Hyperparameter Optimization

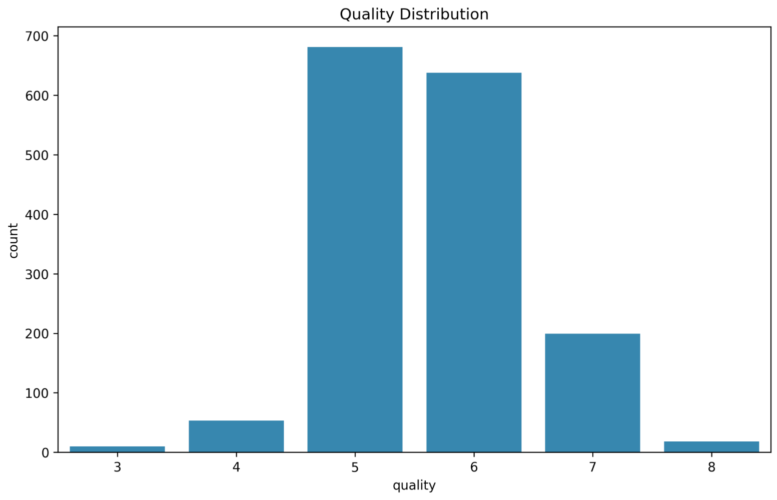
Farshad Ghorbanishovaneh

1. Introduction

In the realm of machine learning, hyperparameter optimization (HPO) is essential for fine-tuning models to achieve peak performance. This process involves selecting the optimal set of hyperparameters, which are the configurations external to the model that dictate its learning behavior. For instance, in predicting wine quality—a task where subtle nuances in data can dramatically influence prediction accuracy—the right choice of hyperparameters such as learning rate, number of trees in a random forest, or depth of the trees can significantly enhance model performance. The challenge lies not only in navigating the vast space of possible hyperparameter combinations but also in efficiently searching this space to avoid excessive computational costs while ensuring robust and generalizable results. To address this Bayesian optimization is employed, aiming to strike a balance between exploration of the hyperparameter space and exploitation of the most promising regions.

2. Dataset Description

The Wine Quality Dataset consists of 1,599 samples of wine, each characterized by 12 physicochemical properties and quality ratings. The properties include various acidity measures, sugar, sulfur dioxide levels, and alcohol content. All columns in the dataset are fully populated with non-null values. The quality ratings vary from 3 to 8, with the majority of the wines rated between 5 and 6 (Figure 1). Here is a detailed breakdown of the dataset



structure and a correlation heatmap to visualize the relationships between different properties:

Attribute	Description	Data Type
Fixed Acidity	Most acids involved with wine	float64
Volatile Acidity	Amount of acetic acid in wine	float64
Citric Acid	Found in small quantities, citric acid	float64
Residual Sugar	Amount of sugar remaining after fermentation	float64
Chlorides	Amount of salt in the wine	float64
Free Sulfur Dioxide	Free form of SO2; prevents microbial growth	float64
Total Sulfur Dioxide	Amount of free and bound forms of S02	float64
Density	Density of the wine	float64
pH	Describes the acidity or basicity of wine	float64
Sulphates	Wine additive which can contribute to sulfur dioxide gas (S02) levels	float64
Alcohol	Alcohol content of the wine	float64
Quality	Score between 3 and 8 (inclusive)	int64

Table 1. Dataset Description

Besides the 'quality' column, which serves as our target variable, all other attributes in our dataset are of the float data type. The distribution of these data points is depicted in the boxplot Figure 2. In Figure 3, you can observe the correlations between each variable within the dataset, reflecting how each physicochemical property relates to the others and to the overall quality of the wine. The strongest positive correlation with quality is exhibited by alcohol content, suggesting that higher alcohol levels often correspond to higher quality ratings. Conversely, volatile acidity has the strongest negative correlation with quality, indicating that lower volatile acidity is associated with higher quality wines. Other properties, such as sulphates and citric acid, display weaker positive correlations with quality, suggesting a modest relationship to higher quality. In contrast, density and total sulfur dioxide show weaker negative correlations, implying that lower values in these properties might be characteristic of higher quality wines.

