A Study of Gradient Boosting: Performance Analysis on Wine Quality Prediction using XGBoost

Minh Ngo and Ian DeLano
University of Florida
STA 4241 Final Project

April 23, 2025

1 Introduction

Predictive modeling is a foundational tool used in data science and plays a vital role in a wide range of disciplines, particularly in contexts where the objective is to provide accurate decisions based on the analysis of structured feature sets. With more complex datasets, there are various options for statistical models that can help balance predictive accuracy, flexibility, and interpretability. Decision trees are one of these models that are commonly used because of their straightforward implementation and comprehensibility. They serve as a good baseline for data analysis, as decision trees can be applied to both regression and classification problems. A simple decision tree works by recursively splitting the data into subsets based on varying threshold values. It starts from the root node splitting into internal nodes that send data down varying branches of the tree; this process continues until the data reach a leaf node that represents the prediction of that model given the input features. The basic structure enables an easy interpretation of the data; however, when used alone, the results often fall short of ideal. A tree that is too shallow or too deep can overfit or underfit the data. Moreover, the overall performance of a decision tree is highly sensitive to small changes in the data which can cause performance issues in terms of generalization; leading to inaccurate predictions. To tackle these limitations, methods such as gradient boosting were introduced to help construct stronger models through the use of ensemble learning, where by combining multiple weak trees it is able to form a more robust and accurate model that is capable of handling complex patterns in datasets.

Gradient boosting aims to improve the performance of weaker models. It accomplishes this via training weak trees, building them sequentially. The basic concept is centered on the idea that each new tree will seek to improve on the errors made by its predecessors. Through multiple stages, the overall performance of the model at each step will improve, with the goal of achieving a model that is more capable of reducing errors while improving accuracy compared to a singular decision tree. A common and widely used implementation of gradient boosting in application, is through a library known as XGBoost. This publicly available library is a specific type of gradient boosting, more specifically Gradient Boosted Decision Trees. Due to its efficiency and fast computation time when handling high-dimensional datasets it is a widely popular machine learning tool for many data science applications. In this project, we will apply XGBoost as the main method of application to demonstrate gradient boosting in action and evaluate its performance in comparison to a

single decision tree. By applying both methods to the same dataset, we seek to understand whether gradient boosting truly has meaningful improvements in model accuracy.

In order to carry out the gradient boosting demonstration mentioned previously, we will analyze a publicly available Wine Quality dataset. This dataset includes a variety of input variables based on physicochemical tests such as acidity, alcohol content, pH, along with a quality score rating ranging from zero to ten. This dataset enables us to approach the prediction task in two different ways: as a regression task where quality score is a continuous score and as a classification task, predicting wine quality score as discrete categories instead. We will apply both a single decision tree and a gradient boosting model using XGBoost to this dataset and evaluate their performance using the appropriate metrics corresponding to the two tasks. In doing so, the aim is to determine the improvements that gradient boosting has for complex datasets compared to a basic decision tree. The paper will include an explanation of the mathematical ideas behind gradient boosting, followed by the application and analysis of gradient boosting through a data analysis of the wine dataset, along with this will be a final analysis of the results.

2 Mathematical Background and Methodology

Gradient boosting refers to an ensemble modeling method that improves prediction accuracy through the use of a series of weak learners combined sequentially. The accuracy is improved by minimizing a loss function. The intuition behind this is that each new weak learner prioritizes fixing the errors made in the current model. In contrast to other ensemble methods like random forest or bagging, gradient boosting creates the model in separate stages, whereby each new component is looking to improve on the previous one. The resulting formula is an additive model expressed as:

$$F_m(x) = F_{m-1}(x) + h_m(x) \quad [1]$$

Where $F_m(x)$ is the prediction model at iteration m, and $h_m(x)$ being the weak learner that is used to approximate the negative gradient of the loss function at the current stage m. The following sections will dive into a deeper understanding of this procedure including the role of residuals, loss minimization, and the use of decision trees as base learners.

