178 Final Project

Kai Malloy
ksmalloy@uci.edu

Kevin Chian
kchian@uci.edu

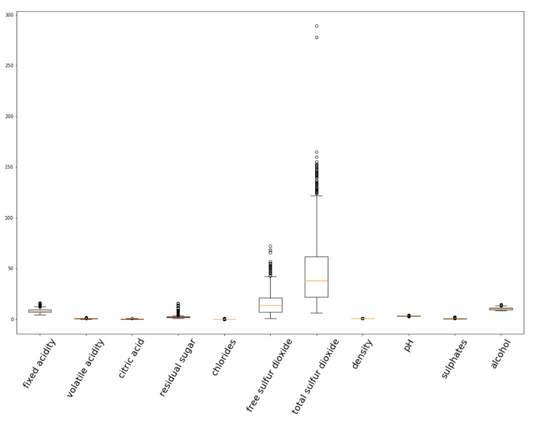
Dylan Feres
dferes@uci.edu

Abstract

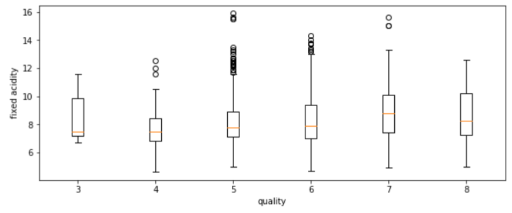
This is our final project for the winter course of 178 with Professor Stephan Mandt. You will find the data exploration of the Wine Quality dataset below. All three of us contributed to data exploration, Kai worked on Random Forest, Dylan worked on SVM, Kevin worked on model comparison and further analysis, and we all filled in our respective sections for this write up.

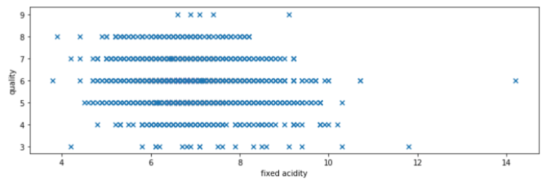
1 Data Exploration

To better grasp the Wine Quality data, we first took a look at the distribution of the features. We did this through printing the min/max/mean/standard deviation for each feature and then transitioning to visualization with a box-and whiskers plot as seen to the right. We noticed that the "total sulfur dioxide" feature has extremely large values which means that we may have to normalize/scale our data for some models. This graph also made it evident that some features have outliers that may have to be addressed to prevent from skewing our learners.

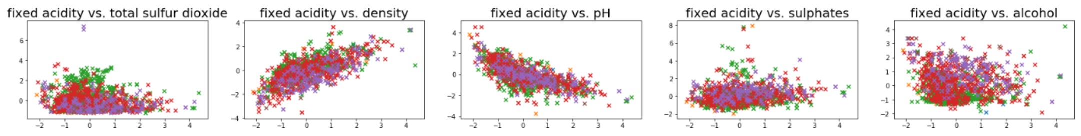


The second step we took was to look at the spread of the target data (quality of the wine). We performed the same analysis of printing out calculations first and then we decided to use a histogram. This graph shows that there is far more data with a 5 and 6 quality rating than any other. This is insightful because it tells us that with fewer samples, we may run into prediction difficulties resulting from distinguishing between a 5 and a 6.

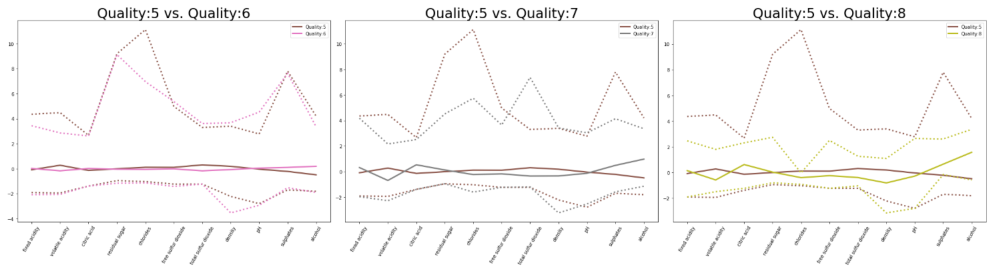




After analyzing the feature and target data separately, we decided to visualize them combined and see the spread of the features between each quality value. As seen in the box-and-whisker plot and scatter plot above, these graphs help to show that the spread is relatively consistent across different quality values.



To find the correlation between features, we graphed a scatter plot for all feature combinations as seen above. This plot was useful in figuring out which features were similar and which to choose from when performing feature-selection for the models. As seen in the graph, fixed acidity roughly correlates inversely with pH as it should, but there are some deviations which make both features necessary. As an extra step, the points on the scatter plot were colored based on quality to see if some pattern was discernible.



