**Multilabel Wine Quality Classification Using Decision Trees, Neural Networks, and K-Nearest-Neighbors**

Kellie Behm, Cole Kassing, Cole Scarbough

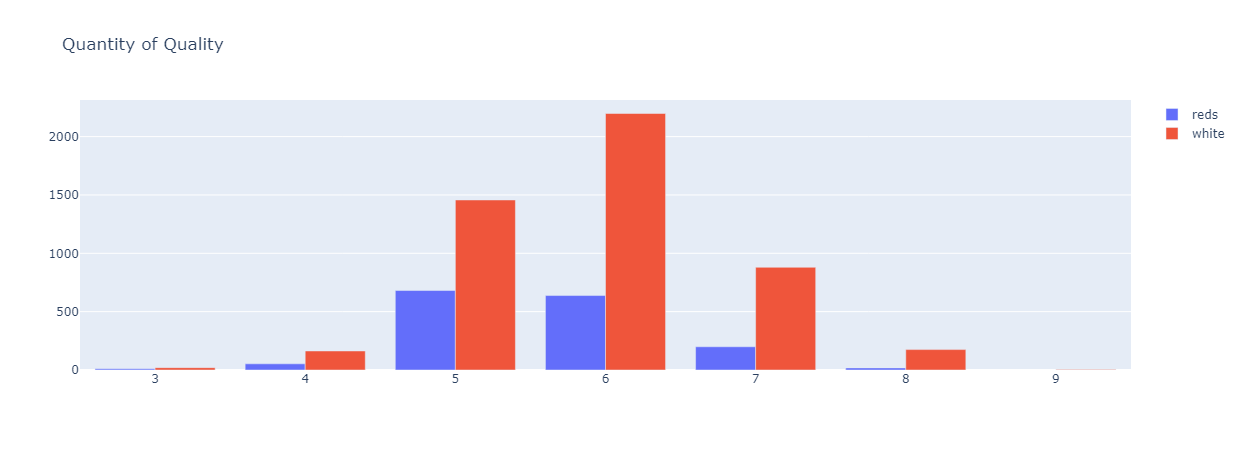
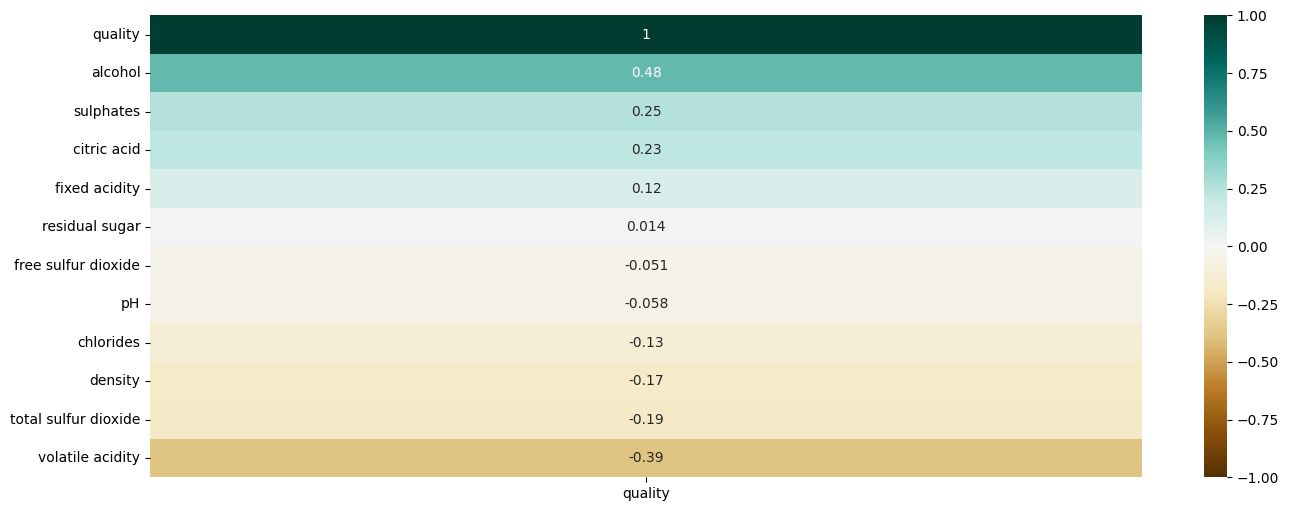
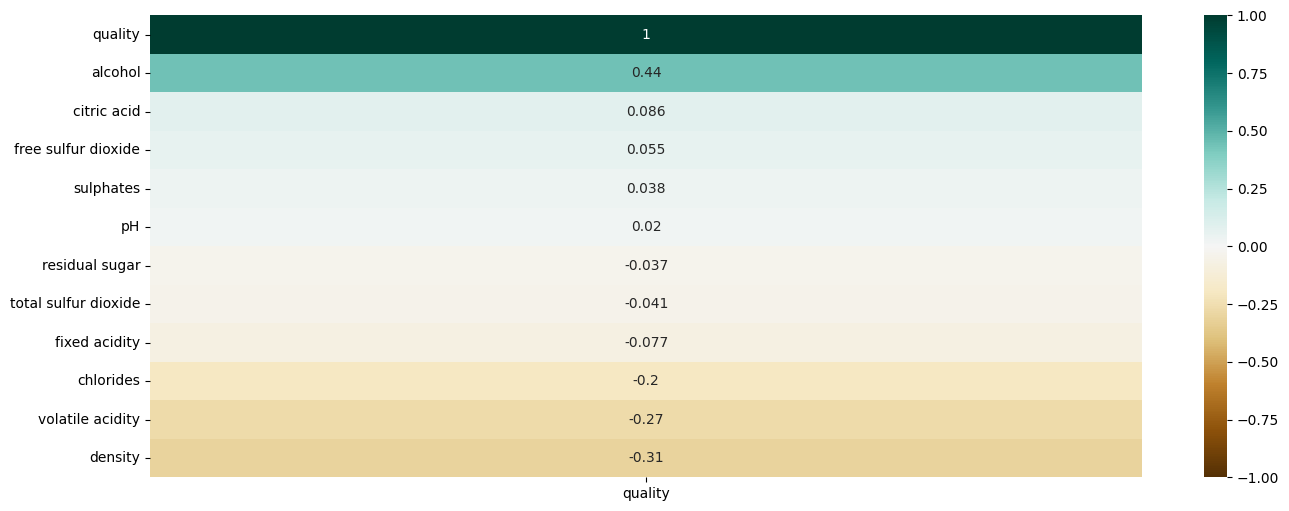
**Abstract:** Even though wine tasting has long been an art for human culture, the ability to classify wines presents itself as a machine-learning problem. In our paper, we explore methods to classify wine with different types of machine-learning models on chemical data of high and low quality wine. Additionally, we performed Exploratory Data Analysis (EDA) on our data. EDA is the process of “identifying obvious errors, as well as better understanding patterns within the data, detect outliers or anomalous events, find interesting relations among the variables.” (1). After conducting EDA on our data, we tried three machine-learning problems classifying red wine into different predetermined qualities of 3,4,5,6,7 and 8. The three models we attempted to use were: a single layer neural-network, k-nearest-neighbors (KNN), and XGBoost, a gradient-boosted ensemble method that uses decision trees. We found that KNN had the best performance with a 61% f1 score. Additionally, we used XGBoost on a combined dataset of both wines to measure the effect of combining the red and white wine datasets; we observed red wine alone having a 59% f1 score and total wine having a 53% f1 score.