Following the previously introduced additive model, we know that gradient boosting adds $h_m(x)$ at each step to iteratively reduce error. However, rather than retraining each individual components from scratch the model will implement a stage-wise approach. We want to find the function $h_m(x)$ that, when added to the current model $F_{m-1}(x)$, will best minimize the overall loss function, At each iteration m, the function $h_m(x)$ can mathematically be identified as:

$$h_m(x) = \arg\min_h \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + h(x_i)\right) \quad [1]$$

Here, L is a chosen loss function and $h(x)$ is trained to improve the residual errors of the previous model.

This stage-wise optimization allows the model to focus on its current shortcomings and gradually improve. The specific choice of loss function plays a central role in guiding these updates and typically depends the nature of the prediction task at hand. These loss functions guide how each weak learner is trained [2].

In regression problems, the squared error loss function is often applied, and is defined as follows:

$$L(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2 \quad [2]$$

For classification problems, a different loss is used, the deviance loss, which is equivalent to log loss:

$$L(y_i, \hat{p}_i) = -\log \hat{p}_{i,y_i} \quad [2]$$

The main idea behind this is that it fits new models in the direction where the current model is underperforming. This direction of improvement is given by the loss function gradient with respect to the latest output of the model. For the squared error case, the function becomes $y_i - F_{m-1}(x_i)$, or the residual. Generally, this process mirrors gradient descent but instead of operating in a parameter space, it happens within a function space. In traditional gradient descent, the optimization updates the parameters of a fixed model (e.g., weights) to reduce the loss [3]. However, with gradient boosting, instead of just tweaking parameters, it builds functions sequentially using the weak learners. You can think of a function space as a location in which each point represents an entire function, and we navigate that space by adding new learners that reduce the loss. Each of these new learners serves to guide the model toward better performance step-by-step in an iterative fashion.

While fitting new weak learners can improve the model's overall accuracy, doing so too aggressively and without caution can lead to a model that is overfitting of the data, making the model unreliable in real-world applications. To reduce this, gradient boosting introduces a learning rate, η, which controls how much each new step contributes towards the final model. The additive formula now becomes:

$$F_m(x) = F_{m-1}(x) + \eta \cdot h_m(x) \quad [1]$$

Where η is a value between 0 and 1 [2] . The idea behind this is that rather than each step making a full correction, the model opts to adjust with smaller and safer steps [1][2]. It encourages the model to improve steadily allowing for more corrections over a longer period of time. A smaller learning rate generally means that more trees are required, but the resulting model often adapts better to new data. There is a tradeoff however, as faster learning risks overfitting, slower learning comes with stronger final performance at the cost of computational time [2].

In gradient boosting, the weak learners that are most commonly used are shallow decision trees. While trees are not the only way gradient boosting can be applied, it is the most common method. These trees are intentionally kept simple with a low amount of depth. In essence, they are small rule makers that capture tiny patterns from the data through each iteration. This simple nature is on purpose as it encourages the model to build complexity over time rather than fitting an entire dataset at once. In this study's implementation, the XGBoost library will be applied following the principles outlined here, but with overall improved optimization built within the library. This is particularly why XGBoost is popular as it is an effective tool when working with complex nonlinear datasets. To illustrate how gradient boosting works in practice, we

can view a general outline of the algorithm. We can see that the method iteratively introduces new models, guided by the negative gradient of a loss function, to correct the mistakes of the ensemble. The following is a formulation of the gradient boosting algorithm [4] [5].

1. **Initialize:** $F_0(x) = \arg\min_\gamma \sum_{i=1}^{n} L(y_i, \gamma)$
2. **For** $m = 1$ to M:
   (a) Compute pseudo-residuals: $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$
   (b) Fit weak learner $h_m(x)$ to $\{x_i, r_{im}\}$.
   (c) Update model: $F_m(x) = F_{m-1}(x) + \eta \cdot h_m(x)$
3. **Return:** Final model $F_M(x)$

With the mathematical foundation established, we will now seek to explore a practical implementation of gradient boosting. By applying its iterative, stage-wise approach, gradient boosting enables for the creation of a model that sequentially improves on its previous errors. In the following section, we will apply XGBoost and examine how it compares to a singular decision tree baseline.

