook
contains code for EDA
'''
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
from tabulate import tabulate
import task_a2
import task_a5


def main():
    '''
    main function like in C
    '''
    # toggle for saving (and showing) images
    save = False

    # get the wine data frame (wdf)
    wdf = task_a2.get_data("winequality")

    # Numerical Summaries
    header = [" ", "count", "mean", "std", "min", "25%", "50%", "75%", "max"]
    print(tabulate(wdf.describe().T, headers=header, tablefmt="latex"))

    # plot histograms
    if save is True:
        plt.figure(figsize=(20, 5))
        filter_list = wdf.color != ""
        for i, w_color in enumerate(["", "red", "white"]):
            if i > 0:
                filter_list = wdf.color == w_color
            plt.subplot(1, 3, i+1)
            plt.hist(wdf[filter_list].quality,
                     bins=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                     align="left")
            plt.title(str(w_color)+" wine quality histogram")
            plt.xlabel("Wine Quality")
            plt.ylabel("Count")
        plt.savefig("histogram.jpg")
        plt.show()

    # Make bar plot
    if save is True:
        i = 1
        fig = plt.figure(figsize=(10, 10))
        for feature in wdf:
            if feature != "color":
                ax = fig.add_subplot(4, 3, i)
                ax.boxplot([wdf[feature][wdf.color == "red"],
                            wdf[feature][wdf.color == "white"]])
                plt.setp(ax, xticklabels=['red', 'white'])
                ax.set_ylabel(feature)
                i += 1
        plt.tight_layout()
        plt.savefig("barplot.jpg")
        plt.show()

    if save is True:
        # matrix scatter plot
        # color coding for wine
```

```
        color_dict = {"red":"red", "white":"blue"}
        # getting the color list
        color_list = [color_dict[c] for c in wdf.color]
        # matrix scatter plot
        scatter_matrix(wdf, alpha=0.01, figsize=(20, 30),
                       diagonal='density', color=color_list)
        plt.savefig("mat_scatter.jpg")
        plt.show()

    # log transformation
    transform_list = [
        'fixed acidity',
        'citric acid',
        'volatile acidity',
        'residual sugar',
        'chlorides',
        'free sulfur dioxide',
        'total sulfur dioxide',
        'sulphates',
        ]
    print("Least count")
    np.min(wdf["citric acid"][wdf["citric acid"] > 0])
    lwdf = wdf.copy()
    # put a shift (0.5*leastcount) on citric_acid (10e-2 is the typical LC)
    lwdf["citric acid"] = lwdf["citric acid"]+10e-2*0.5
    lwdf[transform_list] = np.log(lwdf[transform_list])
    print(tabulate(lwdf.describe().T, headers=header, tablefmt="latex"))

    if save is True:
        # log - matrix scatter plot
        # color coding for wine
        color_dict = {"red": "red", "white": "blue"}
        # getting the color list
        color_list = [color_dict[c] for c in lwdf.color]
        # matrix scatter plot
        scatter_matrix(lwdf, alpha=0.01, figsize=(20, 30), diagonal='density', color=
            color_list)
        plt.savefig("log_mat_scatter.jpg")
        plt.show()

    # rough calculations for quantitative EDA
    from sklearn.metrics import silhouette_score
    print("Red wine mean quality: ", np.mean(wdf["quality"][wdf.color == "red"]))
    print("White wine mean quality: ", np.mean(wdf["quality"][wdf.color == "white"]))
    print("Red wine with quality more than 8: ", len(wdf["quality"][np.logical_and(wdf.
        quality > 8, wdf.color == "red")]))
    twdf = pd.get_dummies(wdf)
    print("Silhoutte score as per color labels", silhouette_score(twdf.iloc[:,0:-2], twdf
        .iloc[:,-1]))
    print("Silhoutte score as per random labels", silhouette_score(twdf.iloc[:,0:-2], np.
        random.randint(0,2, size=(twdf.shape[0]))))

    result = []
    header = [" "]
    for i in lwdf.columns[1:-1]:
        buf = [i]
        header.append(i)
        for j in lwdf.columns[1:-1]:
            buf.append(np.corrcoef(lwdf[i], lwdf[j])[1,0])
        result.append(buf)
    print(tabulate(result, headers=header, tablefmt="latex", floatfmt="0.3f"))

    # comparing red and white wine
    task_a5.do_htest(wdf[wdf.color == "red"], wdf[wdf.color == "white"], label1="R>W:",
        label2="R<W:")

if __name__ == "__main__":
    main()
```

C task_a4.py: Benchmarking Experiments