Finally, we compared the differences between two given qualities over the features. By doing this, we can see how the features change as quality varies and whether or not there are specific features that define certain qualities. In the example above, qualities 5 and 6 are quite similar, a problem also hinted by the analysis of the target data above.

2 Model Exploration

2.1 Random Forest

2.1.1 Model Setup

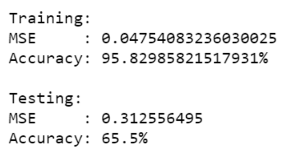
One of the models that our group explored was the Random Forest. We ended up trying out the random forest after considering working with decision trees. Decision trees could work with our data but because they are highly susceptible to overfitting, we decided to go with the collection of decision trees. Random forest will also allow us to have more flexibility in controlling the error due to bias and variance.

To start off, we read in the red/white wine data into our jupyter notebook and split up the feature values from the target values for each. Then we split the feature data into training and test data with a 75/25 split. This way we will have a small subset of data to test the accuracy of our model. Unlike linear and logistic regression, random forest is not affected by normalizing/scaling so we did not have to do any further preprocessing on the data.

For implementing the random forest, we decided to use methods from the Scikit-learn library. This was rather intuitive as we could pass in values as arguments to tweak the hyperparameters. We decided to set up a regression model with the RandomForestRegressor function. To test our performance, we created a function to calculate the mean squared error and a function to calculate the accuracy of the target values compared to the prediction values rounded up to the nearest whole number.

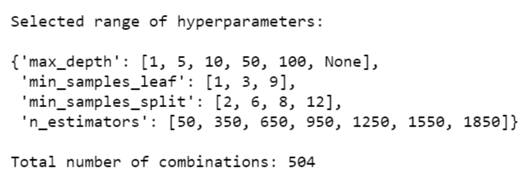
2.1.2 Performance Validation

We first ran the random forest regressor with n_estimators:1,000 (the number of trees) and kept the remaining default values. The default values consist of max_depth: None (expand nodes until all leaves are pure), min_samples_split: 2, min_samples_leaf:1, and bootstrap:True. We were able to predict the test data with 65.5% accuracy while predicting the training data with 95% accuracy. As predicted, training data was performing very well but the testing data seemed low.


```
Training:
MSE      : 0.047540832360300025
Accuracy: 95.82985821517931%

Testing:
MSE      : 0.312556495
Accuracy: 65.5%
```

We believed that this was the result of overfitting and we decided to combat this through using the scikit-learn functions RandomizedSearchCV and GridSearchCV. These functions implement k-fold cross validation while training on a range of different hyperparameter values to find the most optimal ones. Our plan was to use RandomizedSearchCV on a wide range of hyperparameters and then to use GridSearchCV to further narrow the range of our hyperparameters.



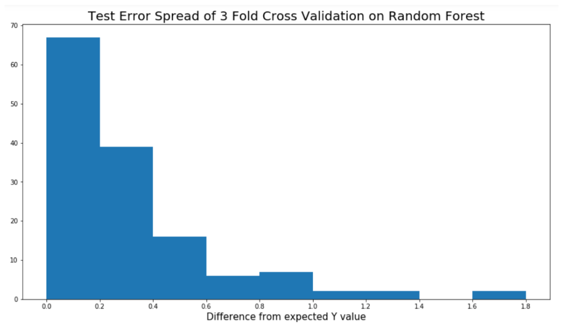
```
Selected range of hyperparameters:

{'max_depth': [1, 5, 10, 50, 100, None],
 'min_samples_leaf': [1, 3, 9],
 'min_samples_split': [2, 6, 8, 12],
 'n_estimators': [50, 350, 650, 950, 1250, 1550, 1850]}

Total number of combinations: 504
```

This plan was essential for tweaking the hyperparameters because we needed to make a performance vs. time tradeoff. Ideally, we would test all 504 combinations of hyperparameters as seen to the left, but we decided to randomly select 100 combinations because training a model was costly with the additional load of 3 iterations per combination from cross validation (3-fold). We were able to test 100/504 (19.84%) of all combinations while running 300 iterations which took roughly 8 minutes.

Even after implementing cross validation and tweaking the range of hyperparameters, we were seeing only a slight increase in performance to 68% accuracy. From analyzing the misclassified data, we noticed that 75.18% of the data was only off by 0.4 causing the classifier to round it up or down from 0.5 as seen in the histogram to the right. From this, we concluded that we may need to work on the way in which we determine accuracy as we are performing regression and rounding up to whole number values for the target.