**Introduction:** The overarching problem is to take data containing information about specific wines and to be able to determine their quality. The data we chose to use was “Wine Quality” from Paulo Cortez at the University of Minho, Guimaraes, Portugal (2). However, we obtained this data through the University of California Irvine’s Center for Machine Learning and Intelligent Systems repository (4). The data consists of two datasets: “winequality-red”, which contains only red wine, and “winequality-white”, which includes white wine data. Both contain red and white variants of the Portuguese “Vinho Verde” wine (2). Moreover, Cortez points out that the dataset is imbalanced regarding quality, meaning we cannot use traditional accuracy measures. Additionally, the red wine dataset has 1599 observations, while the white wine dataset has 4898 observations, with neither missing any values for any attribute. Below is a table containing the descriptions of the attributes in both datasets (3) - all numerical**.**

|  |  |
| --- | --- |
| Attributes | Description |
| fixed acidity | Most acids involved with wine or fixed or nonvolatile (do not evaporate readily). |
| volatile acidity | The amount of acetic acid in wine. |
| citric acid | Found in small quantities, citric acid can add “freshness” and flavor to wine. |
| residual sugar | The amount of sugar remaining after fermentation stops. |
| chlorides | The amount of salt in the wine. |
| free sulfur dioxide | Free form of S02; prevents microbial growth and the oxidation of wine. |
| total sulfur dioxide | Amount of free and bound forms of S02; free S02 concentrations over 50 ppm, becomes evident in the nose and taste of wine. |
| Density | The density of a substance is its mass per unit volume. |
| PH | Describes how acidic or basic a substance is 0-->14 |
| Sulphates | A wine additive, which acts as an antimicrobial and antioxidant. |
| Alcohol | The percent alcohol content of the wine |
| Quality -- class | Score between 0 (low quality) and 10 (high quality). Integer |