3 Model Implementation and Tuning

We will be working with the publicly available Wine Quality dataset. It includes a variety of input features that measures the physicochemical properties of red wine, and each wine sample is then rated by a quality score from 0 to 10 [6]. Prior to modeling , a few steps are taken to prepare the data for analysis along with basic exploratory data analysis of the raw data. To better understand the relationship between variables, a correlation analysis was completed. It is seen that alcohol content had a strong positive correlation with wine quality while volatile acidity and density were negatively correlated , these results suggested that these covariates do have an impact on wine quality score.

To prepare for the data for modeling, certain columns such as 'ID' were dropped to reduce noise as it is not a feature that affected wine quality. The data was then split into two matrices: the feature matrix X where quality was dropped, and matrix Y that solely stored 'quality' variable. Finally, standard scaling was applied to all the input features to ensure consistency across models in order to for fair comparisons.

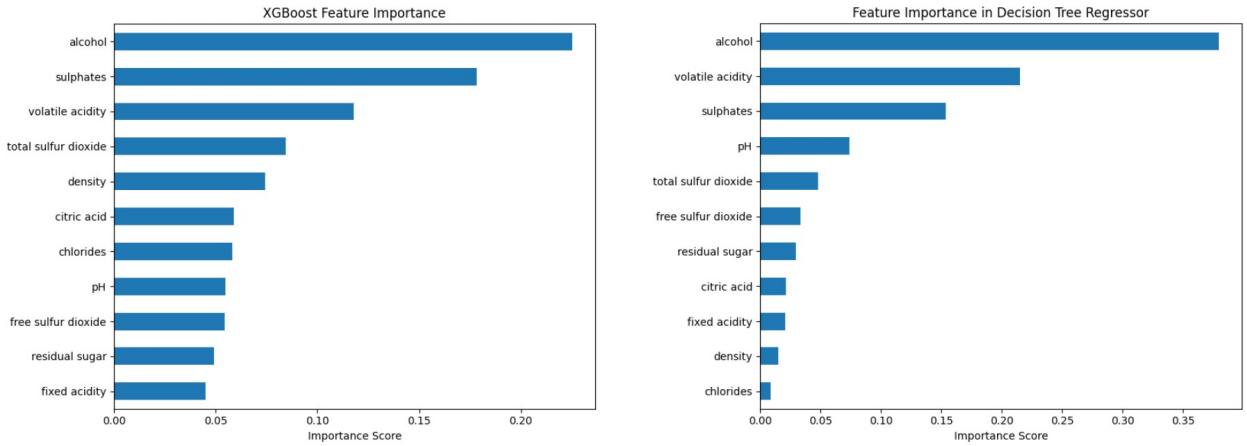
In the regression portion of this study, we implemented two models for this task, a basic decision tree that is simple and easy to interpret and a XGBoost regressor that helps us efficiently apply gradient boosting to build the subsequent model. The data was divided into a standard training and testing set using a 70/30 split. From this, a Decision Tree Regressor was trained employing varying levels of depth to find the optimal parameter. As for the XGBoost Regressor, to improve the overall performance of the model we applied GridSearchCV with a 3-fold cross-validation to perform parameter tuning. The grid included values such as the number of boosting rounds, the learning rate, depth of each tree, and fractions of samples and features used by each tree. This process helped choose the most optimal combination of parameters that minimized the prediction error and generally produced better results compared to a XGBoost model without cross validation. To evaluate the regression models' performance, we used the appropriate regression metrics: Root Mean Squared Error (RMSE) and R-Squared R^2 values.

For the classification task, instead of treating the wine quality score as continuous values, it was transformed into discrete categories instead, thus changing the task from that of regression to classification. Using the same preprocessed and scaled data, we once again applied the same models as before for a Decision Tree and XGBoost, but using the classifier versions instead. Similarly as before, cross validation was done to tune the parameters for the XGBoost Classifier. Lastly, performance was measured using the corresponding classification metrics: accuracy and F_1-score.

