Report - Wine Quality

Summary

The two Wine Quality datasets[1] can be found on UCI's Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Wine+Quality). They describe objective properties of White and Red "Vinho Verde" wines from Portugal and includes subjective grading of their quality performed by wine critics ranking the wine from 0 (worst) to 10 (best). I've chosen to combine the two datasets into one and create 6 groups based on the type of wine and the quality grade received. I consider 4 and below "Poor"-, $5-7$ "Normal"- and 8 and above "Excellent"-quality. My goal has been to predict from the chemical properties of the wine which of these 6 groups it belongs to with an eye towards recommending excellent wines to a potential end user:

1. Poor Red
2. Poor White
3. Normal Red
4. Normal White
5. Excellent Red
6. Excellent White

Workflow

I've used the package 'caret' for this project as it provides a uniform interface to many different Machine Learning algorithms. After exploratory analysis, I used the built in pre-processing tools to evaluate several steps to make the data suitable for a Machine Learning task.

To do these estimations, I have evaluated four steps of pre-processing of the data:

1. Removing variables with near zero variance (none)
2. Removing highly correlated variables (none)
3. Centering all variables around zero by subtracting their average
4. Scaling all variables so they are in the interval $[-1, +1]$ by dividing them by their Standard Deviation.

This results in a dataset that is quite suited for many Machine Learning techniques, of which I use 80% for training models and have set aside 20% for testing the finished models. I chose these proportions due to the sparsity of "Excellent" and "Poor" wines. The low number of the extremes makes the algorithms need more training data to predict these. This $80-20$ proportion still keeps some samples of "Poor" and "Excellent" wines left for verification in the test set.

Results

My original thought was to create an ensemble model of the diverse models I trained and get even better results this way, but in the end, the various models had prediction problems that also affected the ensemble model. This caused an ensemble to perform worse than the top performing individual model so the end result is from a Random Forest model using the package `ranger` **which gives an accuracy of 93.89%**.

[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. ISSN: 0167-9236.

Exploring the dataset

The source data when downloaded consists of two files with 12 columns like this:

fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
7.4	0.70	0.00	1.9	0.08	11	34	1	3.51	0.56	9.4	5
7.8	0.88	0.00	2.6	0.10	25	67	1	3.20	0.68	9.8	5
7.8	0.76	0.04	2.3	0.09	15	54	1	3.26	0.65	9.8	5
11.2	0.28	0.56	1.9	0.07	17	60	1	3.16	0.58	9.8	6
7.4	0.70	0.00	1.9	0.08	11	34	1	3.51	0.56	9.4	5
7.4	0.66	0.00	1.8	0.07	13	40	1	3.51	0.56	9.4	5

Columns 1-11 are the various objective properties of the wine that I use as predictor variables. Column 12 is the subjective quality of the wine evaluated by experts with a median of at least 3 evaluations made by wine experts.

Evaluating pre-processing steps

Highly correlated variables

In the description of the dataset, the authors state that they think there might be strong correlation between several variables, and while there is correlation between several variables, it does not rise to the level deemed significant enough to remove any by `caret`'s data processing.

Near Zero Variance

All the 11 variables have enough variance that they provide some information useful to classification of the different wines so none have been removed.

Centering & Scaling

Different Machine Learning algorithms can be more or less sensitive to having data be centered around zero and scaled to $[-1, +1]$. For this project, I have chosen to include these two steps for all algorithms.

Models

I wanted to try a range of models on this dataset to evaluate which would be most effective and potentially prove to be useful for an ensemble. As I am working with a laptop, I started with faster algorithms and tuned them quite a bit before going up in processing time to some small Neural Nets to see if I could further improve my results. I also enabled parallel processing through the package `doParallel` to allow `caret` to take advantage of this wherever possible.

K-Nearest Neighbors

K-Nearest Neighbors is one of the easiest to understand classification algorithms. To get a feel for what I can do with this task, this was my first choice to test. I chose the baseline `knn` package that comes with `caret` and after some tuning I got an accuracy of 92.54% on my test set. When looking at the Confusion Matrix for this algorithm,

	Reference					
	excellent_red	**excellent_white**	**normal_red**	**normal_white**	**poor_red**	**poor_white**
Prediction						
excellent_red	0	0	0	0	0	0
excellent_white	0	0	0	0	0	0
normal_red	4	0	299	2	12	3
normal_white	0	36	5	905	1	34
poor_red	0	0	0	0	0	0
poor_white	0	0	0	0	0	0

it is apparent that despite having a high accuracy, the algorithm has simply predicted everything to be "Normal" quality. Thus it achieves a high accuracy despite not actually achieving the objective.

Partial Least Squares

Another popular method is to use a Partial Least Squares algorithm. I decided to use the `kernelpls` algorithm from the `pls` package to represent these. After tuning, I achieved an accuracy of 92.85% on the test set. The confusion matrix

	Reference					
	excellent_red	**excellent_white**	**normal_red**	**normal_white**	**poor_red**	**poor_white**
Prediction						
excellent_red	0	0	0	0	0	0
excellent_white	0	0	0	0	0	0
normal_red	4	0	302	1	11	2
normal_white	0	36	2	906	2	35
poor_red	0	0	0	0	0	0
poor_white	0	0	0	0	0	0

shows that this algorithm is also predicting everything to be of "Normal" quality thus achieving an artificially high accuracy.

