An analysis and comparison of various machine learning algorithms used to predict wine quality

Alvaro Robledo Vega
Imperial College London
Electrical and Electronic Engineering
ar5115@ic.ac.uk

1. Introduction

This paper aims to study and implement multiple linear and non-linear machine learning algorithms and evaluate their performance.

The chosen dataset consists of red and white wine samples from the north of Portugal. Each instance is composed of 12 variables: 11 physicochemical properties and a quality score which ranges from 1 (lowest) to 10 (highest) [1].

Our aim is to investigate predictors which, taking the 11 physicochemical properties as inputs, can predict the wine quality with the smallest error possible.

Throughout this paper, unless explicitly stated otherwise, we will consider the combined red and white wines dataset.

1.1. Dataset observations

The given dataset consists of 6497 wine instances, with most quality scores centred around 5 and 6 (as shown in Figure 1)

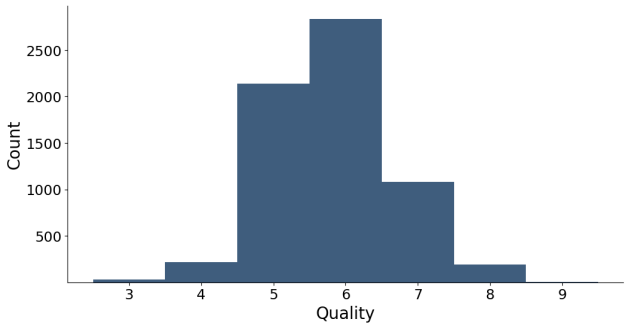

Figure 1: Histogram of wine quality scores

Such an unbalanced distribution is likely to pose a challenge for our learning models. Most algorithms will centre their predictions around 5 and 6, making the rest of the scores harder to predict.

Plotting the correlation between all the given variables shows that the most positively correlated with quality is alcohol. Thus, alcohol is likely to have the biggest effect when predicting quality.

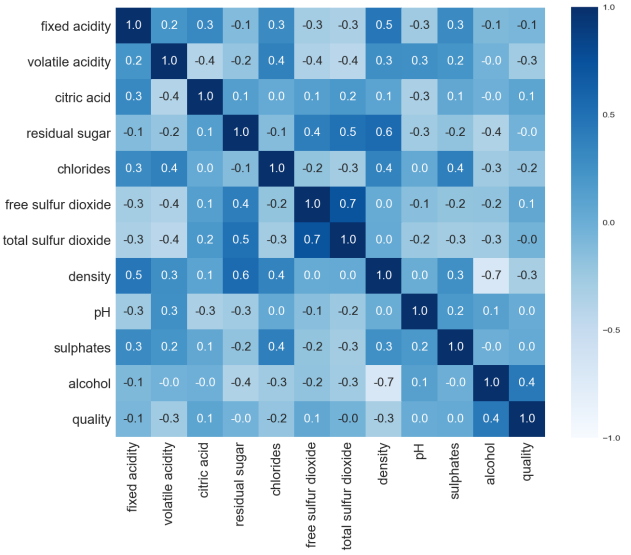

Figure 2: Correlation between variables in the dataset

Additionally, this also suggest that some variables could potentially be dropped in search for a higher performance.

1.2. Data preprocessing

We first split our data into training (80%) and test sets (20%) [2].

Since our dataset is quite sparse and has attributes of varying scales, it is important to standardize our data (instead of applying other popular data preprocessing techniques such as Min-Max scaling or a transformation to unit norm)[3].

To achieve this, we standardise the training data to have zero mean and one standard deviation. We then use these training parameters to scale our test data, which is the correct way of performing these kinds of tasks [4].

1.3. Loss function and metrics

In our models, we would like to capture the idea that, for a wine of quality 5, an incorrect prediction of 2 is inherently somewhat worse than an (also incorrect) prediction of 4. For this purpose, we initially choose a loss function equal to the squared error

$$\ell(\hat{y}, y) = (\hat{y} - y)^2$$

This is a widely used loss function in regression problems. The idea behind squaring the error is so that negative values do not cancel positive ones.

