Abstract

In this paper we will be considering the problem of predicting the quality of a wine based on various measurable physical characteristics of the wines. We will do this in two ways, first we will set a (somewhat arbitrary) cutoff defining a wine as either good or bad, allowing us to view the problem as a binary classification problem. Next, we will attempt to use regression to predict the quality of the wine based on a 1 to 10 scale.

Section I - Introduction

The data set we will be using comes from Paulo Cortez at the University of Minho, Guimaraes, Portugal and contains 1,599 different wines. For each wine the attributes, shown in table one, are given along with a quality score from 1 to 10. The wines are all red variants of the Portuguese "Vinho Verde" wine, there is no data however on the specific grape varietals used or any non-physical characteristics of the wine (such as the price, brand, location, etc.).

Attribute	Description
Fixed Acidity	Tartaric Acid (g/dm^3)
Volatile Acidity	Acetic Acid (g/dm^3)
Citric Acid	Citric Acid (g/dm^3)
Residual Sugar	Residual Sugar (g/dm^3)
Chlorides	Sodium Chloride (g/dm^3)
Free Sulfur Dioxide	Unbound Sulfur Dioxide (mg/dm^3)
Total Sulfur Dioxide	Sulfur Dioxide (mg/dm^3)
Density	Density (g/cm^3)
pH	pH
Sulphates	Potassium Sulphate (g/dm^3)
Alcohol	(vol %)

Table 1 Physical Attributes

Being able to accurately predict the quality of a wine solely based on measurable physical characteristics of a wine could help vineyards decide which candidate crops should see an increase in production with more concrete data than just internal testing and at lower cost than a large-scale external test. In addition, it could help vineyards better predict what the optimal price for their wine would be. The data we have available does not include any pricing information so we will be focused solely on the quality score prediction, however using this model a vineyard could supplement it with pricing/sales data to provide additional insight.

In this paper we will step through the process we used to explore the data, what potential errors we looked for and how we would remedy them, the success of the models used for binary classification of the wines as either "good" or "bad", and the success of the models used for predicting the quality score from 1 to 10.

Section II – EDA

We made the assumption that the data collection process was sound and did not question the manner in which it was accumulated and recorded. That part of the process is outside the scope of both this assignment and class. Furthermore, the dataset has been widely used by many others which further supports our assumption. The records were already in a single CSV file format, so we didn't need to combine any files. This made it very easy to import using the pandas API in Python. By bringing in the file as a pandas dataframe, we were able to manipulate and explore very quickly. We found that there were 1,599 entries each representing a different wine variety. For each one, there were 12 columns of information, 11 which were features and one final “target” column indicating quality.

Each of the feature columns was a float datatype, which is important for the algorithms later. Since none of the features were categorical, or even binary, we were enthusiastic that some good patterns would emerge. The target column of quality was in an integer format, theoretically from 0-10. However, we observed only records from 3-8. This made the data relatively concentrated near the center, which creates a challenge in terms of classing wines. In fact, the distribution centers heavily around those wines at either a 5 or a 6. So even if a classification algorithm were to be very close and guess a 6, when in reality the wine was a 5, that would adversely impact accuracy.

The data itself is very complete and workable. None of the columns had missing values or excessive zeros. Generally, excessive zeros indicate that a record is missing and was just given a ‘0’. We tested this formulaically with a simple python script. By simply summing up the total records for each feature with a zero, we could quickly see the results. We found sort of an odd outcome: one column had about 10% of the records with a zero for that field. This is unusual for the dataset at large since not one other column had a zero. We did not see any missing values either, so for there to be a 10% error in only field did not seem right. In order to help gain comfort, we did some research and found that there are indeed wines that contain zero citric acid. Individuals who are sensitive to acidity may select one of these alternatives to help alleviate the problem. This is more of a unique situation, the exception rather than the norm. Thus if 10% of our data had zero citric acid, this is most likely a valid result. Therefore, we have decided to keep all the data without making alterations for these zeros.