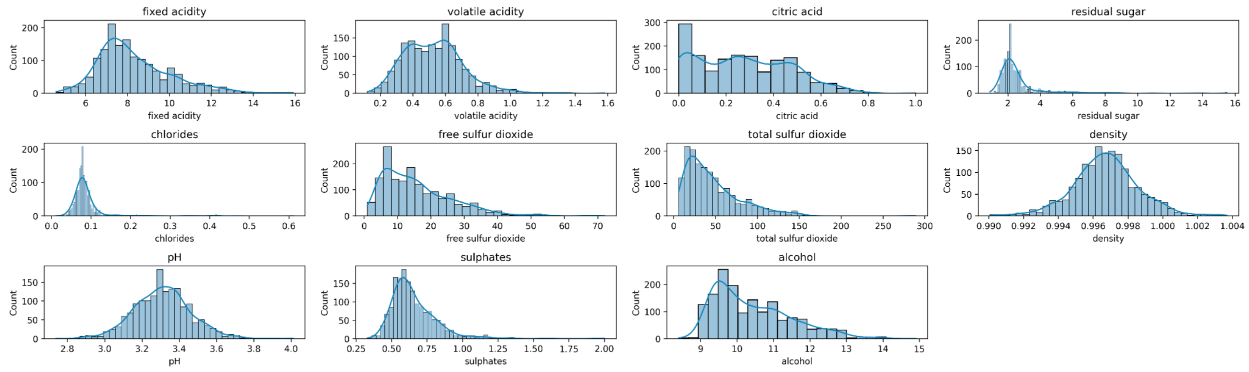


Figure 2. The image shows a boxplot for each physicochemical property and quality rating in the Wine Quality Dataset.

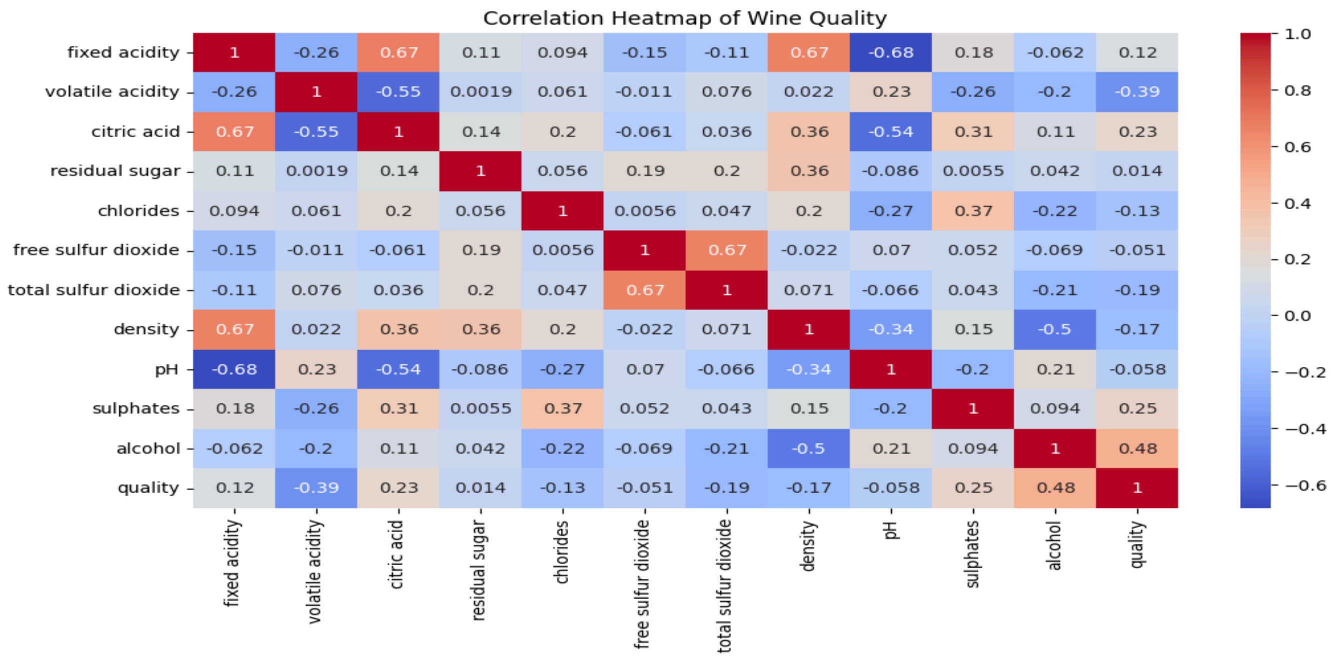


Figure 3. Correlation heatmap of wine quality dataset

3. Experimental Setup

In this study, I utilize Python as the primary programming language, leveraging its extensive support for data manipulation and machine learning operations. Key libraries employed include Pandas for handling and preprocessing data, NumPy for numerical operations, Matplotlib and Seaborn for generating insightful data visualizations, and Scikit-learn for implementing various machine learning algorithms. Additionally, the BayesianOptimization library from the `bayes_opt` package in optimizing the hyperparameters of the models.