```
'''
run this module as a script to conduct benchmarking experiments
also performs hypothesis tests to find the best model

note:
    1. regressors and classifiers are compared separately
    2. all estimators that need hyper-parameter tunigh are wrapped
       with GridSearchCV (with refit=True) and the whole strategy is
       validated in the outer loop
    3. more details in the report
'''
import os

# data manipulation utilities
import pickle
import pandas as pd
from tabulate import tabulate

# numeric computations
import numpy as np

# sklearn utilities and estimators
from sklearn import metrics
from sklearn.utils import shuffle
from sklearn.model_selection import GridSearchCV, KFold, cross_validate
from sklearn.dummy import DummyRegressor, DummyClassifier
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.svm import SVR, SVC
from sklearn.neural_network import MLPRegressor, MLPClassifier
from sklearn.ensemble import BaggingRegressor, BaggingClassifier

# stat utils
from scipy.stats import norm

# reusing code from other tasks
import task_a2


def get_estimators(variety):
    '''
    all estimators under consideration are defined here
    returns regressors or classifiers as per request

    variety argument takes "regressor" or "classifier" as values
    '''
    # check validity of input argument
    if variety != "regressor" and variety != "classifier":
        raise ValueError("Unrecognized variety of estimator")

    # initialize list of regressors
    r_est = []  # regressors
    c_est = []  # classifiers

    # Dummy Regressor
    combo = DummyRegressor()
    r_est.append((combo, "DR"))

    # Linear Regressor
    param_grid = {
        "fit_intercept": [True, False]
        }
    combo = GridSearchCV(LinearRegression(),
                         param_grid,
                         refit=True,
                         scoring="neg_mean_squared_error",
                         n_jobs=-1)
    r_est.append((combo, "LR"))

    # SVR
```

```
    param_grid = {
        "C": [0.1, 1, 10],
        }
    # uses RBF kernel by default
    combo = GridSearchCV(SVR(),
                         param_grid,
                         refit=True,
                         scoring="neg_mean_squared_error",
                         n_jobs=-1)
    r_est.append((combo, "SVR"))

    # neural_network Regressor
    param_grid = {
        "hidden_layer_sizes": [(32, 32,), (16, 16, 16,)],
        "learning_rate_init": [0.01, 0.001]
        }
    combo = GridSearchCV(MLPRegressor(),
                         param_grid,
                         refit=True,
                         scoring="neg_mean_squared_error",
                         n_jobs=-1)
    r_est.append((combo, "NNR"))

    # Ensemble Regressor
    combo = BaggingRegressor(base_estimator=None,  # uses trees by default
                             n_estimators=20,  # atleast 10 as per instructions
                             n_jobs=-1)
    r_est.append((combo, "ER"))

    # Dummy Classifier
    combo = DummyClassifier()
    c_est.append((combo, "DC"))

    # Logistic Classifier
    param_grid = {
        "fit_intercept": [True, False],
        "C": [0.1, 1, 10]
        }
    combo = GridSearchCV(LogisticRegression(),
                         param_grid,
                         refit=True,
                         n_jobs=-1)
    c_est.append((combo, "LC"))

    # SVC
    param_grid = {
        "C": [0.1, 1, 10]
        }
    # uses RBF kernel by default
    combo = GridSearchCV(SVC(),
                         param_grid,
                         refit=True,
                         n_jobs=-1)
    c_est.append((combo, "SVC"))

    # neural_network classifier
    param_grid = {
        "hidden_layer_sizes": [(32, 32,), (16, 16, 16,)],
        "learning_rate_init": [0.01, 0.001]
        }
    combo = GridSearchCV(MLPClassifier(),
                         param_grid,
                         refit=True,
                         n_jobs=-1)
    c_est.append((combo, "NNC"))

    # Ensemble Classifier
    combo = BaggingClassifier(base_estimator=None,  # uses trees by default
                              n_estimators=20,  # atleast 10 as in instructions
                              n_jobs=-1)
    c_est.append((combo, "EC"))

```

```
    # group estimators and return as per variety
    if variety == "regressor":
        return r_est
    return c_est


def cond_mean(truth, preds):
    '''
    custom scorer
    gives mean of squared error conditioned on training split
    '''
    sq_diff = (truth - preds)**2
    return np.mean(sq_diff)


def cond_var(truth, preds):
    '''
    custom scorer
    gives variance of MSE conditioned on training split
    '''
    sq_diff = (truth - preds)**2
    return (np.var(sq_diff, ddof=1))/truth.shape[0]


def cond_01_mean(truth, preds):
    '''
    custom scorer
    gives mean of squared error conditioned on training split
    '''
    loss = (truth != preds).astype(float)
    return np.mean(loss)


