Hyperparameter Optimization Report

Soudabeh Bolouri

Introduction:

In this exercise, we use some machine learning models including Random Forest, SVM, and Gradient 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.
Three functions **(hyperopt_rf, hyperopt_svm**, and **hyperopt_gb)** are defined to optimize hyperparameters for Random Forest, Support Vector Machine, and Gradient Boosting, respectively. 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. The 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 $0.0001 <$ 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.

Gradient Boosting classifier takes three hyperparameters as input and they are defined using [3] as follows:

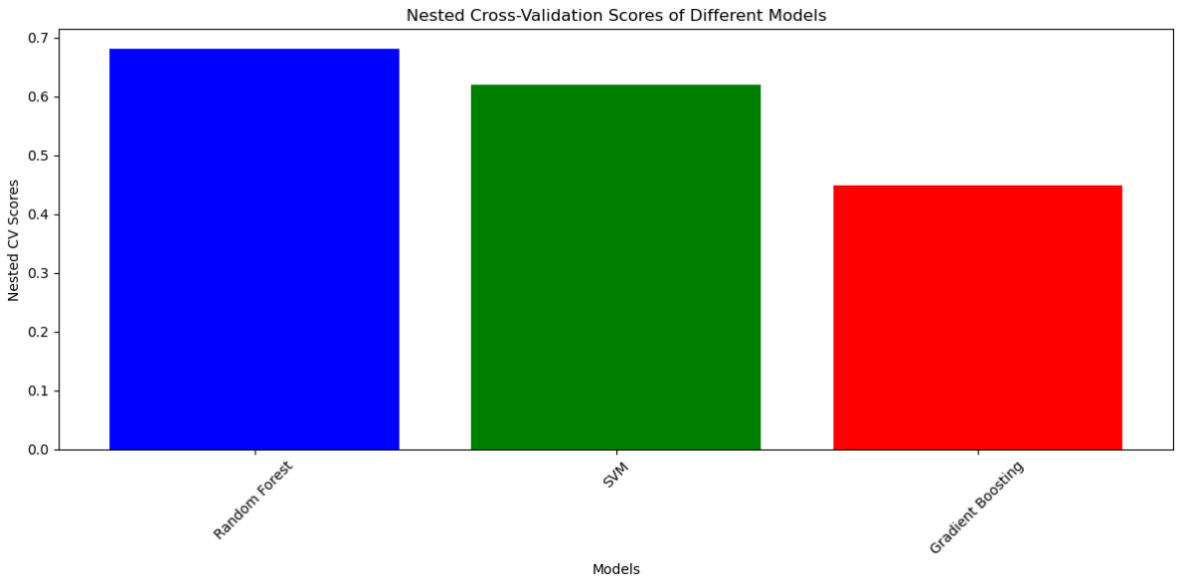
- **n_estimators**: Number of weak predictors (decision trees) in the ensemble. This parameter controls the number of iterations the boosting algorithm goes through. More trees generally lead to a more powerful model, but there's a risk of overfitting. Here we used a range of (10, 100) - **Default Value:** Typically, 100 in Scikit-learn**.**
- **learning_rate**: Reducing the step size to correct errors from previous iterations. Smaller values make the optimization process more robust, but they require more iterations. Here we used a range of (0.01, 0.3)- **default Value:** Usually 0.1 in scikit-learn**.**
- **max_depth**: The maximum depth of the individual decision trees in the ensemble. Controlling the depth helps prevent overfitting. Here we used a range of (5, 40)- **default Value:** Generally, 3 in Scikit-learn.

Results:

The models were evaluated based on accuracy, which is appropriate for this balanced classification problem.

Model	Best Parameters	Nested CV Score
Random Forest	• Max Depth: **35** • Max features: **0.1** • Min Samples Split: **4** • Number of Estimators: **64**	**0.6803**
SVM	• C: **73.23** • Gamma: **5.99**	**0.6203**
Gradient Boosting	• Learning Rate: **0** • Max depth: **38** • Number of Estimators: **75**	**0.4488**

In the chart below, we compare the results of three different models and can see that the Random Forest model has the highest accuracy.



In this experiment, the Random Forest model demonstrated the highest generalization capability on the Wine Quality dataset. Its ability to handle complex datasets effectively makes it the superior choice among the three models. However, SVM's performance indicates it could be a viable alternative, especially with further tuning and potential feature engineering. The Gradient Boosting model's results suggest the need for a more careful examination of the hyperparameter space or the consideration of alternative ensemble methods.

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