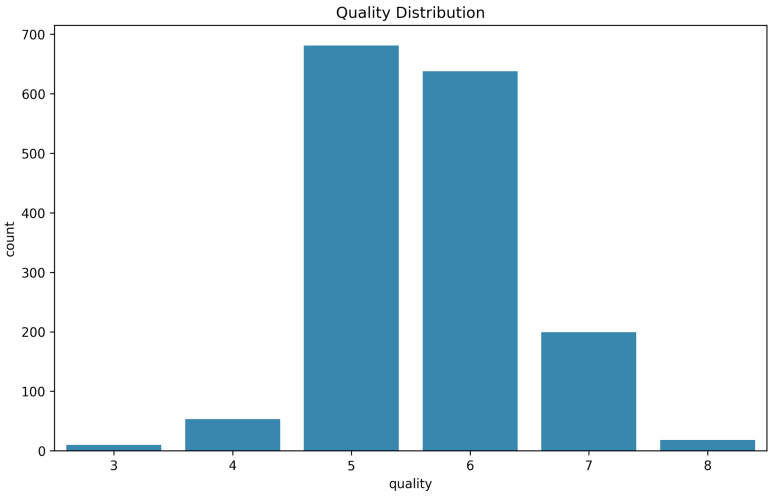
Hyperparameter Optimization

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 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.

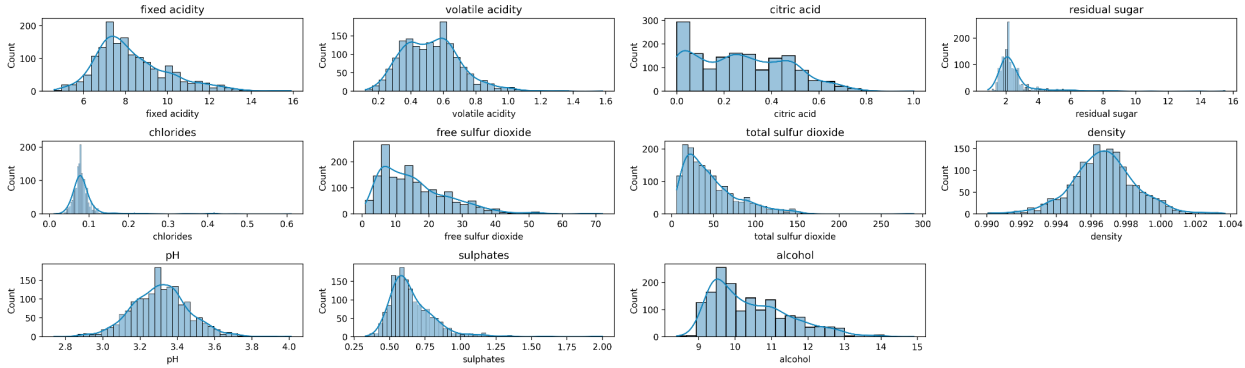


Figure 2. The image shows a boxplot for each physicochemical property and quality rating in the Wine Quality Dataset.

