Hyperparameter Optimization Report

Soudabeh Bolouri

Introduction:

In this exercise, we use various machine learning algorithms and their associated hyperparameters to find the optimal configuration for our dataset. 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:

1. **Data Preprocessing:** 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. In order to design machine learning models or conduct experiments, consistency and reproducibility are crucial. The result should be the same if we rerun the same code with the same data, ensuring that we don't have any random factors influencing our results. 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.

2. **Hyperparameter Optimization Functions**: Three functions **(**hyperopt_rf**,** hyperopt_svm, and hyperopt_gb**)** are defined to optimize hyperparameters for Random Forest, Support Vector Machine, and Gradient Boosting, respectively. Using 5-fold cross-validation, each function creates a machine learning model with different hyperparameters as input and evaluates its performance. The mean accuracy is calculated and returned.

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

- **n_estimators**: the number of trees in the forest. Here we used a range of (10, 250).
- **max_depth**: the maximum depth of each tree in the forest. Here we used a range of (1, 50).
- **min_samples_split**: the minimum number of samples required to split an internal node. Here we used a range of (2, 25).
- **max_features**: the maximum number of features that are considered for splitting at each node in a decision tree within the forest. Here we used a range of (0.1, 0.999).
- The **random_state** parameter is set to 42 to ensure reproducibility.

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 lowtesting error. Higher **C** values lead to smaller margins and potentially higher training accuracy. Here we consider $0.1 < c < 100$.

- **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 $0.0001 < \text{gamma} < 10$.

Gradient Boosting classifier takes three hyperparameters as input:

n_estimators, **learning_rate**, and **max_depth**. These hyperparameters are crucial for configuring the Gradient Boosting model and 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).

- **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).

- **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).

3. **Bayesian Optimization**:

In order to maximize the mean accuracy of each machine learning algorithm, Bayesian optimization is used to search for the best hyperparameters. During optimization, the best hyperparameters and their scores are extracted from each algorithm. These are the hyperparameter settings that maximize model performance.

4. **Model Training**:

   Based on the best hyperparameters found for each algorithm, models are created and trained on the entire training dataset.

5. **Model Evaluation**:

   In addition to evaluating the trained models on test datasets, the test accuracy of each algorithm is calculated. This measures how well the algorithm performs on unseen data.

Results:

After running the program, we can see the best hyperparameters found, the highest accuracy score achieved during optimization (best score) and the test accuracy score, indicating the performance of the model on the test dataset (test score).

The best hyperparameters, the best score and test score for Random Forest are as follows:

```
|   iter    |  target   | max_depth | max_fe... | min_sa... | n_esti... |
-------------------------------------------------------------------------
| 1         | 0.5666    | 6.957     | 0.5508    | 13.18     | 142.6     |
| 2         | 0.6429    | 47.95     | 0.795     | 10.08     | 189.9     |
| 3         | 0.6302    | 28.68     | 0.2649    | 15.72     | 212.9     |
| 4         | 0.6258    | 15.16     | 0.3603    | 14.19     | 87.68     |
| 5         | 0.6189    | 13.92     | 0.7733    | 17.16     | 224.2     |
| 6         | 0.6452    | 49.25     | 0.8574    | 10.34     | 189.4     |
| 7         | 0.6455    | 50.0      | 0.999     | 2.0       | 238.9     |
| 8         | 0.6187    | 22.94     | 0.1729    | 17.45     | 26.55     |
| 9         | 0.6105    | 50.0      | 0.1       | 25.0      | 250.0     |
| 10        | 0.609     | 50.0      | 0.999     | 25.0      | 173.8     |
| 11        | 0.6457    | 48.78     | 0.999     | 2.0       | 223.2     |
| 12        | 0.6136    | 49.36     | 0.3215    | 24.8      | 210.7     |
| 13        | 0.646     | 34.91     | 0.999     | 2.0       | 232.4     |
| 14        | 0.6557    | 48.96     | 0.2977    | 2.145     | 70.36     |
| 15        | 0.6118    | 50.0      | 0.999     | 22.11     | 71.34     |
=========================================================================
Best Random Forest hyperparameters: {'max_depth': 48.96278847174345, 'max_features': 0.29774473792622497,
 'min_samples_split': 2.1454352977300792, 'n_estimators': 70.35867056328624}
Best Random Forest score: 0.6556884822894675
Random Forest Test Score: 0.6928571428571428
```