Throughout this paper, we will make use of 2 different metrics in order to properly interpret and compare our results: accuracy (%) and mean squared error (MSE):

$$\text{accuracy (\%)} = \frac{\text{correct predictions}}{\text{total number of predictions}} \times 100$$

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y} - y)^2$$

Accuracy is a widely used metric in this field for performance benchmarking, hence why we will be reporting it. However, since this dataset does not have an equal number of instances of each quality score, it loses relevance and fails to capture the whole picture. To overcome this, we will also be using the mean squared error: this will provide us with extra information about the quality of the predictions (distance to the correct label), which the accuracy metric fails to provide.

When looking at accuracy, higher is better. When looking at MSE, it will be the opposite.

2. Initial approaches

In this section we try to solve the problem using regression. There are two reasons for this: first, as mentioned earlier, the choice of our loss function, to incorporate a distinction between the different levels of getting the label wrong. Second, the fact that our quality attribute is discrete numeric (ideal for regressions).

2.1. Baseline regressor

Before even attempting a simple linear regression, it is important to consider the very most baseline predictor: one that always predicts the output to be the same number. In this case, we choose this to be the mean of the quality attribute (of the training dataset).

Throughout the paper, we will make use of a popular Python library called *scikit-learn* to implement our algorithms. In particular case, we use the very simple *DummyClassifier()* function to create this baseline regressor.

The performance it achieves is a mean squared error of 0.763477, and a classification accuracy (after rounding our predictions to the nearest integer) of 43.62%. Interestingly, despite there being 10 classes (the quality scores from 1 to 10), this algorithm achieves an initially shocking high accuracy. This is, again, due to the fact that most wine instances in our dataset (more than half of them, in fact) have a quality of 5 or 6, as shown earlier in Figure 1.

We can assume that the rest of the regression algorithms we try will, therefore, perform better than this.

2.2. Linear regression

Our first serious approach is to implement a classic linear regression. This aims to minimise a squared error loss function, as explained in the lectures [5]

Note that, although in the rest of the paper we will be using cross-validation to pick the best hypothesis, there is no point in using it here because our hypothesis set is unique. The results are reported in the next section, to allow for comparison with regularisation techniques.

2.3. Adding regularisation

It is usually a good idea to introduce some regularisation (penalty) in linear regressions, particularly to avoid overfitting. We will consider 3 different methods, also introduced during the lectures: Lasso (L1), Ridge (L2) and Elastic Net (L1 and L2)[5].

In theory, we expect the Ridge regression to provide better results than Lasso, since this dataset only has 11 features, and therefore the main advantage of Lasso, its feature selection property, will not really come into play.

We use 10-fold cross-validation to select the best parameter λ for each of the three regressions.

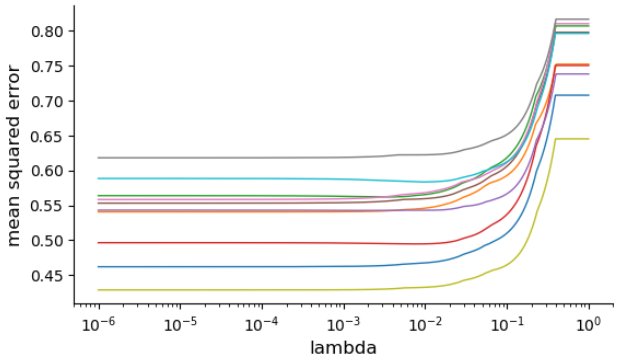

Figure 3: MSE for different values of λ in a Lasso regression using 10-fold cross-validation

After performing cross-validation, the model with the optimal λ parameter is retrained on the whole training data.

To implement this, we relied on the scikit-learn functions *LassoCV()*, *RidgeCV()* and *ElasticNetCV()*, which perform their respective names' regression after a k-fold cross-validation [6]

	No reg.	With regularisation		
	LR	Lasso	Elastic Net	Ridge
Acc.	50.23%	50.08%	50.23%	50.23%
MSE	0.56844	0.56839	0.56830	0.56833

Table 1: Results obtained for Linear, Ridge, Elastic Net and Lasso Regressions

As expected, even the most basic linear regression already improves the baseline's accuracy and MSE (which, recall from the previous section, was 43.62% and 0.763477, respectively). Interestingly, adding regularisation only slightly outperformed the basic linear regression. This could be because it never suffered too much from overfitting in the first place.

We can confirm however, that Ridge does indeed outperform Lasso, as expected, and that Elastic net is slightly the best of all. This is due to the fact that it combines the advantages of both L1 and L2 penalty functions[5].

3. More advanced algorithms

Luckily, linear regression is not the only tool within our reach. In this section, we will propose and discuss several other non-linear approaches: Feature Transformation, Support Vector Machines (SVM) and Neural Networks (NN).