4 Results and Evaluation

In this regression task our study aimed to predict wine quality as a continuous variable using the two models. These models being a baseline Decision Tree Regressor with a standard depth and the XGBoost Regressor, one without, and one that applied cross validation to tune its parameters. The Decision Tree Regressor, was kept relatively shallow to avoid overfitting with a depth parameter of 5. Additional increases in depth saw the RMSE value increasing significantly suggesting a model that was overfit. With a depth parameter of 5, the singular decision tree achieved a RMSE value of 0.7103 and an R^2 value of .1324. These metrics clearly suggest limited predictive capabilities, we can see from the low R^2 value that the decision tree ultimately failed to capture the majority of the features in the data. This underperformance was expected, as singular decision trees are not well-suited for complex datasets with many interacting features.

As for the XGBoost Regressor, which was from the implementation of gradient boosting, even without the use of cross validation, the model significantly improved predictive accuracy. Without tuning, the model achieves an RMSE of 0.5848 and an R^2 value of 0.4013. While these values are not ideal, it demonstrated a substantial improvement over the decision tree. Applying GridSearchCV hyperparameter tuning, we were able to further improve on accuracy with the RMSE dropping to 0.5782, and the R^2 rising to 0.4251. While not a large change from the untuned model, it demonstrates how model performance is sensitive to the parameter choices and with tuning it can better optimize the model.



(a) XGBoost Regressor

(b) Decision Tree Regressor

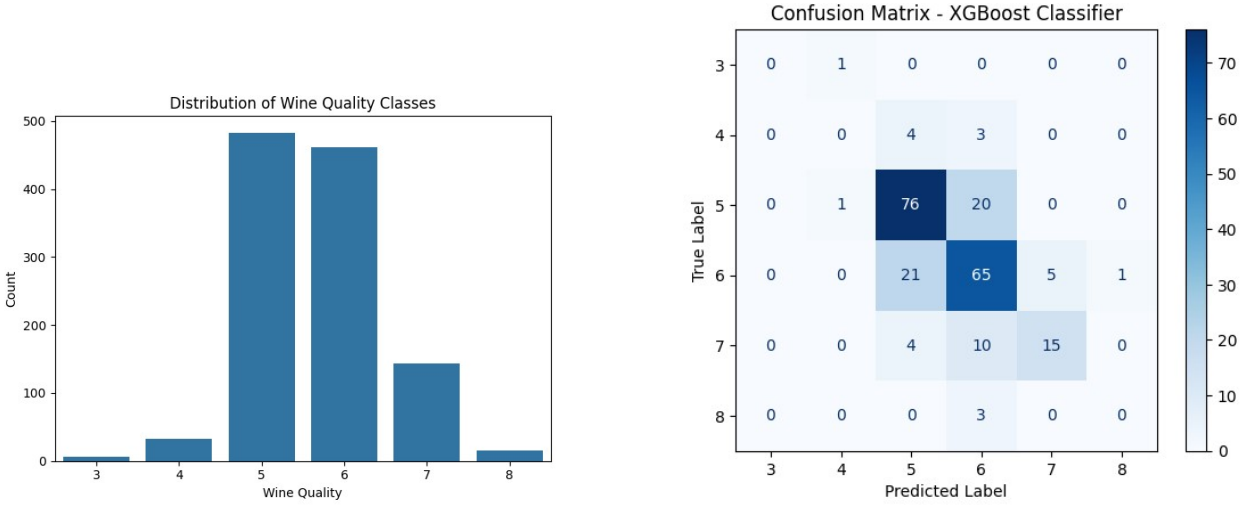
Figure 1: Feature Importance Comparison Between Models

Additionally, feature importance visualizations were generated for both regression models. The XGBoost model's chart (see Figure 1a) highlights alcohol as the most influential feature, followed by sulfates and then volatile acidity. This result aligned well with what we saw in the earlier exploratory analysis of cor-

relation. Compared to the decision tree regressor (Figure 1b), it also ranked alcohol and volatile acidity as key contributors however assigning very little weight to the other features. This is in line with how singular decision trees usually tend to capture dominant features and are prone to overlooking other relationships within the dataset. As for the XGBoost model, it is seen that the model assigned higher importance to the several other features which highlight that while these features are not as dominant, they still play a role in model prediction accuracy. This demonstrates the strength of gradient boosting, through its ability to detect more complex features and utilize weaker features that basic trees perhaps ignore.