def cond_01_var(truth, preds):
    '''
    custom scorer
    gives variance of MSE conditioned on training split
    '''
    loss = (truth != preds).astype(float)
    return (np.var(loss, ddof=1))/truth.shape[0]


def get_estimator_performance(e_dat_list, score_dict, cv_split):
    '''
    e_dat: dict of estimator and data
    score_dict: scores for cval
    cv_split: consistent split for cval

    returns the score as tuples for all e_dat
    '''
    rval = []

    for e_dat in e_dat_list:
        print("Training: "+e_dat["name"])
        buff = {}
        score = cross_validate(e_dat["estimator"],
                               e_dat["x data"],
                               e_dat["y data"],
                               scoring=score_dict,
                               return_train_score=False,
                               cv=cv_split)
        # fill the results buffer
        buff["name"] = e_dat["name"]
        # the means are still estimates conditional on
        # training set (but they have low variance)
        buff["score"] = [np.mean(score["test_"+str(k)]) for k in score_dict]
        rval.append(buff)
    return rval


def save_score(fname):
    '''
```

```
    the main function:
    speaks for itself
    '''

    print("Importing Data...")
    wdf = task_a2.get_data("winequality")

    # log transform
    transform_list = [
        'fixed acidity',
        'citric acid',
        'volatile acidity',
        'residual sugar',
        'chlorides',
        'free sulfur dioxide',
        'total sulfur dioxide',
        'sulphates',
    ]
    lwdf = wdf.copy()
    # put a shift (0.5*(leastcount=10e-2)) on citric_acid
    lwdf["citric acid"] = lwdf["citric acid"]+10e-2*0.5
    lwdf[transform_list] = np.log(lwdf[transform_list])
    wdf = lwdf

    print("Processing Data...")
    # shuffle data
    wdf = shuffle(wdf)
    # get dummy variables
    wdf = pd.get_dummies(wdf)
    # reuse the same CV split for all models
    cv_split = KFold(n_splits=5)
    # separate features and targets
    y_data = wdf["quality"]
    x_data = wdf.iloc[:, 1:]

    # for both regression and classification
    for kind in ["classifier", "regressor"]:
        print("Seting up the necessary "+kind+"s...")
        ests = get_estimators(kind)

        # create a dictionary of scorers
        if kind == "regressor":
            score_dict = {
                "mean": metrics.make_scorer(cond_mean),
                "var": metrics.make_scorer(cond_var)
                }
        else:
            score_dict = {
                "mean01": metrics.make_scorer(cond_01_mean),
                "var01": metrics.make_scorer(cond_01_var)
                }
        # column masks for features
        fea_masks = ["", "CHM", "COL"]

        # list to hold estimators and corresponding data
        test_list = []
        for est in ests:
            for mask in fea_masks:
                buff = {}
                buff["name"] = est[1]+" "+mask
                buff["estimator"] = est[0]
                if mask == "":  # all features
                    buff["x data"] = x_data
                elif mask == "CHM":  # only chemical composition
                    buff["x data"] = x_data.iloc[:, 0:-2]
                elif mask == "COL":  # only color
                    buff["x data"] = x_data.iloc[:, -2:]
                else:
                    raise ValueError("Unknown mask specification")
                buff["y data"] = y_data
                test_list.append(buff)

```

```
        score = get_estimator_performance(test_list,
                                          score_dict,
                                          cv_split)
        # save scores to file
        with open(kind+"."+fname, 'wb') as handle:
            pickle.dump(score, handle)


def tabulate_results(fnames):
    '''
    reads the files and tabulates results
    this is not a reusable function
    many stuffs are hard coded
    '''
    for fname in fnames:
        if fname == "classifier.score":
            print("Comparison of Regressors\n")
            # prep for 95% CI calculation
            alpha = 0.05
            z_val = norm.ppf(alpha/2)
            # read scores from file
            with open(fname, 'rb') as handle:
                score = pickle.load(handle)
            table = []
            for dic in score:
                table.append(
                    [dic["name"],
                     *dic["score"],
                     "(" +
                     f"{dic['score'][0]+z_val*np.sqrt(dic['score'][1]):.4f}" +
                     ", " +
                     f"{dic['score'][0]-z_val*np.sqrt(dic['score'][1]):.4f}" +
                     ")"])
            print(tabulate(table,
                           headers=["Classifier",
                                    "Mean(0/1 loss)",
                                    "Var(Mean(0/1 loss))",
                                    "95% Confidence Interval"],
                           tablefmt="latex",
                           floatfmt=".6f"))
            print("\n\n")
        if fname == "regressor.score":
            print("Comparison of Regressors\n")
            # prep for 95% CI calculation
            alpha = 0.05
            z_val = norm.ppf(alpha/2)
            # read scores from file
            with open(fname, 'rb') as handle:
                score = pickle.load(handle)
            table = []
            for dic in score:
                table.append(
                    [dic["name"],
                     *dic["score"],
                     "(" +
                     f"{dic['score'][0]+z_val*np.sqrt(dic['score'][1]):.4f}" +
                     ", " +
                     f"{dic['score'][0]-z_val*np.sqrt(dic['score'][1]):.4f}" +
                     ")"])
            print(tabulate(table,
                           headers=["Regressor",
                                    "Mean(SE)",
                                    "Variance(MSE)",
                                    "95% Confidence Interval"],
                           tablefmt="latex",
                           floatfmt=".6f"))
            print("\n\n")