**Other approaches:** Most attempts attempted to slightly modify the original problem. Instead of multilabel classification, other experiments classify the data as a binary problem. To classify the red wine dataset Vishal Kumar chose to split the red wine dataset based on the quality label into [0-6] to 0 and [7-10] to 1 (5). This led to varying f1 scores with different algorithms such as random forest (87%), support vector classifier (86%), and stochastic gradient descent classifier (84%). In our experiments (KNN, Single Layer Neural Network, and XGBoost) on red wine alone, all performed less than binary classification. Additionally, many other attempts to classify this dataset only work on red or white wine, not combined. In our approach, we wanted to measure the effectiveness of trying a model on both datasets combined, to do this we used XGBoost.

**Our Approach:** Our first step in our experiment was to analyze the data in a raw format. We chose to use red wine as our dataset for the three separate implementations of models. First, we checked the overall balance of the datasets regarding quality.

There appeared to be a similar distribution of types of classes between white and red wines. It appeared that the dataset is very imbalanced as 5 and 6 contained the most objects. Moreover, feature extraction was necessary to determine the features related to quality; to do this, we used a heatmap based on the correlation coefficients of attributes to quality. Correlation coefficients can be roughly described as “how strong a relationship is between data” (8). These values can range from –1 to 1 and represent the negative or positive correction, meaning the correlation of two types of data is 1, there is a pure direct relationship (both increase together), while if that correlation is –1, the data has a purely inverse relationship. Therefore, we wanted attributes close to –1 or 1.We found that the attributes of residual sugar, free sulfur dioxide, and pH had a low correlation with quality. Similarly, we performed correlation analysis on the total wine dataset. We split the red wine data into a ”red\_train” (0.8) and ”red\_test” (0.2) dataset and stratified the data according to the distribution of quality. We also combined the white and red wine datasets into ”winequality\_total,” which we split into train and test sets with the same method as red.  
We found that there are less attributes that have a strong correlation in the total wine dataset.  
  
**Preprocessing:** After the train and test sets were loaded, additional preprocessing comprised scaling the data for the K-nearest neighbors and neural network models. The motivation for scaling is to ensure all attributes have comparable means, standard deviations, or ranges so that the model treats each attribute equally important rather than emphasizing the disparity of raw values. Additionally, in the red wine dataset, our class values of “quality” ranged from [3 - 8], and were mapped to a range of [0,5]. However, this step was only required for the Neural Network and XGBoost implementations, as not all qualities were present. Both algorithms expect only the number of present classes to be listed for prediction. In our implementation with XGBoost with the total wine dataset, we mapped values ranging from [3 – 9] to a range of [0 – 6]. Finally, we considered the aspect of outlier removal; we applied the Tukey Rule. The Tukey Rule states that if a value is outside of a boxplot’s lower or upper fence, you consider it an outlier. We defined an observation as containing two or more attributes with values from the upper or lower fence. In both red wine and total wine, there were very few. As we had an imbalanced dataset with small quantities in some classes, we decided to leave the outliers in – to preserve the outliers representing classes with little representation.

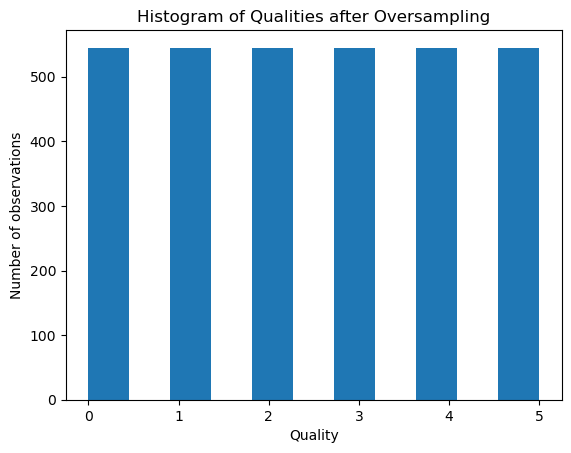
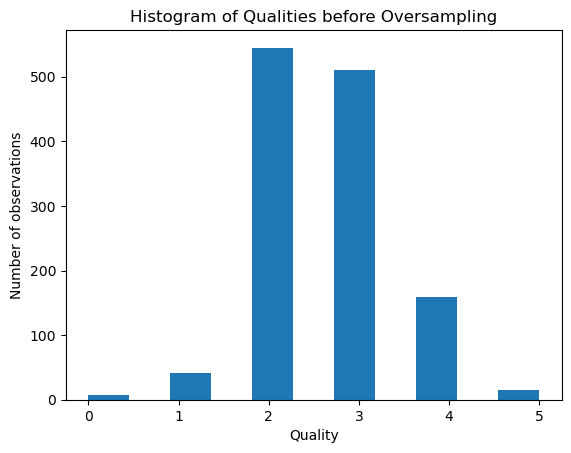
**Necessary libraries:**

