CSCI 5525: Analysis of different ML models using Red Wine Dataset

DAVID HWANG

May 11, 2022

1. Introduction

The history of wine dates back well over 3,000 years with many cultures playing a role in the development and popularization of the drink. The Greeks and the Romans were well known for using wine as part of their religious ceremonies serving as a way to communicate with the gods making it a highly valuable product [1] but mostly consumed by the rich. Nowadays, wine has become more widespread being available all over the world with the US being the largest wine consuming nation at over 329 million cases of wine being sold in 2013 [2]. In fact, in America the most preferred type of wine is red wine at 51% of the dollar volume sold over the second most preferred type being white wine at 46% [3].

With wine being consumed by the masses and with the necessity of distinction between brands and taste the quality of the wine became more of a focus. Wine certifications to ensure the quality of the production of wine has been developed over time and has served as a benchmark for consumers [4]. A multitude of factors in determining wine quality has been contested over the years: the complexity of the flavors, the balance of the taste, the wine tasting as it should, and the intensity of the wine and finish [5]. It has been debated that these factors are directly a result of the wine physiochemical properties and so a direct relation to physiochemical properties and the wine's quality have been put into question [6].

With this relationship in mind, winemakers would then be able to optimize their winemaking process by altering the properties of the wine in order to directly increase the wine's quality. Thus, I decided to use a multitude of machine learning algorithms such as neural network, gaussian naive Bayes, logistic regression, and support vector machines in order to see what algorithm is the best predictor in linking specifically red wine's properties to its end quality.

In addition to this, I wanted to use feature engineering to improve the performance of the models I use. Through this process I would be able to identify the most relevant components of the red wine, understand more clearly the relationship between the wine's chemicals, and also its link to the overall quality.

2. Related Work

In order to observe the link between the wine's chemicals and its quality multiple datasets have been created in order to predict the relationship. The dataset that I use for red wine is the Wine Quality dataset that was created by Cortez et al. in 2009 [7]. They used this dataset in order to perform multiple regression techniques under a computationally efficient procedure for both red wine and white wine [17]. The techniques that they compared was neural networks, support vector machines, and multiple regression. The conclusion was that support vector machines was best suited for classifying the data.

Since the paper's release in 2009 new advancements have been made in neural networks such as the Adam optimizer which was created in 2014 [8]. Also it only used limited set of models to compare and contrast the performance of the wine prediction which also ran on both the white wine and red wine datasets.

Noah et al. also examined the same dataset but with using naive gaussian bayes, SVM, PRISM, and decision trees [9]. They evaluated the prediction on red wine and white wine and found that PRISM was a standout between the models. However, it seemed that there were complications within PRISM's procedure and so its result was inaccurate. Discluding

PRISM, they found that decision trees was an overall better predictor over guassian naive bayes, and support vector machines.

This paper did not fully optimize their machine learning algorithms and so their final results could be debated. They also did not make use of feature engineering and did not correctly use PRISM in order to get an accurate result. Thus, this paper could be highly inaccurate in order to get a sense of what is the optimal classifier.

3. Data Information

The dataset that I use is the red wine dataset in the UCI learning repository [7]. I specifically used only the red wine dataset as red wine and white wine has different proportions of properties and combining the models would make it difficult to preprocess the data as I would need to take into account the relationship between white wine and red wine. As I mentioned above that red wine is more popular than white wine I decided to base my predictions on the red wine dataset.

The dataset contains 1599 observations and 12 features: fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol, and quality.

4. Feature Engineering and Data Analysis

One way that a model can have poor performance is through overfitting the data. This is due to the model being connected too closely tied with the training data set so when we try to generalize it to a test data set it will perform poorly.

To minimize these effect we can use feature engineering. With feature engineering we can preprocess and transform the data in order to choose the most relevant features to model. This helps eliminate training data which can overfit the model and give us a better predictor.

First we check if there are any missing values in the data. If there is any values that are null that could be a problem in analyzing the data. Fortunately, this dataset is clean and there is not any missing values.