As for the classification task, in similar fashion to before, we trained the two types of models but treating the scores this time into discrete classes ranging from three to eight. Due to the lack of available data in wine quality for both extrema, this change was implemented to have a model with less gaps in data which may drag the predictive power of the model down. Starting with the baseline Decision Tree Classifier, it returned a model accuracy of 59.83%, while the XGBoost Classifier once again outperformed the tree model achieving an accuracy of 68.12%. After further tuning using cross validation the accuracy increased by a small amount to 69.43%.

Although there was a slight improvement, a further inspection into the data reveals certain issues with how the quality score was distributed causing imbalance. Figure 2a shows how the data has an uneven distribution of quality rating, with most of the wines being rated in the mid five to seven range and considerably less samples with classes like 3, 4, and 8. It can be assumed that this model directly influenced the classifier's learning causing uneven predictions across classes.



(a) Distribution of Wine Quality Classes

(b) Confusion Matrix for XGBoost Classifier

Figure 2: Visualizations showing data imbalance and model prediction behavior

This imbalance is further seen within a confusion matrix (Figure 2b) for the classifier and is also reflected within the individual class' F_1 scores. The confusion matrix demonstrates a high level of misclassifications occurring for quality values that were underrepresented; it supports the idea that the model is overfit and tends toward the more dominant labels that were available in this dataset. These findings highlight the limitations of the model, showing its inability to predict the entire range of classes.

5 Conclusion

This study sought to understand and demonstrate the effectiveness of gradient boosting, more specifically through the application of XGBoost on our Wine Quality dataset. By evaluating how gradient boosting compares to a basic decision tree model in both regression and classification task, the results show strong evidence for the superiority of gradient boosting in terms of generalization and robustness. Moreover, our findings also reinforce the idea that gradient boosting is extremely useful for modeling complex datasets through the integration of weak learners by sequentially minimizing the error. This is one of its most valuable aspects in flexibility due to how it builds the model incrementally applying gradient descent in function space and learning from previous errors at each iteration. However, within our applications we see challenges such as the class imbalance that caused our model to misclassify due to a lack of data points during training. Despite these limitations, gradient boosting remains a reliable and efficient choice for modeling complex data over more basic and weaker models.

References

[1] Analytics Vidhya. (2021). *Gradient Boosting Algorithm: A Complete Guide for Beginners*. Retrieved from https://www.analyticsvidhya.com/blog/2021/09/gradient-boosting-algorithm-a-complete-guide-for-beginners/h-what-is-a-gradient-boosting-algorithm

[2] Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). *The Elements of Statistical Learning* (2nd ed.). New York: Springer. Chapter 10: Boosting and Additive Trees, pp. 337–384. ISBN: 978-0-387-84857-0.

[3] IBM. (n.d.). *What is Gradient Descent?* IBM Think. Retrieved from https://www.ibm.com/think/topics/gradient-descent

[4] GeeksforGeeks. *ML — Gradient Boosting*. Retrieved from https://www.geeksforgeeks.org/ml-gradient-boosting/

[5] Friedman, J. H. (February 1999). *Evaluation of Predictive Models in Imbalanced Classification Tasks*. Retrieved from https://web.archive.org/web/20191101082737/http://statweb.stanford.edu/ jhf/ftp/trebst.pdf

[6] Yasser H. (2018). *Wine Quality Dataset*. Retrieved from https://www.kaggle.com/datasets/yasserh/wine-quality-dataset