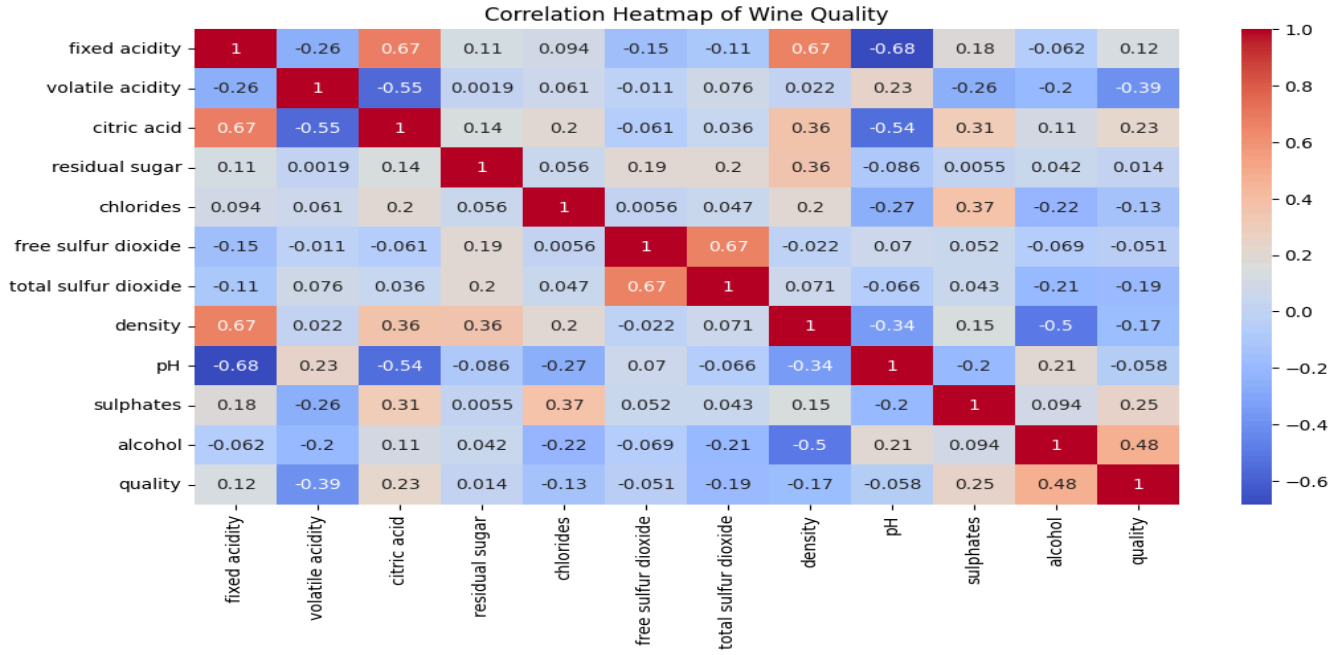


Figure 3. Correlation heatmap of wine quality dataset

3. Experimental Setup

In this study, I utilize Python as the primary programming language. 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 finding optimized value with BayesianOptimization the Support Vector Machine (SVM), two key parameters were adjusted: C and γ. C, ranging from 1 to 100 γ, set between 0.001 and 0.1, affects the curvature of the decision boundary, with lower values implying a smoother boundary. For Gradient Boosting, we tuned three parameters: n_estimatorsn_estimators (100 to 400) determining the number of sequential trees to be modeled, max_depthmax_depth (3 to 10) limiting the complexity of each tree, and learning_ratelearning_rate (0.01 to 0.3) which scales down the contribution of each tree to prevent overfitting. Random Forest hyperparameters included n_estimatorsn_estimators for the number of trees in the forest, max_depthmax_depth to control the depth of each tree, min_samples_splitmin_samples_split (2 to 10) and min_samples_leafmin_samples_leaf (1 to 4) to define the minimum number of samples required at each node to split and at a leaf node, respectively, balancing detail against the risk of overfitting. Lastly, the Decision Tree's hyperparameters such as max_depthmax_depth (3 to 20), min_samples_splitmin_samples_split (3 to 20), and min_samples_leafmin_samples_leaf

(3 to 8) were optimized to refine the tree structure, impacting both the depth and the minimum criteria for making splits and creating leaf nodes.

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.

In the Random Forest model, I implemented nested resampling by dividing the data into five outer splits using cross-validation. For each of these outer splits, I conducted Bayesian optimization inside an inner cross-validation loop to tune the hyperparameters. The best hyperparameters identified from this inner loop were then used to train the Random Forest model on the entire training set of the corresponding outer fold. Finally, the trained model was evaluated on the test set of the outer fold, ensuring that each assessment of the model's performance was based on independently optimized parameters.

This experimental setup includes 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—which the n_iter parameter represents in the package—and 5 exploration iterations, which the init_points parameter represents as random points for discovering new possibilities. Initially, more weight is given to exploration to avoid local optima and to understand the broader landscape of the parameter space. As the process progresses and more data is

