Gabe B. Wenchel
Data Bootcamp Final Project
Professor Jacob Koehler
May 9th 2025

Decoding the Taste: Predicting Wine Quality Using Machine Learning

1. Introduction

The dataset I have chosen is provided by the UCI Machine Learning Repository, it is about the chemical composition of red wines. The data ranges from measurements of fixed acidity, pH, to sugar content, and density, ultimately with their quality rating. While to the average person, a wine's quality may be proportional to the cost of the bottle, for the professional drinkers, or sommeliers, a wine's quality is objective and can be analytically assessed. The merit in being able to understand the relationship between a wine's composition and quality can be useful for people in wine production, or people who want to sound pretentious. Thus, this predictive model aims to identify the characteristics that contribute to a wine's quality and will try to minimize the mean standard error of predicting quality. To ensure prediction is as accurate as possible, we will utilize four separate predictive models, ranking them amongst each other and a benchmark or baseline prediction. In summary, the model that had the best predictive power (MSE of 0.4905) was a neural network.

2. Data Description

As previously mentioned, the dataset is from the UCI Machine Learning Repository, though it was re-uploaded on Kaggle. Furthermore, due to privacy and other reasons, the names of the wines and their grape varieties are not included in the dataset. Without further ado, below is an overview of the various columns of the dataset.

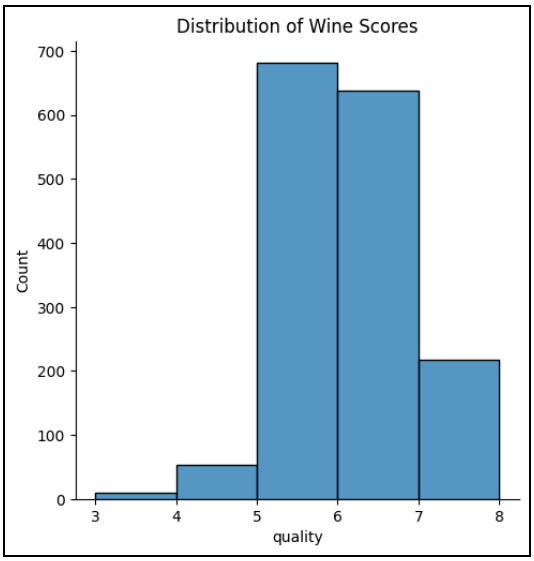
fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality

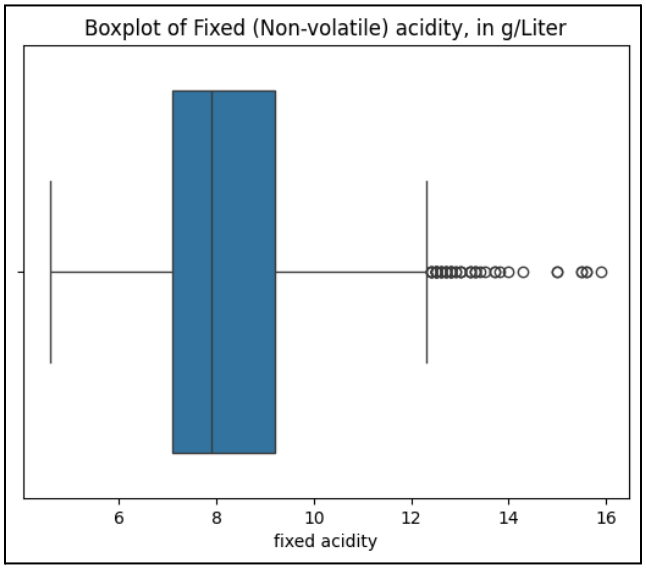
The data consists of 1599 entries, and thankfully due to a reliable provenance, has no issues with inconsistent, missing, or otherwise messy data. Furthermore, the data, owing to its nature, are all continuous floats, with the exception of the discrete quality column.

2.1 Exploratory Data Analysis

As with any data analysis, an Exploratory Data Analysis (EDA) should be done to explore some relationships that may or may not be useful in our final predictive model.

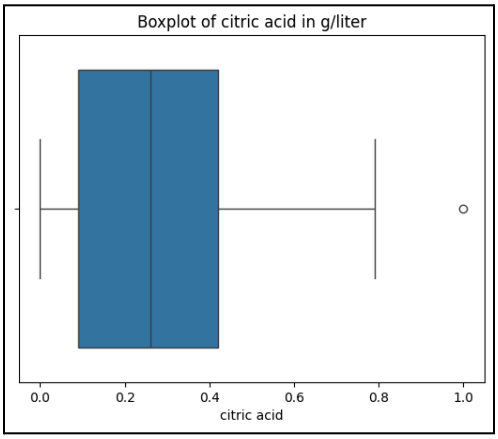
To begin, this is a histogram of our wine qualities. Immediately, we notice that the majority of wines are clustered around the 5-6 quality rank. In fact, there are only 18 wines that have a quality of 8, and none above that.