|  |  |
| --- | --- |
| General Utility + Data Visualization | numpy, pandas, sklearn, matplotlib, plotly, seaborn |
| Neural Networks | numpy, pandas, matplotlib, tensorflow, keras, sklearn |
| K-Nearest-Neighbor | numpy, pandas, matplotlib, sklearn |
| XGBoost | XGBoost |

**Experiments:**

**Single Layer Neural Network:** We first built a model using the data in its minimally preprocessed form. We built a model with as many inputs as attributes and six outputs for the six possible class labels. The 'softmax' activation was chosen to output the probability of each class label being correct. The label with the highest probability is thus the predicted label for that observation. With such a small data set and heavily imbalanced class representation, high accuracy was not an expectation. However, several methods to improve the model's performance were explored, motivating additional preprocessing steps (scaling) and methods of dealing with imbalanced data (oversampling and using class weights). The technique of the latter two methods was supplemented with a GitHub article on the topic (7).

The next model uses data that has been scaled using the min-max scaler from sklearn. The statistical summary of each column motivates the need to normalize the different attributes' data. After scaling, we can see that each attribute aligns with a range of [0,1]. We build, summarize, compile, and finally run the model as before.

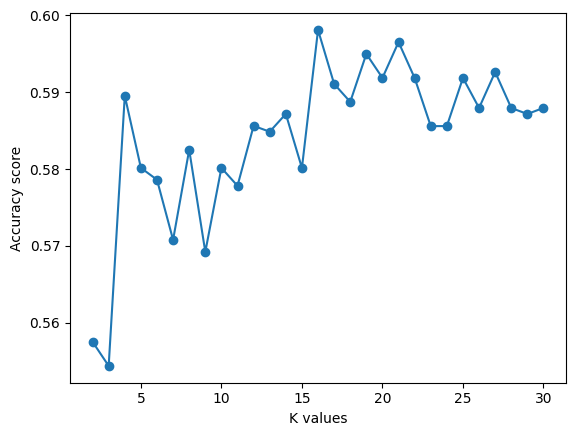
The next approach is the first method for dealing with imbalanced data. To create a balanced data set, we can use either undersampling (subsetting the overrepresented classes to make each class equal in number) or oversampling (sampling with replacement of the underrepresented classes to make each class equal in number). We chose to oversample to fully utilize our already-small data set. We found that the quality rating 5 had the most observations (545), so each other class value was sampled 545 times with replacement.

The alternative method for dealing with imbalanced data utilizes a specific parameter in the tensorflow model function called class\_weights. Providing a dictionary of corresponding class labels and weights in the setup of the model tells the model to assign initial “importance” to the different class values based on how well-represented in the data set they are. This counteracts the possibility of the model developing a bias towards majority classes.

Each of these models (raw, scaled, oversampled, and weighted) are built, fit, compiled, and run. Accuracy, precision, recall, and F1 score is output for each model so we can see how they compare. The results show that the weighted model outperforms the scaled and oversampled models in both accuracy and F1 scores. The weighted model performs with 42.19% accuracy and an F1 score of 25.22%. Although the scores for the weighted model are still low, this is expected given the relatively small number of observations and single-layer structure of the neural network.

**K-Nearest Neighbors:** The KNN algorithm works by finding the k-nearest data points in the training set to the test instance and then predicting the class based on the majority vote of the k-nearest neighbors. To help formulate this algorithm, we took in additional data from a website specializing in KNN and implemented it using our data (6).

After preparing our data, we train a KNN classifier with k = 5 on the training data and store the feature values(X\_train) and target values(y\_train) in memory so we can make predictions on the test data. Our prediction accuracy came out to be 72%. To see what the best k number should be used, we compute the cross-validation scores over a range of 2-31, to better visualize this, we plot the points in a “K values vs. Accuracy Score” graph using Matplotlib.