Discriminant Analysis

Moving on to Discriminant Analysis, I chose the algorithm `hdda` from the package `HDclassif` to represent this type of algorithm. After tuning, I achieved an accuracy of 90.24% on the test set which is quite a bit below the other two methods tried so far. Considering the relative weakness of the previous two methods, this is already more interesting. The confusion matrix

	Reference					
	excellent_red	**excellent_white**	**normal_red**	**normal_white**	**poor_red**	**poor_white**
Prediction						
excellent_red	0	0	6	2	0	0
excellent_white	0	0	0	0	0	0
normal_red	4	0	273	7	9	1
normal_white	0	36	5	896	1	34
poor_red	0	0	20	0	3	0
poor_white	0	0	0	2	0	2

shows that now at least the algorithm is predicting other things than all wines being "Normal". However, considering that all the "Excellent" wines have been predicted as "Normal", the end result is not that useful for someone looking to find an excellent wine.

Support Vector Machine

The authors of the dataset mention that they achieved good results for their admittedly quite different task using Support Vector Machine algorithms, so I decided to try one of these. I chose the algorithm `svmLinear3` from the package `LiblineaR` to represent these. After tuning, I got an accuracy of 92.85% on the test set which is up again since the Discriminant Analysis. The confusion matrix reveals that unfortunately, the model is again predicting that all wines are "Normal":

	Reference					
	excellent_red	**excellent_white**	**normal_red**	**normal_white**	**poor_red**	**poor_white**
Prediction						
excellent_red	0	0	0	0	0	0
excellent_white	0	0	0	0	0	0
normal_red	4	0	302	1	11	1
normal_white	0	36	2	906	2	36
poor_red	0	0	0	0	0	0
poor_white	0	0	0	0	0	0

Random Forest

Random Forests have a good history for classification so I wanted to try the algorithm `ranger` from the package `ranger` for this type of algorithm. After tuning, I achieved an accuracy of 93.77% on the test set which is another leap above the previous algorithms that have predicted everything to be "Normal". The confusion matrix shows that we are predicting more accurately the "Excellent" wines, with some "Normal" reds being erroneously predicted to be "Excellent" as well.

	Reference					
	excellent_red	**excellent_white**	**normal_red**	**normal_white**	**poor_red**	**poor_white**
Prediction						
excellent_red	1	0	0	0	0	0
excellent_white	0	12	0	0	0	0
normal_red	3	0	301	2	12	1
normal_white	0	24	3	905	1	35
poor_red	0	0	0	0	0	0
poor_white	0	0	0	0	0	1

Finally, at least, this is a useful result for recommending wines.

Second random forest

Due to the success of the first Random Forest algorithm, I wanted to try a second one, so I chose the algorithm `Rborist` from the package by same name. After tuning, I got an accuracy of 93.62% on the test set which is comparable to the first one. The confusion matrix

	Reference					
	excellent_red	**excellent_white**	**normal_red**	**normal_white**	**poor_red**	**poor_white**
Prediction						
excellent_red	1	0	0	0	0	0
excellent_white	0	12	0	0	0	0
normal_red	3	0	301	2	12	0
normal_white	0	24	3	903	1	36
poor_red	0	0	0	0	0	0
poor_white	0	0	0	2	0	1

shows the same predictions with a couple more "Normal" white wines predicted to be "Poor" wines.

Neural Net

Neural Nets are quite computationally heavy, but they have a history of producing good results for prediction, so lastly I wanted to train one or more of these to see if I could further improve results. I chose the algorithm `avNNet` from the package `nnet` for this. After tuning, I achieved an accuracy of 93.24% on the test set which is quite good. The confusion matrix

	Reference					
	excellent_red	**excellent_white**	**normal_red**	**normal_white**	**poor_red**	**poor_white**
Prediction						
excellent_red	0	0	0	0	0	0
excellent_white	0	1	0	0	0	0
normal_red	4	0	302	1	12	0
normal_white	0	35	2	904	1	31
poor_red	0	0	0	0	0	0
poor_white	0	0	0	2	0	6

shows that unfortunately the gains over predicting all wines are "Normal" come from improved accuracy on the "Poor" wines but not the "Excellent" ones.

Second try Neural Net

As the first model is a single layer Neural Net, I also wanted to at least try a multilevel approach. I chose the algorithm `mlpML` from the package `RSNNS` to try a Neural Net with 3 levels. After tuning, I achieved an accuracy of 93.01% on the test set, which is very similar to the first Neural Net. The confusion matrix

	Reference					
	excellent_red	**excellent_white**	**normal_red**	**normal_white**	**poor_red**	**poor_white**
Prediction						
excellent_red	0	0	0	0	0	0
excellent_white	0	0	0	0	0	0
normal_red	4	0	301	2	11	1
normal_white	0	36	3	902	1	29
poor_red	0	0	0	0	0	0
poor_white	0	0	0	3	1	7

shows that this algorithm has the same weakness and that any gains made are on "Poor" wines.

Conclusion

This dataset is complicated to work with due to how few of the wines fall in the groups I called "Poor" and "Excellent". This limits the models quite a bit and only one algorithm really produced somewhat useful results for recommending wines to a user; it was Random Forest. The best results were achieved using the algorithm `ranger` and can reliably predict excellent wines for an end user based on chemical properties.

Future steps

It might be interesting to use subsambling methods as described here http://topepo.github.io/caret/subsampling-for-class-imbalances.html to try to increase accuracy on different models, but that is a more complex topic that would require that I retrain all the models for evaluation.