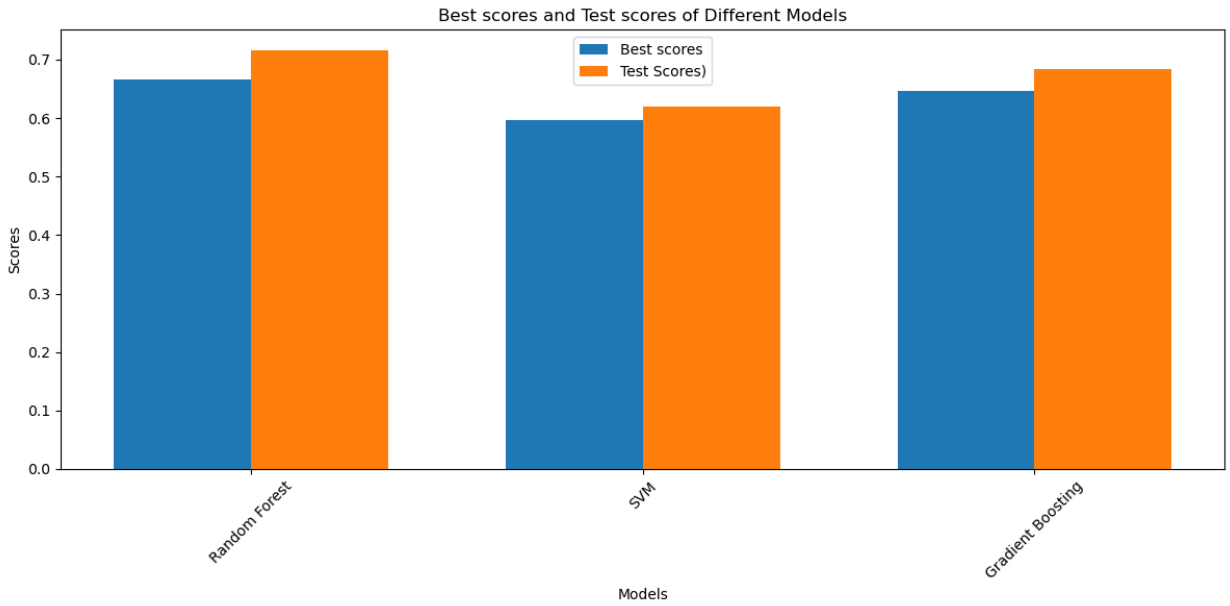
The best hyperparameters, the best score and test score for SVM are as follows:

```
/main.py
|   iter    |  target   |     C     |   gamma   |
-------------------------------------------------
| 1         | 0.596     | 85.13     | 3.252     |
| 2         | 0.596     | 10.23     | 8.647     |
| 3         | 0.5962    | 18.59     | 4.795     |
| 4         | 0.596     | 91.58     | 3.145     |
| 5         | 0.596     | 90.31     | 7.618     |
| 6         | 0.596     | 22.75     | 3.23      |
| 7         | 0.596     | 2.405     | 9.61      |
| 8         | 0.5962    | 18.56     | 4.787     |
| 9         | 0.596     | 18.14     | 7.406     |
| 10        | 0.596     | 18.56     | 3.463     |
| 11        | 0.5962    | 18.23     | 5.358     |
| 12        | 0.5962    | 19.56     | 5.702     |
| 13        | 0.5965    | 21.14     | 5.996     |
| 14        | 0.5965    | 22.01     | 6.381     |
| 15        | 0.596     | 21.48     | 7.473     |
=================================================
Best SVM hyperparameters: {'C': 21.135021615817923, 'gamma': 5.995951077208287}
Best SVM score: 0.5964764641488779
SVM Test Score: 0.6224489795918368
```

And, finally, the best hyperparameters, the best score and test score for Gradient Boosting are as follows:

```
|   iter    |  target   | learni... | max_depth | n_esti... |
-------------------------------------------------------------
| 1         | 0.5896    | 0.1782    | 27.23     | 51.2      |
| 2         | 0.623     | 0.2295    | 17.78     | 98.7      |
| 3         | 0.6085    | 0.02793   | 22.27     | 46.71     |
| 4         | 0.6       | 0.1521    | 39.44     | 66.88     |
| 5         | 0.6115    | 0.08054   | 16.32     | 13.67     |
| 6         | 0.6355    | 0.07601   | 10.02     | 41.26     |
| 7         | 0.645     | 0.2533    | 9.762     | 40.35     |
| 8         | 0.5827    | 0.01      | 10.63     | 38.05     |
| 9         | 0.6424    | 0.0935    | 8.711     | 40.9      |
| 10        | 0.6442    | 0.1599    | 7.668     | 39.68     |
| 11        | 0.6205    | 0.1533    | 5.875     | 40.6      |
| 12        | 0.6439    | 0.2653    | 7.109     | 37.67     |
| 13        | 0.6136    | 0.1147    | 5.479     | 36.8      |
| 14        | 0.6381    | 0.06258   | 8.282     | 38.56     |
| 15        | 0.6427    | 0.1648    | 8.477     | 44.43     |
=============================================================
Best Gradient Boosting hyperparameters: {'learning_rate': 0.25328530787266806, 'max_depth': 9.761532712537566, 'n_estimators': 40.34662296378681}
Best Gradient Boosting score: 0.6449719159694529
Gradient Boosting Test Score: 0.6928571428571428
```

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



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