3.1. Feature Transformation

Before completely moving away from linear regression algorithms, it could be interesting to try transforming the features with different polynomials. Recall that a transformation of a set of 2 features (X_1, X_2) to second-order will yield a new set of features $(1, X_1, X_2, X_1X_2, X_1^2, X_2^2)$. In our case, since we have 11 features, a transformation of degree 2 will yield 78 features, and one of degree 3, 170.

For this purpose, we make use of the scikit-learn function *PolynomialFeatures()*, which preprocesses our data to create the new set of features. We run the same regressions as in the previous section (same 10-fold cross-validation, etc.) but this time with the new set of features

	No reg.	With regularisation		
	LR	Lasso	Elastic Net	Ridge
Acc.	52.23%	50.85%	52.31%	52.15%
MSE	0.51827	0.536802	0.54941	0.54734

Table 2: Results after degree 2 polynomial feature transformation (using 10-fold cross-validation)

For degree 2, the best parameters chosen via cross-validation were: $\lambda = 0.000001$ (Lasso), $\lambda = 0.0219$ and $\alpha = 0.1$ (Elastic Net), $\lambda = 220$ (Ridge)

For degree 3, the best parameters chosen via cross-validation were: $\lambda = 0.00067$ (Lasso), $\lambda = 0.001323$ and $\alpha = 0.3$ (Elastic Net), $\lambda = 12$ (Ridge)

	No reg.	With regularisation		
	LR	Lasso	Elastic Net	Ridge
Acc.	52.92%	52.08%	52.15%	52.62%
MSE	0.54090	0.51904	0.51666	0.51318

Table 3: Results after degree 3 polynomial features transformation (using 10-fold cross-validation)

The first important observation is that all models outperform their counterparts in the previous section. This was somehow expected, since 11 features was quite a low number to begin with, and 78 (and 170) features have the potential of giving us more information.

There is however something slightly controversial: when using a degree of 2, the non-regularised model surprisingly outperforms the regularised ones, but this is not the case when using a degree of 3. A possible explanation could be that the simple regression using degree 2 features does not suffer from overfitting yet (and therefore regularisation is not advised), while when using degree 3 features, the simple model actually begins to overfit, and regularisation starts doing a better job.

Another interesting observation is that, in the degree 2 case, the Lasso parameter λ tends to 0. This is due to the fact that a lasso regression with $\lambda = 0$ is just a basic linear regression.

Ridge was consistently the best of the regularised regressions in both degree 2 and 3. This is probably because, as we saw in the first section of the report (Figure 2), there is some multicollinearity between several of the features. Applying a polynomial transformation does not change this (in fact, if anything, it would worsen it), and one of the main advantages of the L2 penalty is, precisely, that it helps address multicollinearity [7].

Finally, it is worth mentioning how the Ridge regression, because it has a closed form solution, was the fastest to compute of the three.

3.2. Support Vector Machines

A Support Vector Machine (SVM) aims to maximize the margin between data points and a separating hyperplane. This hyperplane is defined by the so-called Support Vectors [8].

SVMs use kernels to transform the original data into a new space. Some of the most widely used kernels are: linear, polynomials, RBF (Radial Based Function) and sigmoid. The choice of kernel depends on the dataset. Generally, RBF tends to outperform the others [9], and, in fact,

a quick testing of each kernel against our dataset can confirm that this is also the case here.

The RBF kernel is a nonlinear and flexible one, which might be prone to overfitting. To avoid this problem, we need to carefully finetune its hyperparameters, which are γ, ϵ and C. We do this using 10-fold cross-validation, with the help of the *GridSearchCV()* function from sckikit-learn.

γ is the free parameter of the Gaussian RBF function [5]. A lower γ helps to fight overfitting (by increasing the bias and decreasing the variance). ϵ, despite controlling something different (the size of the tube within which we tolerate prediction errors), ends up having a similar influence. Finally, C is the cost parameter, which controls the trade-off between the model complexity and the amount up to which deviations greater than ϵ can be tolerated.

We use the *SVR()* function from the scikit-learn library to implement our SVM Regression. This uses an ϵ-insensitive loss function, which penalises errors greater than ϵ [10]. Note that this is a different loss function than the one used in linear regression (which was the squared error loss).