One thing to check is to see if there is any class imbalance with the quality of the wine. We can see that the quality is mostly concentrated at 5 and 6 while fewer observations were made for 3 and 8. Thus, the class is imbalanced with more common quality wine observations than poor or great wines [figure 1 in appendix]. We can take this into account when creating our models.

When choosing the best features to use we can use correlation analysis to see what features are irrelevant. If there is any features that are highly correlated with each other than multicollinearity could be a problem. This occurs when two or more independent variables are highly correlated which would cause a problem when we want to determine the effects of our independent features to our label. This would make it harder to interpret the model [10]. In addition to this, one of our models, gaussian naive bayes, also requires that each feature is independent from other features. In figure 2, we can see that free sulfur dioxide and total sulfur dioxide are highly correlated at 0.67, and citric acid, density, fixed acidity are also highly correlated between each other. So we drop density, citric acid, and free

sulfur dioxide

We also need to remove any outliers in the data to make sure that our models aren't affected by data that isn't expected. Outliers are data points that are not within the expected range of a feature and could be caused due to incorrect measurement or by outside forces so we should seek to remove any of these points in the data set. One way to do this is by identifying the 25th and 75th quantiles for the data and then the difference between the quantiles is the interquantile value. We can then get a minimum and maximum range of our feature by subtracting to the 25th quantile by the interquantile and adding to the 75th quantile by the interquartile [11]. If there are any values that are outside of this range we identify them as outliers and remove them from the data set [figure 3,4].

Looking at the feature distribution [figure 5], we can see that some of our features do not have a normal data distribution. This results in our features being skewed positively or negatively and could increase the overall bias in the model [12]. We can use a Box Cox transformation to transform these features into normal data distributions. A Box Cox transformation is based around the computation of the value of λ, which λ is selected based on if it transforms the data to a normal distribution. The transformation is as follows [13].

$$y(\lambda) = \begin{cases} (y^\lambda - 1)/\lambda, & \text{if } \lambda \neq 0 \\ log(y), & \text{if } \lambda = 0 \end{cases} \tag{1}$$

First we take a look at which features have the most skew to identify what features need to be transformed. We can see from figure 6 that fixed acidity, total sulfur dioxide and alcohol have the most skew. After transforming these features with a Box Cox transformation we can see that the features now have low skew [figure 7] so the features now have a normal data distribution.

5. Experimental Setup

Discussion of hyper parameters chosen are discussed in methods and algorithms. Code for the implementation of the models is provided with the final report.

In order to reduce the effects of overfitting in each of the models I utilized 10-fold cross validation. 10 fold cross validation works by splitting the data into 10 partitions where 1 partition is the test set and the remaining is the training set this is one fold. then we select a different partition for the test set so that it does not repeat with another fold ultimately ending up with 10 folds. We then evaluate the model on each fold and then average the overall error. This allows us to to use all of the data and not be constrained to just one train and test split. For the neural network I split the data with 10 fold cross validation and then took the errors from the models across the last epoch and averaged it to get the result.

In evaluating the effectiveness of each model since the label we will predict is the the wine quality which is a value from 3 to 8. We can just use the metric of accuracy to predict how well the model generalizes to test data. This is calculated as shown below.

$$Error = 1 - \sum_n (Prediction_n == Actual_n)/size(Actual) \tag{2}$$

First the accuracy of the model is calculated by getting the sum of all predictions that are equal to actual data points which we then divide by the total of the data points in actual. Then we subtract 1 from the accuracy to get the error. This gives us a metric to compare which models evaluate better to the data set.

6. Methods And Algorithms

6.1 Gaussian Naive Bayes

Gaussian Naive Bayes is based on Bayes' theorem which evaluates the probability of an event based on given events. This classifier uses a gaussian probability density function when making predictions which calculates the probability of each class, in our case the wine quality, given the input. By choosing the the class with the highest probability this allows us to predict the wine quality from the chemical properties of the wine.

Some of the advantages of Gaussian Naive Bayes is that it is quick to train as we only need to calculate the probability of the label given the input. In addition to this it is also easy to implement. However, the drawbacks are is that this classifier is based on the assumption that each feature is mutually independent from one another. Also since it uses Guassian probability it assumes that each feature is normally distributed.

