Hyperparameter Optimization Report

Soudabeh Bolouri

Introduction:

In this exercise, we use some machine learning models including Random Forest, SVM, and AdaBoost (Adaptive Boosting) and their associated hyperparameters to find the optimal configuration for the dataset. Nested cross-validation was employed to ensure a robust evaluation of each model's performance. Using Bayesian optimization, a sophisticated method for tuning hyperparameters, we aim to identify the most effective machine learning algorithms and hyperparameter settings.

Dataset Description:

The dataset that we utilized in this exercise is the "Wine Quality" dataset, precisely the "white" wine version. It contains data on different features of white wines, including fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, PH, sulphates, and alcohol. The dataset includes a totality of 4,898 rows and 12 features. The "quality" of the wine is the target variable, which we desire to predict. Also, we did not find any missing values in the dataset.

Experimental Setup:

We set up the experiment as follows using the Python programming language:

We split the dataset using the "split_data" function into training and testing sets with an 80-20 split ratio and employed a fixed random seed (random_state=42) for reproducibility. Then, we separated the features and the target variable (quality) for both the training and testing sets using the "separate_features_target" function. This ensures machine learning algorithms can be trained on the features to predict the target variable.

The hyperopt_fn function is defined to optimize hyperparameters for Random Forest, Support Vector Machine, and AdaBoost. Bayesian optimization was used for hyperparameter tuning within an inner 3-fold cross-validation loop. Subsequently, a 5-fold outer cross-validation was applied to assess each model's generalizability. Parameter tuning is conducted 15 times.

The hyperparameters for a **Random Forest** Classifier are defined using [1] as follows:

- **n_estimators**: the number of trees in the forest. Here we use a range of (10, 250) - **default:** Usually set to 100.
- max_depth: the maximum depth of each tree in the forest. Here we use a range of (1, 50)- **default:**None, which means nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- **min_samples_split**: the minimum number of samples required to split an internal node. Here we use a range of (2, 25) - **default:** 2.
- **max_features**: the maximum number of features that are considered for splitting at each node in a decision tree within the forest. Here we use a range of (0.1, 0.999)- **default:** The default is usually the square root of the number of features for classification tasks (sqrt(n_features)).
- The **random_state** parameter is set to 42 to ensure reproducibility- **default:** None.

SVM takes two hyperparameters as input: **C** and **gamma**. These hyperparameters are essential for configuring the SVM model and are defined using [2] as follows:

- **C** is the regularization parameter, which controls the trade-off between achieving a low training error and a low testingerror. Higher C values lead to smaller margins and potentially higher training accuracy. Here we consider $0.1 < c < 100$. **Default Value:** Usually, the default value of C in SVM implementations (like in Scikit-learn's SVC) is 1.0.
- **gamma** is a parameter that defines the influence of individual data points in the dataset. Higher gamma values can lead to more complex decision boundaries. Here we consider $0.0001 < \text{gamma} < 10$. **Default Value:** In many SVM implementations, the default value of gamma is either 1/n_features (where n_features is the number of features in the dataset) or a value like 'scale' or 'auto' which are based on the features.

AdaBoost classifier takes two hyperparameters as input and they are defined as follows:

- **n_estimators**: This parameter specifies the maximum number of weak learners to be used during the boosting process. Here we use a range of (50, 100) - **default Value**: The default value for n_estimators in Scikit-learn's AdaBoostClassifier is 50.

- **learning_rate**: The learning rate is a weight applied to each classifier at each boosting iteration. A higher learning rate increases the contribution of each classifier. Reducing the learning rate means the model will need more estimators to fit complex data but can generalize better. Here, we use a range of (0.01, 2) - **default Value**: The default value for the learning_rate in Scikit-learn's AdaBoostClassifier is 1.0.

Results:

The models were evaluated based on accuracy, and we can see the results in bellow table.