Continuing on with our data check, we investigated the possibility of outliers. Here we wanted to again see if any data had been erroneously entered or recorded and thus skew the results. Given we have a smaller set of records to work with, the presence of outliers could have a greater impact on the models. As we had the luxury of each feature column being numeric, we ran a box-and-whisker plot for each one and visually inspected each one. While most fit the classic structure, we did see some that, at first glance looked like there may be outliers. However, when we considered the scale of each one, we did not conclude that there were any mis-recorded data points. For example, consider the results of one of the attributes, chloride, shown below in Figure 1. While it may first appear that there are multiple outliers, the scale shows how close these entries are. We see this as a valid set and do not intend to remove any potentially outliers. We believe that keeping the records is more beneficial.

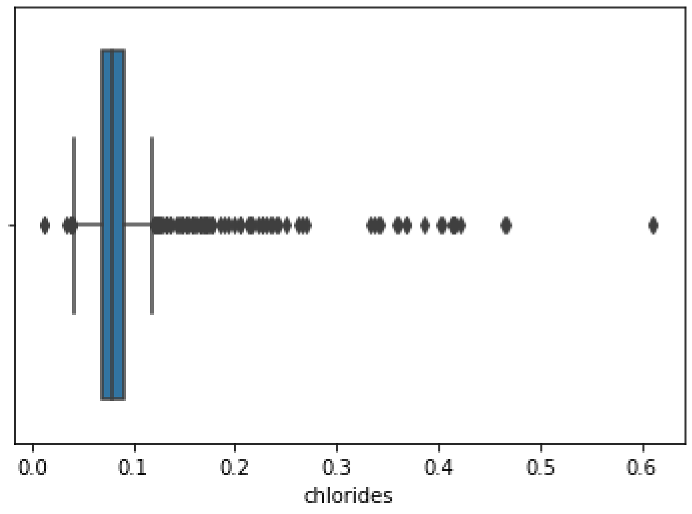

Figure 1: chlorides

We also checked for duplicate records, again using Python, and unsurprisingly found none. After going through these different tests, we have gained a great deal of comfort with the dataset and its accuracy.

Next, to help guide our approaches, we ran a general correlation matrix on the entire set. Here we employed some of the built-in functions of pandas, such as the correlation, and plotted the results. Similar to inspecting the outliers visually, it was helpful to see where there may be some correlations to investigate. Had our data and defined problem been a bit more ambiguous, we would have dug in further here. Fortunately, we already had a clear target variable so we wanted to start working with the most meaningful relationships. We used pandas again to check the correlation of each feature with the target (quality). We saw some relatively strong correlations, with alcohol at almost .5 (out of 1). There were also some negative correlations as well that we want to include as well. Our big takeaway was that we had some correlations to work with, but none that were too strong. With highly correlated data, the risk of having a poorly trained model becomes greater. After checking, we do feel that any columns should be removed due to an overly high correlation.

Section III – Binary Classification

Our first goal was to simply classify a wine as either "good" or "bad". To create a binary target, we split the wines roughly in half: those with a recorded quality of less than 6 were considered bad, and those with 6 or greater were considered good. Our new field we labeled as "qualitybinary", and the distribution, shown below, gave us a nice and balanced target.

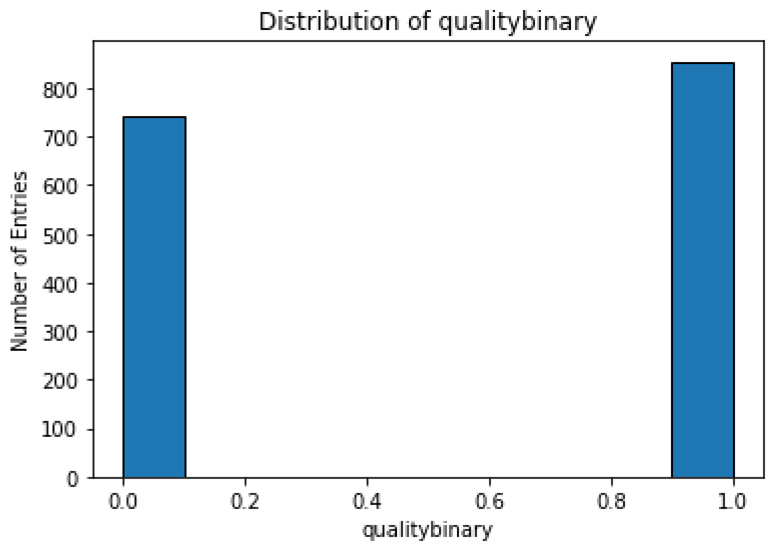

Figure 2: Quality distribution

On the other hand, as mentioned above, most records were either a 5 or 6. So the differences between the two are very slim. As such, this will make for a greater challenge for the algorithms.