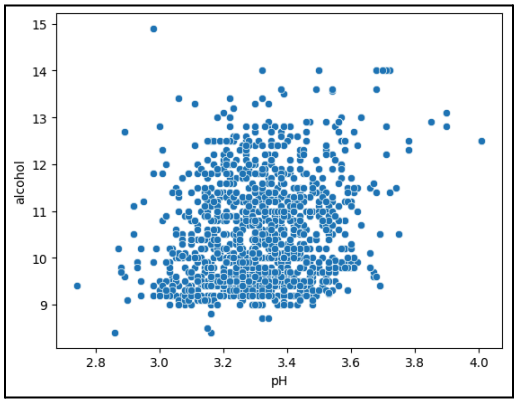




To our left is a boxplot of fixed acidity in g/Liter for all our data. Fixed acidity refers to the presence of acids such as tartaric, malic, and succinic acids. This reveals that 75% of wines are generally low in their fixed acidity, which is to say that the top 25% have disproportionately large amounts of fixed acidity.

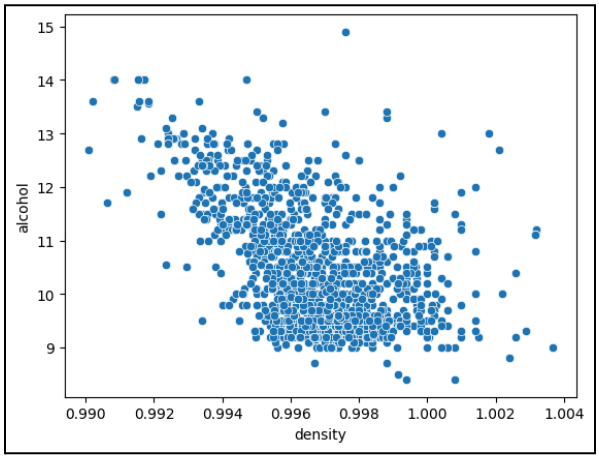
Citric acid is in fact also considered a fixed acid, but the dataset gave it its own column due to its importance. The story is similar to that of the other fixed acids.





While on the topic of acids. Here is a plot of pH and alcohol (ethanol). There seems to be a somewhat weak positive relationship between the two. However, recalling basic chemistry, ethanol has a pH similar to that of water (7) which is to say that its effect on pH is less predictive than anticipated.

Lastly, here is another interesting relationship. Density, which is g/ml, where water is very close to 1, plotted against alcohol. Here we see a strongly negative trend, this also makes sense considering alcohol is less dense than water, which implies that the more alcoholic a wine, the less dense it is. What I thought was interesting were the wines that are denser than water.



3. Models & Implementation

In class we learned several different predictive models, including linear regression, Neural Nets, and others including random forest predictions.

For my dataset I chose to compare four models, Linear Regression, K-Nearest Neighbors, Random Forest, and a Neural Net to see which can provide the best prediction, measured by Mean Standard Error.

To begin, I ran a benchmark prediction, which in layman's terms is guessing randomly. This was executed by this code:

```
y = df['quality']
baseline_preds = np.ones(len(y))*y.mean()
mse_baseline = mean_squared_error(y, baseline_preds)
mse_baseline
```

Which yielded a baseline MSE of 0.652. The magnitude checks out, as it is predicting quality which has values ranging from 3 to 8, so a MSE of less than 1 is already a good sign.

Prior to beginning the modelling, we will create a train test split for our data, using a test size of 20% and a random state of 101. Furthermore, as previously stated, our data is already entirely numerical thus we will not need to encode it.

3.1 Linear Regression

The linear regression is likely one of the simplest to set up, needing little code.

```
lrmodel = LinearRegression()
lrmodel.fit(X_train,y_train)
```

After instantiating it, it had a test MSE of 0.526, which is a very good start.

```
Train MSE: 0.3911585968079635
Test MSE: 0.5261410255981681
Baseline MSE: 0.651760539830827 7
```

For additional analysis, I ran a feature importance function to see in a linear regression which features affect quality the most. Interestingly, alcohol has by far the greatest impact, which to me was surprising as I imagined it would be more complex.

fixed acidity	-0.004857
volatile acidity	0.108175
citric acid	0.015811
residual sugar	-0.000725
chlorides	0.008988
free sulfur dioxide	0.005670
total sulfur dioxide	0.018540
density	0.000590
pH	0.009272
sulphates	0.051760
alcohol	0.261270

3.2 K-Nearest Neighbors

For the KNN regression we must scale our data as it is a distance based algorithm (Euclidean distance to be exact). So we create a pipeline that uses a standard scaler and applies it to our data. Then we create a parameter grid and use GridSearch to find and fit our model. We then save the best result. In this case the best K was 20, and it predicted a test MSE of 0.525 which is minutely better than the linear regression.

```
KNN Test MSE: 0.5253749999999999
KNN Train MSE: 0.3778127443315089 7
```

Seeing as this was not much of an improvement we will move on to the random forest.