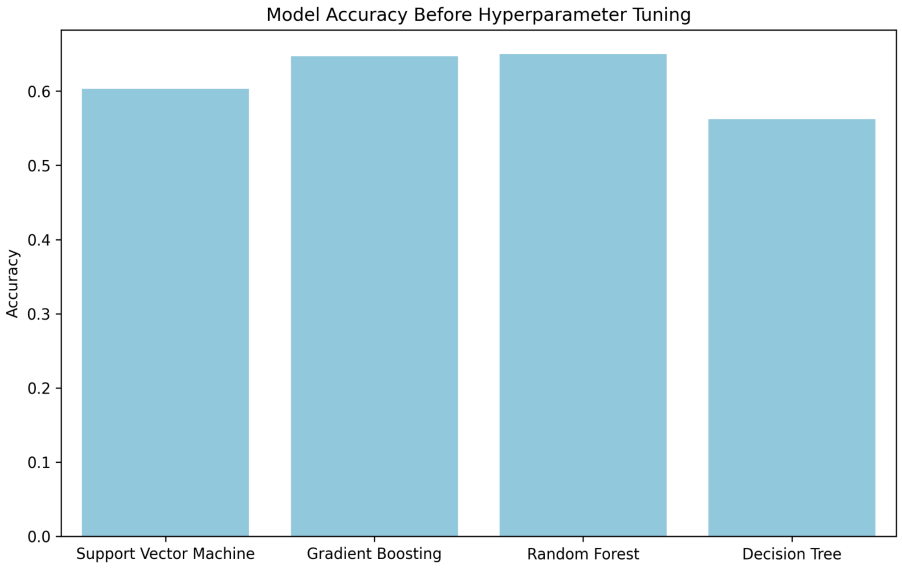


Figure 4. Model Accuracy Before Hyperparameter Tuning

accumulated, the focus shifts towards exploitation, refining the search around the areas that have shown the most promise. This approach ensures that the model doesn't just stick to exploiting known solutions, which could lead to getting stuck in a local optimum. While this exploration doesn't guarantee immediate success, it allows the model to search more broadly in the parameter space, increasing the chances of finding a global optimum.

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, 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. Before hyperparameter tuning, we observed the following accuracy results across different models: the Support Vector Machine achieved an accuracy of 0.603125, the Gradient Boosting model recorded an accuracy of 0.646875, the Random Forest model came in slightly higher at 0.65, and the Decision Tree model had the lowest accuracy at 0.5625.

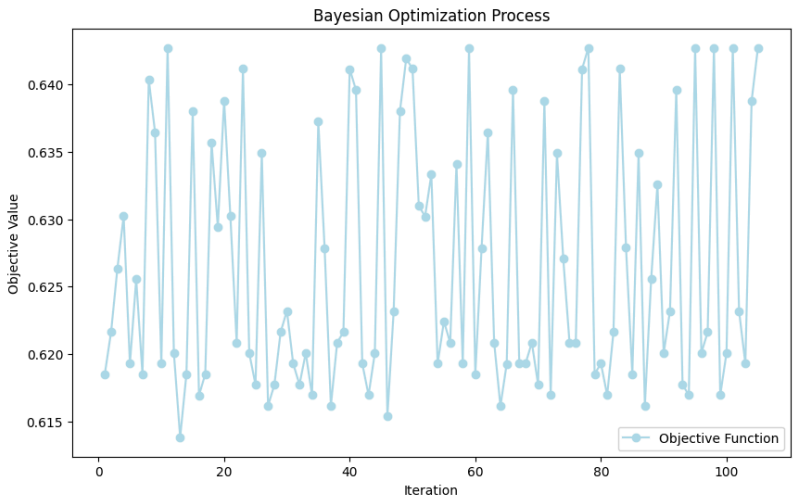


Figure 5. Bayesian Optimization Process for SVM

SVM Optimization

The results show several iterations where the accuracy fluctuated around the mid-50% range, peaking at an accuracy of 64.27% with parameters C=10.179 and gamma=scale. This performance was higher than what was achieved using the default parameter values, which yielded 60.3125%, representing an increase of approximately 3.957 percentage points or about 6.56%. 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. The plot below shows that continuing to use Bayesian optimization for more iterations can still yield better hyperparameters. We aim to show that with enough time and resources, we could find better hyperparameters. As the plot below shows, performance improved after about 1000 iterations, and this process can continue. We used SVM as an example.

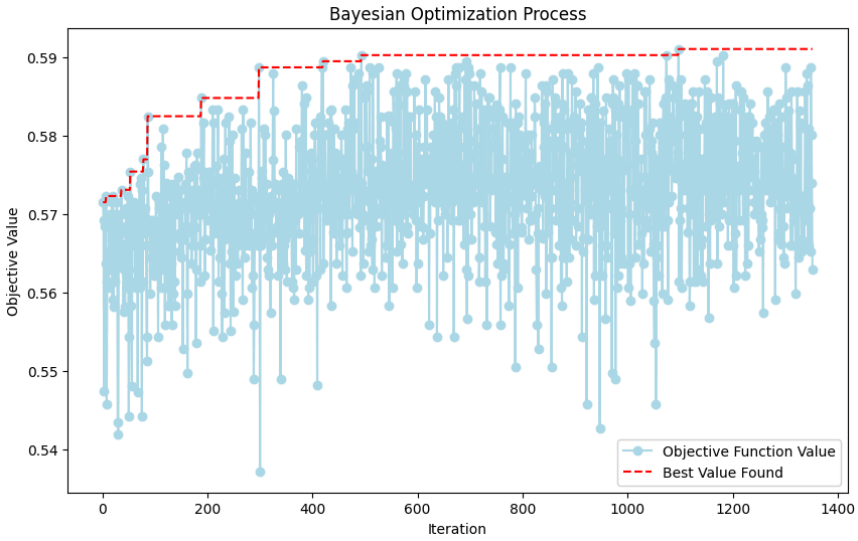


Figure 10. After running Bayesian Optimization for 1400 iterations to tune the SVM hyperparameters.