The most important thing we tried to control for was consistency across approaches. By standardizing the train/test split, and the way the functions interact with the data, we can be sure to capture consistent results. We set up a “driver” function where we can pass a model and get back the relevant metrics. This also allows for easier document flow.

Our binary classification approaches were Naive Bayes, Decision Tree, Gradient Boosting, and Random Forest. We have discussed all of these throughout the class, and all are appropriate for this type of data. We wanted to test a couple of different things here to try and find the optimal model. Looking at hyperparameters is of course one aspect, but we did some specific tuning that fit with each algorithm.

Beginning with the simplest approach, Naive Bayes, we gave ourselves a baseline. Because Naive Bayes is the least sophisticated algorithm, we assumed it to give the least accurate results. If we use the most basic classification algorithm with no tuning, then each of the next models should do better. If not, then something is certainly wrong and would cause for additional investigation. We also chose to assume Gaussian distribution, as that is statistically the most likely outcome. Additionally, every single feature we tested was numeric. So Gaussian Naive Bayes is the most appropriate choice. Without any tuning, we achieved ~70.4% accuracy, which is decent, but also ended up being the worst performing algorithm. In this way, it achieved exactly what we wanted by providing a simple baseline.

Next, we ran a decision tree classifier. This is also a relatively unsophisticated algorithm, but allows for lots of hyperparameter tuning. In order to optimize this approach, we ran a cross-validation grid search. By creating a pipeline of variables to test, we could allow the cross-validation score to determine which ones to ultimately use. We focused on criterion (either gini or entropy) and the max depth. The result of some heavy computations indicated that gini and a max depth of 6 yielded the best results. Indeed, a decision tree with these hyperparameters beat the Naive Bayes with a ~71.4% accuracy. We would like to note that the computational expense of achieving this optimized approach is almost not worth the small improvement in accuracy.

Moving forward with more sophistication, we tested a variant of gradient boosting next. The gradient boosting algorithm also allows for much tuning, and is also a good candidate for a grid search. Given the low improvement on the decision tree grid search, we decided to take a different approach. There is a “pre-tuned” version of gradient boosting available known as XGBoost (which stands for Extreme Gradient Boosting). Here we have a canned algorithm that should perform better than a stock gradient boosting approach, but perhaps not quite as well as a full grid search. The tradeoff is that it is very quick to implement, and gave excellent results. Without touching a single hyperparameter, the XGBoost was able to achieve ~77.1% accuracy. This far exceeds both of the other approaches, while also saving on computational time.

Finally, we tried one last approach, the random forest. Another sophisticated algorithm, the random forest is actually a collection of decision trees, so we wanted to see how it perform against our previously implemented decision tree. To tune, we focused on (arguably) the most important factor, n_estimators (or number of trees). We did a mini grid-search and iterated through 20 different estimators and kept track of the best one. For our dataset, this turned out to be 14, which gave us the best accuracy of any binary classifier we tried: ~78.5%. While more computationally expensive than the XGBoost, it was still much faster than running the full grid search for the decision tree. Our loop took less than a second to run, while the grid search took over 30 seconds of near full CPU running time.

Our takeaway from this portion of the exercise was that the more sophisticated algorithms generally perform better. Also, it is most likely worth tuning some hyperparameters in an effort to maximize accuracy, provided the resources are available. Regarding a benchmark, the XGBoost approach is just so quick and easy to implement that we believe that would be a better benchmark going forward. It takes no more time than a simple Naive Bayes algorithm, yet yields substantially better results. If a user can find a combination of hyperparameters for a different algorithm that beats it, then that is a good indicator of success.

Being able to classify wines as either “good” or “bad” with accuracy in the high 70’s is a decent result. However, keeping in mind that the boundary line is actually the difference between a 5 or 6 out of 10, we believe these results are slightly more impressive. Overall, we are satisfied with our binary classification approach.