3.3 Random Forest

For the random forest model we used a random state of 101 and used these parameters for the parameter grid.

```
] param_grid = {
        'n_estimators': [50, 100, 200],
        'max_depth': [None, 5, 10],
        'min_samples_split': [2, 5],
        'min_samples_leaf': [1, 2]}
```

After saving the best parameters, we ran our model.

```
Best parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
```

This produced an MSE of 0.425, significantly better than the previous two.

```
Random Forest Test MSE: 0.4249024999999996
```

fixed acidity	0.004746
volatile acidity	0.081349
citric acid	0.010359
residual sugar	0.004354
chlorides	0.018803
free sulfur dioxide	0.012206
total sulfur dioxide	0.043600
density	0.003373
pH	0.015498
sulphates	0.156048
alcohol	0.341506

I decided to also look at the feature importance for the random forest model. Here we see that alcohol is still the most influential parameter, but we see that sulphates which were much less significant in the linear regression, are playing a greater role in the random forest model. Suggesting that the sulphates relationship to quality is not linear.

3.4 Neural Net Model

For our neural network, we had to briefly treat our data. First I used the Standard Scaler again to scale our data. Then I had to convert our data into torch tensors for them to be usable in our model.

I chose my neural network to have **2 hidden layers** with 32, and 16 nodes respectively.

```
nnmodel = nn.Sequential(nn.Linear(X_train_tensor.shape[1],32), nn.ReLU(),
                        nn.Linear(32,16), nn.ReLU(),
                        nn.Linear(16,1))
```

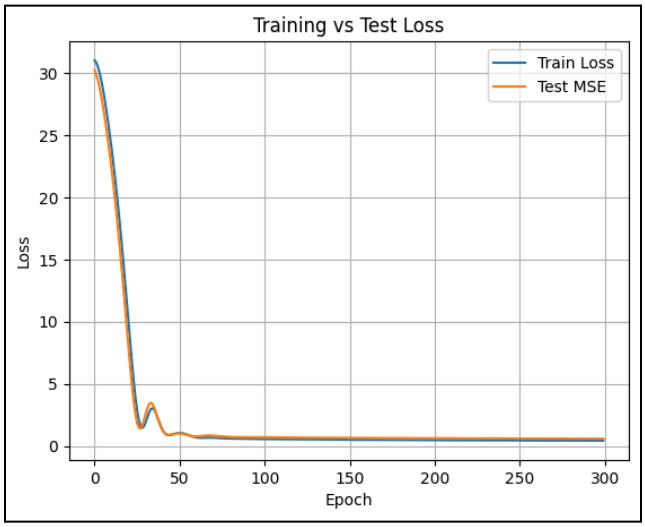
Furthermore, this is my loss and optimizer functions. I chose SGD as I tried Adam and it worked less reliably. Furthermore, I also chose a lower learning rate as a higher one began overfitting. The momentum was added to help correct it.

```
loss_fn = nn.MSELoss()
optimizer = optim.SGD(nnmodel.parameters(),lr=0.001,momentum=0.9)
```

The neural net model unfortunately fell short of expectations, instead it actually fared worse than all other models with an MSE of 0.581. Likely this was due to either overfitting, the data not being very appropriate for a neural network, or another factor.

```
Best Test MSE: 0.5813
```

Here is a comparison of the evolution of the neural network for both the test and training loss. We see that it learned rapidly in the first 50 epochs, however, afterward the progress was very minimal.



4. Results and Interpretation

Name of Model	Test MSE	Train MSE
Baseline	0.652	
Linear Regression	0.525	0.391
KNN Regression	0.526	0.379
Random Forest	0.425	0.0427
Neural Network	0.581	0.422

From this table we can clearly see that the random forest algorithm by far was more accurate than any of the other models.

There were a few reasons I could think of as to why the random forest algorithm performed better:

1. Can handle multicollinearity better
    a. As mentioned earlier, citric acid is correlated to fixed acidity, so the random forest algorithm could likely adapt to this.
2. Can also handle non-linearity better.
3. Adept at threshold based rules
    a. Similar to a decision tree, the algorithm can make decisions based on thresholds which coincides with the non-linearity point, too.

Some other reasons why other models maybe performed worse than expected:

KNN:

1. Distance based algorithm can mean that some variables lose meaning when it is all scaled down, i.e. alcohol and pH

Linear Regression:

1. Variables' effect on quality is likely not a linear relationship.

Neural Network:

1. Dataset is only around 1600 entries, which means it may overfit.

5. Conclusion and Next Steps

In this project, I explored four predictive modeling techniques—Linear Regression, K-Nearest Neighbors, Random Forest, and a Neural Network—on a dataset of red wine chemical properties with the goal of predicting wine quality. Among all approaches, the Random Forest model achieved the best performance, with a test MSE of 0.425, outperforming both simpler models and more complex neural networks.

The Random Forest likely succeeded due to its ability to:

- Handle non-linear relationships,
- Manage feature interactions and correlated variables, and
- Leverage threshold-based splits, which suit this type of tabular data well.

For future work, I would consider applying cross-validation to better evaluate model stability, and performing more thorough hyperparameter tuning on the Random Forest model. Additionally, exploring feature engineering, such as creating interaction terms, could help simpler models like linear regression perform better. I could also experiment with model ensembling or reframing the task as a classification problem (e.g., high vs. low quality) to see if that improves accuracy.