Initial data processing involves checking for null values, which are absent in this dataset, followed by scaling the features using Scikit-learn's StandardScaler to normalize the data, ensuring that no single attribute unduly influences the model's performance.

I apply several machine learning models including Support Vector Machines (SVM), Gradient Boosting Machines (GBM), Random Forests, and Decision Trees. Each model's performance is optimized through Bayesian optimization, which seeks the best hyperparameters by maximizing a given objective function, in this case, the cross-validated accuracy score.

For the SVM model, I optimize the regularization parameter 'C' and the kernel coefficient 'gamma'. The GBM model's hyperparameters, such as the number of estimators, the learning rate, and the maximum depth of the trees, are also fine-tuned. Similarly, the Random Forest and Decision Tree models undergo optimization for parameters including the number of estimators, maximum depth, minimum samples split, and minimum samples leaf.

Each model's performance is rigorously evaluated using a 5-fold cross-validation scheme within the training dataset, ensuring that our findings are robust and generalizable. I use accuracy as the primary metric for evaluation, given its intuitiveness and common application in classification tasks.

This experimental setup, encompassing data preprocessing, model selection, hyperparameter optimization, and performance evaluation. For each model, the Bayesian Optimization process was executed 105 times, consisting of 100 exploitation iterations based on surrogate modeling, and 5 exploration iterations to discover new points or escape from local maxima.

4. Results

In the context of hyper parameter optimization to enhance predictive accuracy, Bayesian optimization serves as a strategic approach to systematically search for the optimal hyperparameters. This method uses prior knowledge of the hyperparameter performance and updates this with sequential experiments to minimize computational expense while exploring the parameter space effectively. The following sections delve into the performance results from Bayesian optimization for several machine learning models, specifically SVM, Gradient Boosting, Random Forest, and Decision Tree. These insights demonstrate how different configurations impact model accuracy and provide guidance on fine-tuning these parameters for improved predictive power.

SVM Optimization

The results show several iterations where the accuracy fluctuated around the mid-50% range, peaking at an accuracy of 58.88% with parameters C=60.54 and gamma=0.001815. This suggests that the SVM is quite sensitive to the choice of C and gamma, where small adjustments can significantly affect the performance, especially with gamma which affects the decision boundary in high-dimensional space.