In order to implement Naive Bayes with marginal Gaussian distributions we can compute the probability density of a feature given the class (wine quality) with the lower equation.

k = class in C, x = feature of the data

$$p(x = v|\ C_k) = (1/(\sqrt{2\pi\sigma_k^2})) * e^{-(v-\mu_k)^2/2*\sigma_k^2} \tag{3}$$

The class priors are computed by $\pi_k = 1/(\#\ of\ classes)$

We compute the probability for a class c by multiplying all the computed features given c, and the class prior. We do this for every class and then we predict the class which has the max probability. this procedure is done for every single observation in X to get our predictions.

Testing Gaussian Naive Bayes with 10-fold cross validation on the red wine dataset we get the results shown in Figure 8. I also included the model performance on the original dataset to see if feature engineering improved the model.

We can see that feature engineering did improve the model. Regardless the model performs very poorly on the dataset and so Gaussian Naive Bayes as a result of the high error is not suited to evaluate the data.

6.2 Logistic Regression

Logistic regression is a classifier that models the probabilities for the possible outcomes. It is an extension of linear regression but instead of fitting a hyperplane the logistic regression model uses a linear model for log odds.

Some of the advantages of using linear regression is that it makes no assumptions regarding the distribution of classes. It is also very fast in classifying labels. Some of the disadvantages is that it requires independent features to not be too highly correlated with each other and also requires linearity of independent variables and log odds.

In order to use logistic regression to predict multiple classes I used multinomial logistic regression. To implement this model I first initialized the parameters which is a weight

matrix w (size of this is number of classes by features of X), a learning rate lr, and then a one hot encoding of y. The one hot encoding allows us to express our y where 1 is the class otherwise it is 0. To give an example, for the Digits dataset a one hot encoding of an observation which has class of 2 would be represented as

[0,1,0,0,0,0,0,0,0,0]

We then go through a given number of training iterations to fit our model.

In these training iterations we multiply our data X to the weight matrix to get our score matrix. We then utilize a softmax function on our score in order to get a matrix of probabilities P. A softmax function takes in a vector(which will be a row in X) and then after applying the softmax function changes each value in the vector to be in the interval (0,1) and summing the vector will equal to 1.

$score = X * w^T$

$softmax(y) = e^y / \sum_y e^y$

$P = softmax(score)$

This P predicts a high probability for the target label and low probabilities for the others. Thus we can use our one hot encoding representation of y to subtract it to P to then get our error matrix. We use this error matrix to create the gradient in which we multiply our X matrix by the error matrix and then update our weight by the learning rate * gradient.

$error = P - y$

$gradient = error^T * X$

$w = w - lr * gradient$

This would be one training iteration and we would repeat this for how many times we specify. After the data has been fit, to predict the observations we now have our w matrix which now has gone through multiple training iterations. We then multiply $X * w^T$ to get our P matrix. Then we predict the classes based on the highest probability of our target label in the P matrix for each observation.

In order to optimize multinomial logistic regression one has to determine the iterations and learning rate of the model. If the learning rate is too large then the optimal value might be missed, if it's too small we may need a lot of iterations to converge to the best value. Using a learning rate of 0.0001 and 10000 iterations I find that it converges to the optimal value.

I also need to optimize the model that utilized the original red wine data set. Using a learning rate of 0.0001 and 20000 iterations I found that it converged to the minimal error.

I tested multinomial logistic regression with 10-fold cross validation on the feature engineered red wine dataset and the original data set the results are shown in Figure 9.

We can see that feature engineering improved the model classification. The model also performed better than Naive Gaussian Bayes.

6.3 Neural Network

Neural networks is comprised of the input layer, then the hidden layers, and finally the output layer connected by feedforward links. Each layers contain nodes which connects to another and has a weight and activation function which can turn off or on the node. The overall network is a combination of function composition of the input and the weight matrices that have an activation function between them. To give an example of a 2 layer neural network. The first layer has a W weight matrix that is multiplied by the input. Then it goes through some activation function which can turn off or on the node. Then that is then multiplied by W_2 weight matrix to get to the output layer.