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 performance was higher than what was achieved using the default parameter values, which yielded 64.6875%. The improvement was

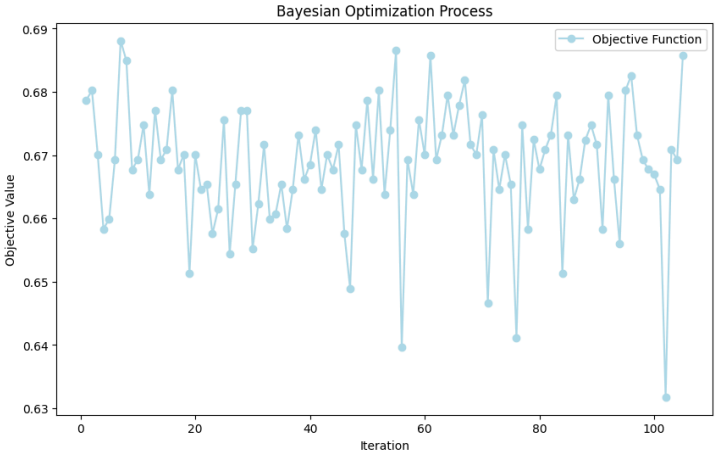


Figure 6. Bayesian Optimization Process for Gradient Boosting

approximately 4.12 percentage points, an increase of about 6.37%.

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 . ThThis performance was higher than what was achieved using the default parameter values, which yielded 65.00%. The improvement was approximately 3.10 percentage points, an increase of about 4.77%. The model's performance reflects a good balance across different parameters, indicating its resilience to overfitting with