Model	Best Parameters	Default Nested CV Score	Optimized Nested CV Score
Random Forest	• Max Depth: 18 • Max features: **0.24** • Min Samples Split: **4** • Number of Estimators: **439**	**0.6839**	**0.6848**
SVM	• C: **73.23** • Gamma: **5.99**	**0.4490**	**0.6203**
AdaBoost	• Learning Rate: **0.05** • Number of Estimators: **119**	**0.4259**	**0.4524**

We compared the performance of three different classifiers: Random Forest, Support Vector Machine (SVM), and AdaBoost. For each model, we performed Nested Cross-Validation (Nested CV) using both default and optimized hyperparameters to assess their predictive accuracy.

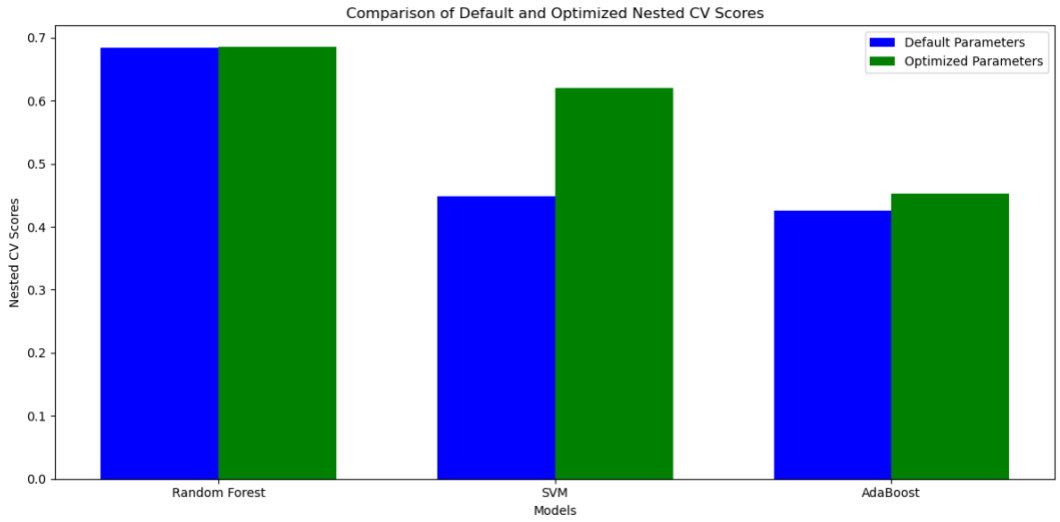
The Random Forest model showed strong performance even with its default settings, achieving a nested CV score of approximately 0.684. After applying Bayesian Optimization, we observed a marginal improvement in the nested CV score, which increased to approximately 0.685.

The SVM model, with its default hyperparameters, yielded a lower nested CV score of about 0.449. However, after optimization, the nested CV score improved significantly to approximately 0.620. The improvement suggests that the SVM model was sensitive to hyperparameter tuning.

The AdaBoost classifier initially presented a nested CV score of around 0.426 with default hyperparameters. Upon optimization, a slight increase in performance was observed, with the score rising to approximately 0.452.

With both default and optimized parameters, Random Forest outperformed the other models. Hyperparameter optimization greatly improved SVM, but only moderately improved AdaBoost.

The bellow bar chart illustrates the nested CV scores for the three classifiers, comparing their performance with default and optimized parameters. The Random Forest model consistently scored above 0.68, demonstrating the highest accuracy among the models tested. The SVM's performance was notably enhanced by optimization, while AdaBoost showed limited gains.



References:

[1] Step-by-step Guide: Bayesian Optimization with Random Forest
https://drlee.io/step-by-step-guide-bayesian-optimization-with-random-forest-fdc6f329db9c

[2] Hyperparameter Tuning for Support Vector Machines – C and Gamma Parameters
https://towardsdatascience.com/hyperparameter-tuning-for-support-vector-machines-c-and-gamma-parameters-6a5097416167

[3] Hyperparameter Optimization in Gradient Boosting Packages with Bayesian Optimization
https://towardsdatascience.com/hyperparameter-optimization-in-gradient-boosting-packages-with-bayesian-optimization-aaf1b27e7b90

[4] https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter

[5] https://matplotlib.org/stable/gallery/lines_bars_and_markers/barchart.html#sphx-glr-gallery-lines-bars-and-markers-barchart-py