Section IV - Regression

For the regression portion of our analysis we looked at the following algorithms: Linear Regression, Random Forest, and a Decision Tree. Initially we just looked at the algorithms varying the minimum correlation for input attributes with our target variable. After doing this we selected the most promising algorithm and did a more thorough hyper-parameter tuning to try and find a more accurate model. I will briefly describe each of the algorithms used and the hyper-parameters we would have looked at tuning if that model had been down selected.

Because the quality scores are not equally distributed across the band of 0 to 10, we felt it would be useful to get a baseline result which we can achieve by simply always estimating the mean value of the distribution, 5.64. When doing this we get a root mean square error (RMSE) of 0.81 and a mean absolute error (MAE) of 0.68.

Linear Regression attempts to fit a line that minimizes some cost function based on the errors, assuming that the relationship between the input attributes and the target is linear. For our analysis we used mean square error as this cost function, however we also evaluated the mean absolute error in order to get a better understanding of the distribution of errors. Other than the error function the other aspect that we can use to tune our linear regression model is the features that are used for the analysis. We looked at three different sets of features which were based on the correlation with the target variable and are shown in Table 2.

Attributes	Root Mean Square Error	Mean Absolute Error	Accuracy
All	0.68	0.52	59.2%
Volatile Acidity Citric Acid Sulphates Alcohol	0.68	0.53	56.3%
Volatile Acidity Alcohol	0.69	0.54	56.0%

Table 2 Linear Regression Results

Next, we looked at decision tree regression which works by splitting the dataset into separate nodes which are similar to each other, attempting to reduce the impurity. Once we get to our final leaf nodes the predicted value for that leaf will be the average quality score of all of the entries in that leaf. We used scikit-learn's default decision tree regression settings, and again used different subsets of features to get our estimates. Table 3 shows our results. We see that in terms of RMSE these decision trees performed worse than our naïve estimate of always choosing the overall mean.

Attributes	Root Mean Square Error	Mean Absolute Error	Accuracy
All	0.84	0.52	56.7%
Volatile Acidity Citric Acid Sulphates Alcohol	0.84	0.51	57.5%
Volatile Acidity Alcohol	0.84	0.51	54.8%

Table 3 Decision Tree Regression Results

Finally, we looked at using random forest regression, which is a technique where we construct multiple decision trees and output the mean value of each of these separate decision trees. We initially used the default values provided by scikit-learn, however after initialy promising results we ran a more thorough look varying the number of estimators, the number of input features, the minimum samples required to split a node, and the maximum tree depth. The results are summarized in table 4. These results are a significant improvement over the baseline of always selecting the average, however we see that due to the fact that the wines are much more likely to be near the average, our models fail to accurately predict the tail ends of the distributions.

Attributes	Root Mean Square Error	Mean Absolute Error	Accuracy
All	0.84	0.52	56.7%
Volatile Acidity Citric Acid	0.84	0.51	57.5%

Sulphates Alcohol			
Volatile Acidity Alcohol	0.84	0.51	54.8%

All
n_estimators = 190
max_depth = 20
min_samples_split = 2
max_features = sqrt
	0.62	0.45	65.0%

Table 4 Random Forest Regression Results

Value	Precision	Recall	F1-Score
3	0.00	0.00	0.00
4	0.40	0.10	0.15
5	0.64	0.58	0.61
6	0.51	0.73	0.60
7	0.62	0.09	0.16
8	0.00	0.00	

Table 4 Random Forest In Depth Results

Section V - At Home Regression

We hoped that this analysis would allow us to predict the quality of wines in the real world, however many of the attributes used in this data set are not readily available outside of the lab environment. In order to create a model that is useable by us at home we attempted to build a model using only ‘Alcohol’ which is listed on all bottles of wine, and ‘pH’ which can be tested using affordable tests. We used the same process as we did for the full set of attributes, and the results can be seen in table 5. Our ability to estimate the quality of a wine using these two parameters is only slightly better than the naïve approach of simply guessing the average.

Model	Root Mean Square Error	Mean Absolute Error	Accuracy
Linear Regression, default	0.74	0.58	55.8%
Decision Tree, default	0.93	0.63	0.48
Random Forest, default	0.77	0.58	53.3%

All
n_estimators = 40
max_depth = 5
min_samples_split = 5
max_features = None
	0.73	0.58	55.0%

Table 5 At Home Analysis

References

1. P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties.
In Decision Support Systems, Elsevier, 47(4):547-553, 2009.