Examining the graph, our best k score would be 22; we then train a classifier with this value and compute the scores. These scores include accuracy, precision, and recall, and since the data is imbalanced, the F1 score. In conclusion, our results show that the KNN classifier performs well with our dataset, with the accuracy and recall being 63%, precision being 64%, and our F1 score being 61%.

**Decision Trees (XGBoost)**: We chose XGBoost, or Extreme Gradient Boosting, which is a decision tree algorithm that uses boosting. Boosting is the process of building models sequentially by minimizing the errors from a previous model, which is a form of ensemble learning – using multiple algorithms for some other goal. This means that you have weak learners and strong learners. Weak learners are models typically with low accuracy that explore a dataset; these weak models aim to be converted into strong learner. In the case of XGBoost, it is a sequential boosting model that combines several weak decision trees sequentially. Weak models in XGBoost usually are referred to as base learners since XGBoost uses gradient descent to optimize the process of generating models, which, instead of fixing errors, tries to get the best results from creating better base models. Gradient descent is the method of reducing some value by systematically changing a model’s data or structure. This value is called the loss function, and the model’s performance increases as loss decreases. In the case of XGBoost, it typically specifies where a specific tree will split a current node and the importance of certain attributes. XGBoost is “extreme” because it can be run on multiple cores, making it much quicker than typical gradient boosting methods (9).

For hyperparameter tuning, we used a RandomizedSearchCV. This is imported from Scikit.learn.model\_selection and creates a set number of cross-validation sets (in our case, 5) to randomly test a fixed amount of hyperparameter choices (in our case, 25). A method like this runs 125 times, and we seeded the search, so it should produce the same results every time. We did this search in all three cases: once for the model predicting only red wine, once for the model predicting total wine but using the same attributes as red wine, and once for the model predicting total wine but only using attributes with a high correlation with total wine. For the XGBoost generally, we had also to specify the evaluation metric and objective. The objective specifies how the decision trees in XGBoost define the classes. Since we were doing multilabel classification, we had to set the objective to “multi: softprob”. This objective allows for more than binary classes in XGBoost. Additionally, we use the area under the curve (AUC) as our metric for the gradient-boosting algorithm; XGBoost does not provide f1 score for a measure, but AUC still works on multilabel classification. AUC takes a plot of all the correct and incorrect classifications as a function and measures how much area is covered, I.e., the integral (10). We used average scores over five tests to measure performance, which were:

Red Wine = {f1 score: 59%, recall: 62%, precision: 65%, accuracy: 62%}

Total Wine (Red Wine Attribute) = {f1 score: 53%, recall: 55%, precision: 57%, accuracy: 55%}

**Conclusion:** In our experiment on the red wine data set, KNN had the highest accuracy. This is because there was not enough data for our Neural Network to work, and KNN has a better time with purely numerical data. Additionally, our models were around 20% more inaccurate than binary classification methods. This makes sense, as having only two different classes is much easier than 6 in the red wine models and 7 in the total wine dataset. When using, the same features present in both datasets led to worse overall accuracy. When comparing the results of the XGBoost model using the total wine dataset (53%) vs. the XGBoost model with only red wine (59%), we see a 6 percent decrease. This is likely because we are mixing two datasets that are similar but are not similar enough as they create too much noise. Without more data, the model is confused and cannot accurately classify the dataset.

**Members Contributions:**

Cole K: EDA, XGBoost, Report/Presentation (XGBoost, Intro, Abstract, Conclusion, Approaches)

Cole S: KNN code and KNN results

Kellie B: NN code and NN implementation & results of report, NN parts of presentation

**References:**

(1) <https://www.ibm.com/topics/exploratory-data-analysis>

(2) 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

(3) <https://rstudio-pubs-static.s3.amazonaws.com/137418_3b37b9fad4284a5d9ff265cdf1d97120.html>

(4) <https://archive.ics.uci.edu/ml/datasets/Wine+Quality>

(5) <https://www.kaggle.com/code/vishalyo990/prediction-of-quality-of-wine>

(6( <https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn>

(7) <https://github.com/christianversloot/machine-learning-articles/blob/main/working-with-imbalanced-datasets-with-tensorflow-and-keras.md>

(8) <https://www.statisticshowto.com/probability-and-statistics/correlation-coefficient-formula/>

(9) <https://aws.amazon.com/what-is/boosting/>

(10) <https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc#:~:text=AUC%20stands%20for%20%22Area%20under,across%20all%20possible%20classification%20thresholds>.