Neural networks uses backpropagation to train the feedforward neural network. This is in order to adjust the weights to minimize error. Backpropogation starts from the output layer and computes the gradient with respect to the loss function and then adjusts the weights based on the gradient. Then it continues all the way to the input layer until all the weights have been updated.

The advantages to using neural networks is that they can create flexible models to solve the linearly inseparable problem [14]. If the data cannot be separated by just a line then neural network can create a separation hyperplane to separate the data. Some of the disadvantages it can be hard to interpret what the model is doing. Also neural network not only requires lots of data but it is computationally expensive [14].

During optimization, in which the goal is to minimize loss, the process can be hindered by local minima or saddle points. One way to resolve this is by using Adam which combines momentum and bias correction to overcome these obstacles. In order to optimize even further we can have the learning rate decay over time. A large initial learning rate can help reduce the effect of noise and then decaying that learning rate will help the model learn more complex patterns [15]. To implement this we can continuously multiply the learning rate by 0.1 after a set amount of epochs have occurred.

Activation function serves to introduce non-linearity in the neural network. Looking at what it does it determines if a node should be turned off or on and also by how much. There are many different types of activation functions like sigmoid which squashes numbers to range [0,1], ReLU which takes the max(0,x). ReLU tends to converge much faster than sigmoid but any values less than zero are mapped to zero which result in dead nodes that never activate. Since the slope of ReLU is 0 the node's gradient is 0 and thus no learning can occur. Leaky ReLU solves this problem as it maps values less than zeros to a very small positive number preventing nodes from turning off completely.

The loss function is used when adjusting weights with backpropagation. A common loss function to use is called Cross-entropy. Cross-entropy considers a target probability distribution as P and an estimate of the target distribution Q then measures the distance between the two distributions [17]. Cross entropy can be more formally represented as the equation below [17].

$H(P,Q) = -\sum_x P(x) * log(Q(x))$

In implementing a neural network for the red wine dataset there are endless architectures that can be built. In my implementation I have decided on using four hidden layers. Just using one hidden layer would just be a linear regression model so I need more than one

layer. Two layers would be able to create a non linear model but would be lacking depth. Greater depth in neural networks have been reported to generally perform better than shallow networks [16]. I use Leaky ReLU and ADAM optimizer for the reasons stated above. Since we are predicting the wine quality which is just a number from 3 to 8, cross-entropy would be an appropriate loss function to see how far a model is from its prediction to actual value. For the learning rate I start at 0.02 and use 50 epochs for both data sets, these are the values that I experimented with which seems to converge to the minimal error. In order to improve the performance I also use a scheduler to decay the learning rate by 0.15 every 10 epochs.

I tested the neural network with 10-fold cross validation on the feature engineered red wine dataset and the original data set the results are shown in Figure 10.

Here we can see that the difference between the engineered red wine dataset and the original data set are small although there is a slight improvement in test data set. We can also see that it performs the best over the classifiers.

7. Results

The results of the tests on the models using 10 fold cross validation are shown in figure 8, 9, 10. As we can see the feature engineering that we applied on the red wine dataset improved the performance of the models. Overall, neural network performed the best with logistic regression performing slightly worse and gaussian naive bayes performing the worst.

With the results that I have received it is clear that there are strengths to applying new techniques to classifiers. Neural network benefited from the new advancements with Adam and learning rate decay having improved performance. In addition to this, by applying feature engineering and comparing it to the original data set we can see that it is generally effective in increasing performance over the models.

Some problems with the experiment is that I only used neural network, logistic regression, and naive bayes to test out the data set. With such a limited amount of classifiers that are tested on, and a large number of classifiers that exist, there could be a classifier that is better to use on the red wine dataset that I have not tested.

8. Conclusion and Future Work

From this project I was able to apply feature engineering to improve the performance of models across the red wine data set. I also found that neural network performed the best with logistic regression trailing behind and gaussian naive bayes performing the worst.