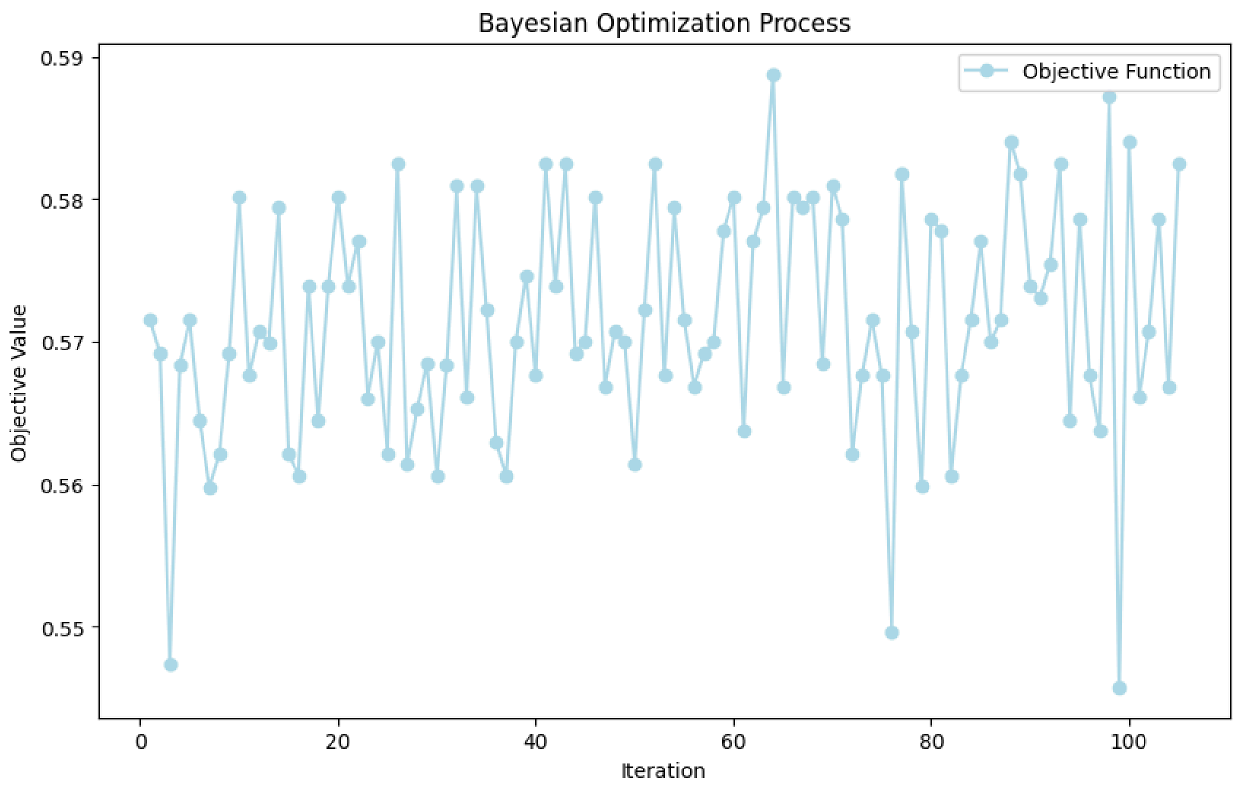


Figure 4. Bayesian Optimization Process for SVM

Gradient Boosting Optimization

For Gradient Boosting, the top accuracy reached was 68.81%, using a learning rate of 0.05663, max depth of 5.793, and 126 estimators. This indicates a more robust performance across various parameter configurations, showing less sensitivity to hyperparameters compared to SVM, except for noticeable improvements at certain values of learning rate and number of estimators.

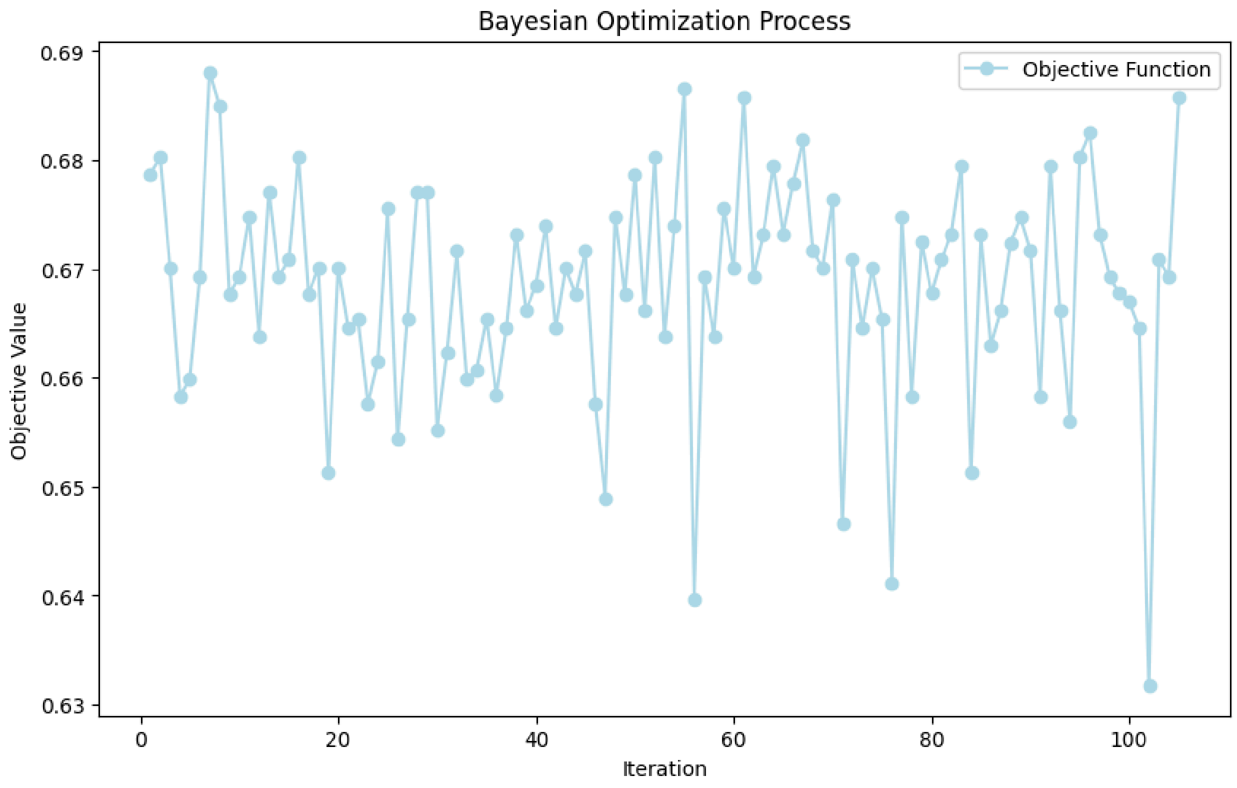


Figure 5. Bayesian Optimization Process for Gradient Boosting

Random Forest Optimization

Random Forest reached its best accuracy at 68.10% with parameters like max depth of 9.23, min samples leaf of 2.03, min samples split of 2.19, and 119 estimators. The model's performance reflects a good balance across different parameters, indicating its resilience to overfitting with increasing depth and complexity due to its ensemble nature