2.2 Support Vector Machines

2.2.1 Model Setup

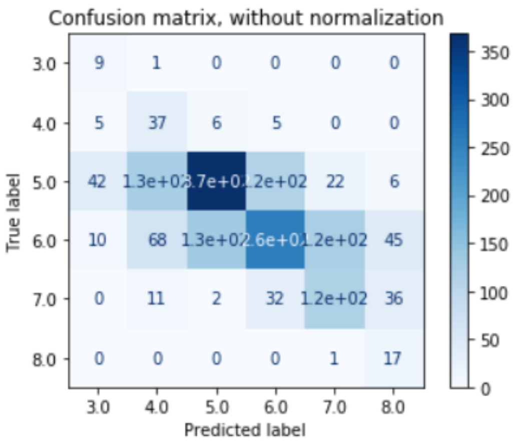
The second model that we determined would be best suited for the wine data set was a support vector machine (SVM) classifier; particularly, we used the Scikit-learn implementation of a support vector classifier (derived from libsvm). An SVM classifier was chosen for its flexibility in choice of kernel types, of which 4 are supported. The non-linear kernels “Radial Basis Function” (RBF) and “Polynomial” were initially chosen primarily because they implicitly map the features into a higher dimensional feature space, yielding a decision boundary that is not limited to producing a straight line; this was key in our consideration of model choice because of the number of classes in the wine dataset and the need for a complex decision boundary. One concern with using either of these kernels (RBF specifically) is that the derivation of both rely on the assumption that the features have a mean of zero and a unit variance; this assumption implies that the data provided should have a normal distribution. In order to accommodate this assumption, all features were scaled down (using sklearn’s “StandardScaler”) and the mean subtracted from each feature. Taking this measure ensured that any outliers in the wine data did not disproportionately affect the objective function.

As a preliminary exploration to test the performance of the kernels, the data was split into a 75/25 training/validation pair (for both the red and white wine datasets) and two separate for-loops were used to train the models. The polynomial model was first explored using various degree polynomials (ranging from a degree of 1 to 10) and the model using the RBF kernel was explored by varying the kernel coefficient. Both models were compared using the mean of the sum of correctly predicted classes within the validation set (accuracy). The model using the RBF kernel performed marginally better, but not enough to justify fixing this kernel as the only one to be further explored; thus, a deeper analysis was needed.

Because of the relatively small number of samples in the data set (creating the potential for overfitting), k-fold validation was used to compare the classifiers with varying hyper parameters. To ensure that each fold contained roughly the same frequency of class occurrence, stratified k-fold validation was used with the value of k varying from 2 to 10. A nested for-loop was used to vary the hyper parameters: C (the penalty parameter), kernel, degree, gamma (the kernel coefficient), probability, and class weight. The mean score was recorded for each iteration and then compared.

2.2.2 Performance Validation

After getting some initial results, we wanted to try and visualize where our predictions were failing. Since we were initially trying to classify each sample into a quality, the "accuracy" result we were calculating was based on the number of correct predictions. As you can see in the confusion matrix below, however, we had a large number of misclassifications between five and seven. This is likely due to similar features. However, it does follow the general trend of an accurate predictor. We needed to reframe the problem, or at least the analysis of our accuracy. The confusion matrix shows that a classifier for this problem was the wrong choice: the difference between 5 and 6 was low enough that we misclassified many examples which inflated our error rate.



Afterwards, we tested a support vector regression model. This model, when tested, gave similar results. The "accuracy" after we rounded the results was about 60%, but the mean squared error was 0.42. In other words, the regression part of the model was not off by much, but trying to round into specific classes was giving us results which seemed overly poor.

3 Model Comparison

The models we decided upon were quite comparable in result. We ended up using a couple of metrics for model selection, most important of which was the mean squared error. At the start, we used an "accuracy" metric, essentially measuring what percent of each quality we got correct after rounding our regression result to fit a class. These initial results were not what we expected; the accuracy result was lower than we wanted for the test set. After extensive parameter tuning and k-fold cross validation to verify our results, we remembered our data exploration stage. The critical piece of information was how common 5 and 6 quality wines were, leading us to visualize our results with a confusion matrix and pay more attention to the MSE than the accuracy.

	MSE	MAE	Accuracy
SVM Classifier	0.440625	0.378125	0.653125
SVM Regression	0.495678	0.429713	0.593750
Random Forest	0.358072	0.424138	0.684898

Both random forest and SVM models performed fairly well with the data we were given. The mean squared error for all three of the models we tested were well below one, which can be interpreted as the model rarely if ever predicting a value which is more than one quality rank off. The random forest model performed with the best accuracy in our tests, but the SVM classifier had the smallest mean absolute error. In other words, the SVM classifier was off less quite consistently, which makes a lot of intuitive sense given the fact that the classifier can only produce whole numbers. Random forest, with the lowest MSE and highest accuracy, makes the most sense to choose as a final model even though the MAE is larger than the classifier's because the MSE punishes large errors more strongly and the accuracy was relatively high as well. Choosing a classifier for a problem like this also loses us some potentially useful information by assuming the quality is a continuous value. Fractional values are simply more precise, one would know whether their wine is closer to 5.5 or 6.4 instead of assuming that it's a flat 6.