Since I only tested out a limited amount of classifiers my next steps would be to use other classifiers such as the support vector machine that was used in Cortez et al. paper in 2009 [18]. They concluded that support vector machine (SVM) was the best classifier over neural network and since this project did not test SVM I am curious to know which one is best. Considering the advancements with neural network, and also if support vector machines would improve with feature engineering it would be interesting to see if neural network would edge out SVM.

Appendix

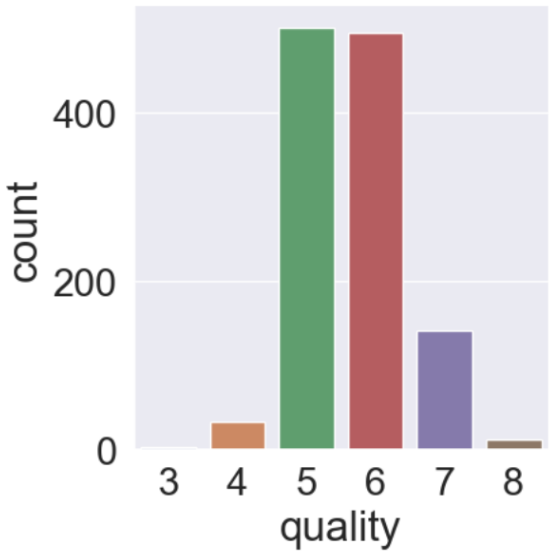

Figure 1: Count of wine quality

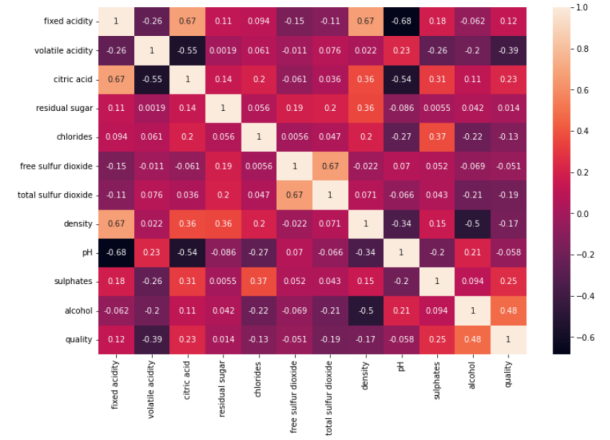

Figure 2: heatmap of feature correlation

Figure 3: Before removing outliers

Figure 4: After removing outliers

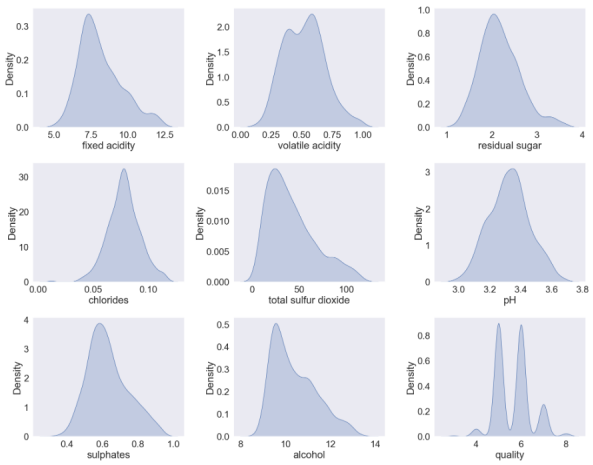

Figure 5: Feature distribution

```
Skewness
 fixed acidity           0.748003
volatile acidity         0.282093
residual sugar           0.625386
chlorides               -0.064911
total sulfur dioxide     0.892078
pH                       0.079758
sulphates                0.513739
alcohol                  0.744075
quality                  0.321122
dtype: float64
```

Figure 6: Feature skew

	Skew	New Skew
total sulfur dioxide	0.890950	-0.022689
fixed acidity	0.747056	0.018871
alcohol	0.743133	0.117989