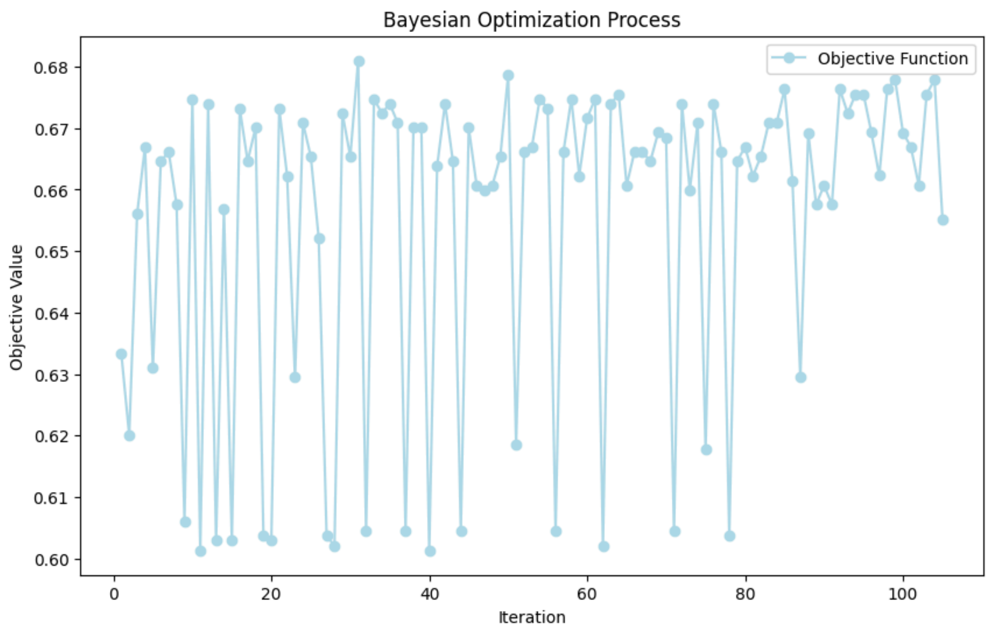


Figure 6. Bayesian Optimization Process for Random Forest

Decision Tree Optimization

The Decision Tree showed the highest accuracy of 60.44%, achieved at a max depth of 6.345, min samples split of 19.34, and min samples leaf of 4.30. This model seems to perform best at a moderate depth, preventing overfitting while allowing enough complexity to capture essential patterns in the data.

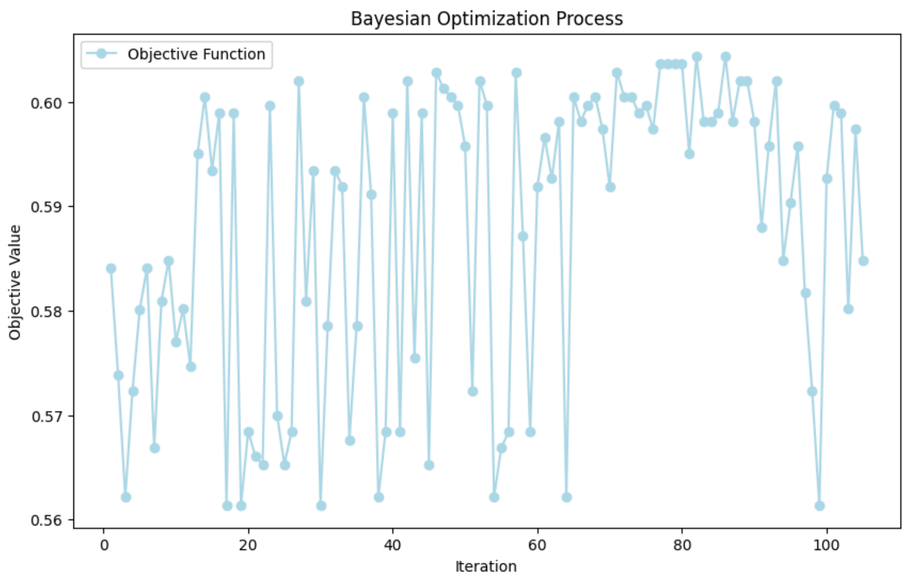


Figure 7. Bayesian Optimization Process for Decision Tree