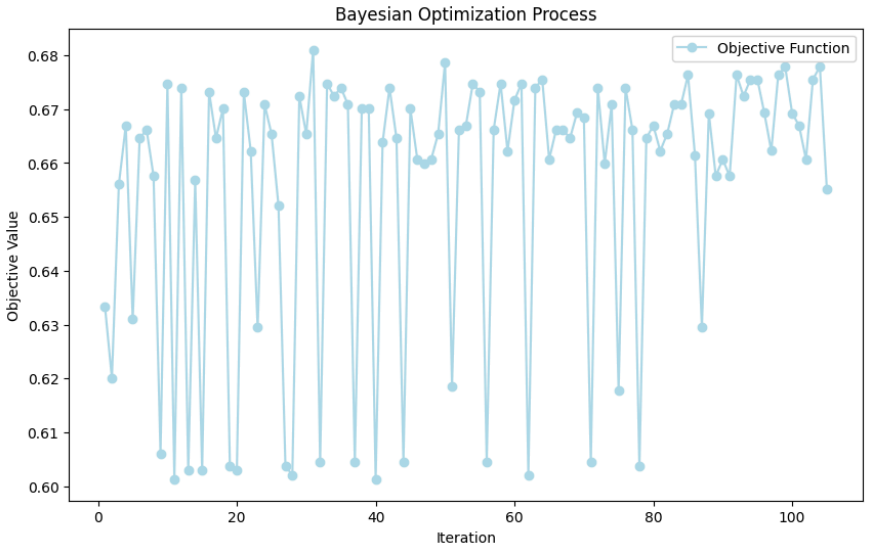


Figure 7. Bayesian Optimization Process for Random Forest

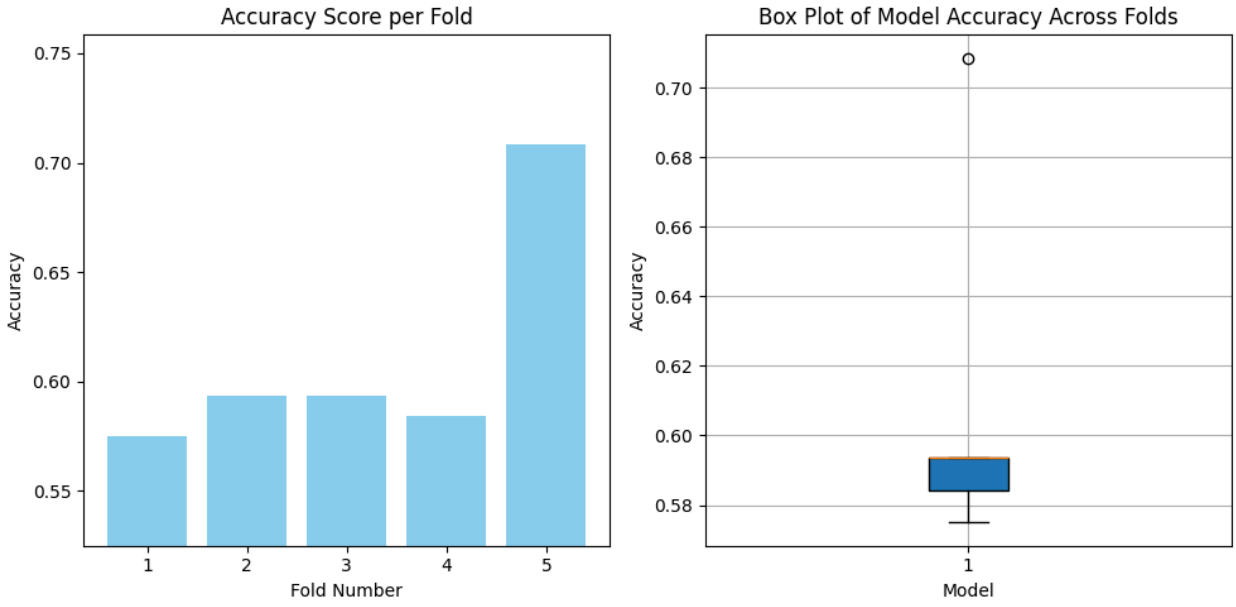


Figure 8. Random Forest Accuracy per Fold

increasing depth and complexity due to its ensemble nature.

In the nested resampling process for the Random Forest model with 5 folds, we obtained Outer CV scores of 0.58, 0.59, 0.59, 0.58, 0.71, resulting in a mean accuracy of 0.61. The best hyperparameters identified through Bayesian optimization varied across the folds: for the first fold, the settings were 6.80, 2.75, 7.09, and 268; for the second fold, 5.62, 1.75, 5.29, and 355; for the third fold, 4.77, 2.90, 6.15, and 188; for the fourth fold, 5.19, 1.29, 7.43, and 299; and for the fifth fold, 5.80, 1.95, 4.07, and 360. These settings—reflecting max depth, min samples leaf, min samples split, and number of estimators respectively. I used Random Forest as a case that serves as an example of how nested resampling can significantly affect the interpretation of model performance and parameter tuning.

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 performance was higher than what was achieved using the default parameter values, which yielded 56.25%. The improvement was approximately 4.19 percentage points, an increase of about 7.45%. This model seems to perform best at a moderate depth, preventing overfitting while allowing enough complexity to capture essential patterns in the data.

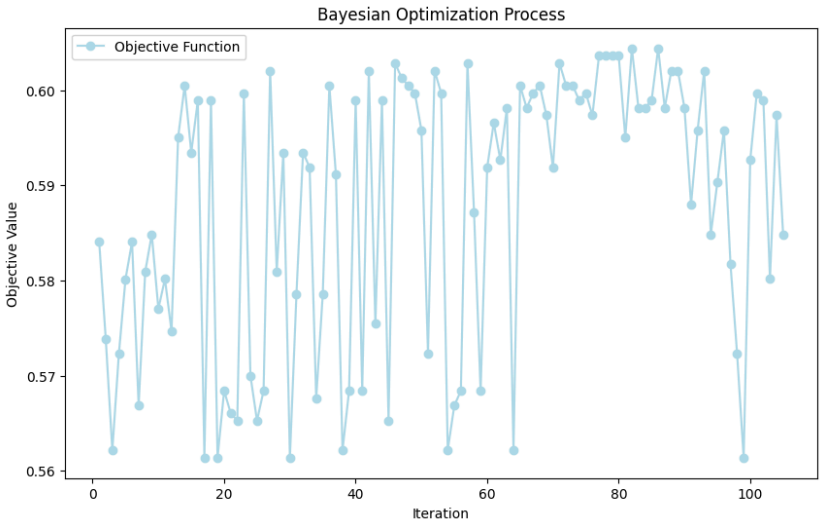


Figure 9. Bayesian Optimization Process for Decision Tree