Figure 7: Feature skew after applying Box Cox transformation

Gaussian Naive Bayes	Feature Engineered Red Wine Dataset	Original Red Wine Dataset
Train Error	0.649	0.707
Test Error	0.666	0.709

Figure 8: Result for Gaussian Naive Bayes

Logistic Regression	Feature Engineered Red Wine Dataset	Original Red Wine Dataset
Train Error	0.476	0.553
Test Error	0.482	0.421

Figure 9: Result for Logistic Regression

Neural Network	Feature Engineered Red Wine Dataset	Original Red Wine Dataset
Train Error	0.41	0.41
Test Error	0.42	0.43

Figure 10: Result for Neural Network

References

[1] mcClain, (2019, Jan, 24) "The Culture, Tradition, and History of Wine in America" Available: https://www.mcclaincellars.com/culture-tradition-history-of-wine/

[2] Ben O'Donnel, (2014, May, 14) "United States Now No. 1 in Wine Consumption" Available: https://www.winespectator.com/articles/united-states-now-no-1-in-wine-consumption-49995/

[3] J Olsen, (2014, Dec, 26) "The Culture, Tradition, and History of Wine in America" Available: https://theconversation.com/wine-drinking-in-america-today-35104

[4] Fiore, (2020) "Quality certifications' impact on wine industry assets performance" Available: https://www.agrojournal.org/26/02-01.pdf

[5] JJ Buckley, (2018, Oct, 23) "The 4 Factors and 4 Indicators of Wine Quality" Available: https://www.jjbuckley.com/wine-knowledge/blog/the-4-factors-and-4-indicators-of-wine-quality/1009

[6] A. Legin, (2003, May, 7) "Evaluation of Italian wine by the electronic tongue: recognition, quantitative analysis and correlation with human sensory perception" Available: https://www.sciencedirect.com/science/article/pii/S0003267003003015?casa_token=k8FgIfsmU9UAAAAA:hauPssx0a003IFpH2Vqd0WWZ9ps_Y0CzkGSZereOmwRs52wSpF0p13ggs_AekObHIt1cnDYVA

[7] Paulo Cortez, (2009) "Wine Quality Data Set" Available: https://archive.ics.uci.edu/ml/datasets/wine+quality

[8] Kingma, (2017, Jan, 22) "Adam: A Method for Stochastic Optimization" Available: https://arxiv.org/abs/1412.6980

[9] Spriggs, "SENG474: Wine Analysis" Available: https://webdocs.cs.ualberta.ca/~alona/dm_projs/wine_final_report.pdf

[10] Chirag, (2021, Mar, 19) "Multicollinearity in Data Science" Available: https://www.analyticsvidhya.com/blog/2021/03/multicollinearity-in-data-science/

[11] Jason, (2020, Aug, 18) "How to Remove Outliers for Machine Learning" Available: https://machinelearningmastery.com/how-to-use-statistics-to-identify-outliers-in-data/

[12] Mortaza, (2007), "Advances in Analysis of Mean and Covariance Structure when Data are Incomplete" Available: https://www.sciencedirect.com/topics/mathematics/nonnormality

[13] (2021, Jul, 20) "Python Box-Cox Transformation" Available: https://www.geeksforgeeks.org/box-cox-transformation-using-python/

[14] Baeldung, (2020, Jun, 26) "Advantages and Disadvantages of Neural Networks" Available: https://www.baeldung.com/cs/neural-net-advantages-disadvantages

[15] Kaichou, (2019, Sep, 26) "How Does Learning Rate Decay Help Modern Neural Networks?" Available: https://arxiv.org/pdf/1908.01878.pdf

[16] Jason Brownlee, (2018, Aug, 6) "How to Configure the Number of Layers and Nodes in a Neural Network" Available: https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/

[17] Inara, (2021, Jan, 3) "Cross-Entropy Loss in ML" Available: https://medium.com/unpackai/cross-entropy-loss-in-ml-d9f22fc11fe0

[18] Paulo Cortez, (2009, May, 16) "Modeling wine preferences by data mining from physicochemical properties" Available: https://www.sciencedirect.com/science/article/pii/S0167923609001377?via%3Dihub