def get_p_table(fname):
    '''
    this function is only for regressor scores
```

```
    it does the following
    1. reads the given file
    2. creates a table with p-value comparing every model
       with every other model
    3. p values are calculated using a std.normal as null distribution
       and by taking 1-cdf(diff.means/sqrt(sum(var_means)))
    '''
    if fname == "regressor.score":
        print("\n\nComparing Regressors")
    elif fname == "classifier.score":
        print("\n\nComparing Clasifiers")
    else:
        raise ValueError("cant process this file")

    # hypothesis testing parameters
    alpha = 0.05
    print("Testing Hypotheses with alpha = "+str(alpha))
    print("Alternative hypothesis is loss of row index < loss of column index")

    with open(fname, 'rb') as handle:
        score = pickle.load(handle)

    # initialize the result list
    result1 = []
    result2 = []
    header = [" "]

    for dic1 in score:
        # initialize the outer loop buffer
        out_buff1 = []
        out_buff2 = []
        out_buff1.append(dic1["name"])
        out_buff2.append(dic1["name"])
        for dic2 in score:
            # find p value
            h_mean = dic2["score"][0] - dic1["score"][0]  # subtract mean
            h_sd = np.sqrt(dic1["score"][1] + dic2["score"][1])  # add var
            p_val = 1-norm.cdf(h_mean/h_sd)  # as mean as per H0 is 0
            # test hypothesis
            if p_val < alpha:
                h_result = "T"
            else:
                h_result = "-"
            in_buff1 = f"{p_val:.2f}"
            in_buff2 = h_result
            # add result to outer buffer
            out_buff1.append(in_buff1)
            out_buff2.append(in_buff2)
        # add out buffer to result
        result1.append(out_buff1)
        result2.append(out_buff2)

        # fill the header
        header.append(dic1["name"])

    print(tabulate(result1, headers=header, tablefmt="latex"))
    print(tabulate(result2, headers=header, tablefmt="latex"))


if __name__ == "__main__":
    if not (os.path.exists("classifier.score") and
            os.path.exists("regressor.score")):
        save_score("score")

    # report results
    tabulate_results(["classifier.score", "regressor.score"])

    # perform hypothesis testing
    get_p_table("regressor.score")
    get_p_table("classifier.score")
```

D task_a5.py: Inference

```python
'''
This file contains code for task a5
'''
import numpy as np
from tabulate import tabulate
from scipy.stats import norm
import task_a2


def do_htest(dfa, dfb, label1, label2):
    '''
    a and b are dataframes with same column headings
    performs t-test comparing the two samples separately
    '''
    alpha = 0.05
    headers = ["feature", label1+" p value", label1+" Result", label2+" p value", label2+
        " Result"]
    result = []

    for feature in dfa.columns[0:-1]:
        x1 = np.mean(dfa[feature])
        x2 = np.mean(dfb[feature])
        v1 = np.var(dfa[feature], ddof=1)/dfa.shape[0]
        v2 = np.var(dfb[feature], ddof=1)/dfa.shape[1]
        # z-statistic
        t = (x1-x2)/np.sqrt(v1+v2)
        # p-value for 1 > 2
        p_val_1g2 = 1 - norm.cdf(t)
        if p_val_1g2 < alpha:
            h_val_1g2 = "T"
        else:
            h_val_1g2 = "-"
        # p-value for 1 < 2
        p_val_2g1 = norm.cdf(t)
        if p_val_2g1 < alpha:
            h_val_2g1 = "T"
        else:
            h_val_2g1 = "-"
        result.append([feature, p_val_1g2, h_val_1g2, p_val_2g1, h_val_2g1])

    print(tabulate(result, headers=headers, floatfmt="0.4f", tablefmt="latex"))

if __name__ == "__main__":
    # get the wine data frame (wdf)
    wdf = task_a2.get_data("winequality")
    # find distinguishing features of good wine
    do_htest(wdf[wdf.quality > 6], wdf[wdf.quality <= 6], label1="G>0:", label2="G<0:")
```