We perform 10-fold cross-validation for the values: $C = [0.1, 1, 10, 100]$, $\epsilon = [0.01, 0.1, 0.5, 1]$ and $\gamma = [0.01, 0.1, 0.3, 0.5, 0.7, 1]$. The best tuple giving the highest cross validation score is $C = 1$, $\epsilon = 0.1$, $\gamma = 0.5$

	SVM (RBF Kernel)
Accuracy	61.38%
MSE	0.449931

Table 4: Results for SVM after 10-fold cross validation

These results clearly outperform all of the other methods we have tried so far. It seems like the RBF kernel does quite a good job in separating our data, especially considering how average this wine dataset is (more on this later)

3.3. Neural Networks

Neural Networks are booming in the field of Machine Learning and Pattern Recognition. In this section we will be fitting a Multi-Layer Perceptron Neural Network, which performs training using the backpropagation algorithm [11].

Once again, to avoid overfitting, we will perform (10-fold) cross-validation over three hyperparameters: the number of hidden (fully-connected) layers, alpha (the L2 penalty used for regularisation) and the activation function at the end of each neuron.

We use the *MLPRegressor()* function of sklearn to implement the multi-layer perceptron, and *GridSearchCV()* to perform 10-fold cross-validation for the values: hidden_layer_sizes = [3, 5, 10, 100], alpha = [0.0001, 0.01, 1, 10, 100] and activation = [relu, 'logistic', 'tanh', 'identity']. The solver used for weight optimization is *lbfgs*, an optimizer in the family of quasi-Newton methods which tends to have one of the best performances [12]. The best tuple giving the highest cross validation score is: 'activation': 'relu', alpha $= 10$ and hidden_layer_sizes $= 100$

	Neural Network (MLP)
Accuracy	54.62%
MSE	0.471624

Table 5: Results for Multi-Layer Perceptron Neural Network after 10-fold cross-validation

This set-up demonstrates a very competitive performance, although slightly under that of the SVM.

4. Discussion of results

Overall, the SVM method performed the best, and thus becomes our preferred method for implementation. However, the MLP Neural Network also left a good impression, and it could be interesting to see future work on it, for example regarding using other types of Neural Networks different than the MLP.

The results, in general, are not very encouraging, especially when we look at accuracy. We only managed to bring it up from 43.62% (baseline case) to 61.38% (using an RBF kernel). We suspect that this might be a consequence of the dataset being particularly sensitive to subjectiveness: the fact that the quality score is formulated by humans, even if these are experts in their field, seems to suggest that there might not be a clear correlation between what society considers to be a good wine and its physicochemical properties. Furthermore, this raises some doubts about the wine tasting industry and its actual measurable contribution to society.

5. Proposals for Future Work

It would be interesting to see whether applying PCA (Principal Component Analysis) to reduce the dimensionality of our data (particularly big after feature transformations) helps in reducing the noise. It might also be a good idea to implement other advanced algorithms such as Decision Trees or Random Forests, as well as different types of Neural Networks other than the MLP.

Finally, on a more personal level, I would be interested in seeing if an unsupervised method (such as k-clustering) is able to produce similar results without learning from the correct quality labels. If this was not the case, it would be another piece of evidence that the act of wine tasting is really more subjective than objective.

6. Pledge

I, Alvaro Robledo Vega, pledge that this assignment is completely my own work, and that I did not take, borrow or steal work from any other person, and that I did not allow any other person to use, have, borrow or steal portions of my work. I understand that if I violate this honesty pledge, I am subject to disciplinary action pursuant to the appropriate sections of Imperial College London.

References

[1] URL: https://archive.ics.uci.edu/ml/datasets/Wine+Quality.

[2] URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3090739/.

[3] URL: https://machinelearningmastery.com/prepare-data-machine-learning-python-scikit-learn/.

[4] URL: http://sebastianraschka.com/Articles/2014_about_feature_scaling.html.

[5] Andras Gyorgy. "EE3-23 Machine Learning 2018 Lecture Notes". In: (2018).

[6] URL: http://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_model.

[7] URL: https://web.stanford.edu/~hastie/THESES/meeyoung_park.pdf.

[8] URL: http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf.

[9] URL: https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XL-2-W3/281/2014/isprsarchives-XL-2-W3-281-2014.pdf.

[10] URL: http://kernelsvm.tripod.com/.

[11] URL: https://www.idosi.org/wasj/wasj5(5)/5.pdf.

[12] URL: http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html.