COSC 5557
Hyperparameter Optimization

Almountassir Bellah Aljazwe

March 2024

1 Introduction

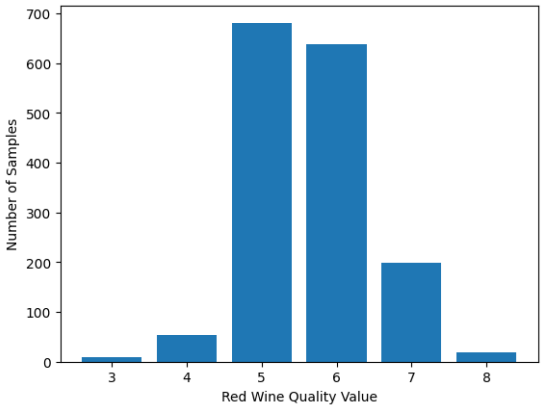
The main objective of finding an optimal predictive model for our "Red Wine Quality" dataset remains in progress. For context, the performance accuracy of five different predictive models were evaluated and compared, in our previous report. What was done, however, is not sufficient in finding an optimal predictive model; this is due to the fact that no hyper-parameter optimization was performed during the previous step. With this in mind, optimizing the hyper-parameters is the necessary upcoming step in accomplishing our main objective.

2 Dataset Description

The 'Red Wine Quality' dataset contains 1599 rows (samples) and twelve columns; eleven are feature columns, while the final twelfth is the target column. The target column contains categorical values; each value is an integer which corresponds to a 'quality' ranking from '1' to '10'. The dataset, however, ranges from '3' to '8' only, with all six values appearing. The dataset contains no missing values for all the provided samples.

A defining characteristic of the 'Red Wine Quality' dataset is its imbalanced distribution of target values. This can be concluded from the following :

3	4	5	6	7	8
10/1559	53/1559	681/1559	638/1559	199/1559	18/1559
$\approx 0.6\%$	$\approx 3\%$	$\approx 44\%$	$\approx 41\%$	$\approx 13\%$	$\approx 1\%$



From the figures, we can clearly observe that the data distribution is cluttered around the middle target values. This, in reality, is reasonable as it may be rare to encounter wine with extremely low quality or extremely high quality; it is reasonable to expect wine with average quality. With regards to our model, the imbalanced nature of this dataset may present obstacles in the performance of our predictions; this is due to the low number of samples for quality values, other than '5' and '6'.

3 Experimental Set-up

The experimental set-up was mostly similar for all the predictive models. Each model was imported through the Sklearn Python library. Depending on the model, some basic data processing was required to allow proper model training and hyper-parameter optimization; the models concerned are the K-Nearest Neighbors, Logistic Regression and Support Vector Machine Classifier; these models either required the data to be normalized, or benefited by converging quicker such as the Logistic Regression model. Each model then goes through a nested evaluation process consisting of an outer 10-fold resampling loop, and an inner 5-fold resampling loop; the inner resampling loop is responsible for optimizing the hyper-parameters of a selected model; once the best hyper-parameters are found, they are configured to the model; the model is then fitted on a single set of training data from the outer resampling loop; the fitted model is then tested on the test data set. This process repeats until all 10 folds in the outer sampling loop are exhausted. Finally, the performance, of all 10 models tested for each of the 10 folds, are stored and plotted. This implementation follows the process explained by Dr. Bernd Bischl in the video titled : "I2ML - Evaluation - Resampling I".

To optimize the hyper-parameters of the mentioned five models, three optimization algorithms were used : Random Search, Bayesian Optimization, and Bayesian Optimization with Hyper Band (BOHB). Three out of the five predictive models optimized their hyper-parameters using two independent trials of Random Search and Bayesian Optimization; the mentioned models are K-Nearest Neighbors, Random Forest, Decision Tree Classifier. Two of the five optimized their hyper-parameters using two independent trials of Bayesian Optimization and Bayesian Optimization with Hyper Band; the mentioned models are Logistic Regression and Support Vector Machine Classifier. Testing of the models and their different hyper-parameters relied on 10-fold cross validation and were evaluated through an accuracy score.

The following tables describe the hyper-parameter space of which the hyper-parameter optimization algorithms received as inputs, for each predictive model :

Table 1: K-Nearest Neighbors

Hyper-Parameter	Space Type	Range
Number of Neighbors	Integer	[1, 1000]
Weights	Categorical	Uniform or Distance
Algorithm	Categorical	Auto, Ball Tree, KD Tree, Brute
Leaf Size	Integer	[1, 100000]
p	Integer	[1, 100000]

Table 2: Random Forest

Hyper-Parameter	Space Type	Range
Number of Estimators	Integer	[50, 1000]
Criterion	Categorical	Gini, Entropy, Log Loss
Max Depth	Integer	[1, 100]
Minimum Samples Split	Integer	[2, 100]
Minimum Samples Leaf	Integer	[1, 100]
Minimum Weight Fraction Leaf	Real Number	[0.0, 0.5]
Maximum Features	Categorical	Square Root, Log2, None

Table 3: Decision Tree Classifier

Hyper-Parameter	Space Type	Range
Criterion	Categorical	Gini, Entropy, Log Loss
Splitter	Categorical	Best, Random
Max Depth	Integer	[1, 1000]
Minimum Samples Split	Integer	[2, 100]
Minimum Samples Leaf	Integer	[1, 100]
Minimum Weight Fraction Leaf	Real Number	[0.0, 0.5]
Maximum Features	Categorical	Square Root, Log2, None

Table 4: Logistic Regression

Hyper-Parameter	Space Type	Range
Solver	Categorical	LBFGS, Liblinear, Newton-CG, Newton-Cholesky, SAG, SAGA
Penalty	Categorical	L1, L2, Elastic Net, None
C	Real Number	[0.1, 1000]
Maximum Iterations	Integer	[1000, 5000]

Table 5: Support Vector Machine Classifier

Hyper-Parameter	Space Type	Range
C	Real Number	[0.1, 100]
Kernel	Categorical	Linear, Poly, RBF, Sigmoid
Degree	Integer	[1, 20]
Gamma	Categorical	Scale, Auto
Coef0	Real Number	[0, 1]
Tolerance	Real Number	[0.001, 1]

The three implementations of hyper-parameter optimization were possible thanks to three helpful Python packages :

- Random Search - sklearn
- Bayesian Optimization - baytune
- Bayesian Optimization with Hyperband - hpbandster_sklearn

4 Results

After the hyper-parameters, of each model, were optimized, the corresponding accuracy scores and the corresponding hyper-parameters of the best results were kept and a comparison is made between the different hyper-parameter optimization algorithms :

Table 6: Random Search

Model	Mean Model Score	Best Hyper-parameters
K-Nearest Neighbors	$\approx 63\%$	Neighbors : 175 Weights : Distance Algorithm : KD Tree Leaf Size : 7374 p : 529 Score : $\approx 66\%$
Decision Tree Classifier	$\approx 56\%$	Splitter : Best Criterion : Entropy Max. Depth : 22 Min. Samples Leaf : 22 Min. Samples Split : 45 Min. Weight Fraction Leaf : ≈ 0.01 Max Features : None Score : $\approx 60\%$
Random Forest	$\approx 60\%$	Estimators : 398 Criterion : Gini Max Depth : 67 Min. Samples Split : 32 Min. Samples Leaf : 25 Min. Weight Fraction Leaf : ≈ 0.05 Max. Features : Log2 Score : $\approx 68\%$

Table 7: Bayesian Optimization

Model	Mean Model Accuracy Score	Best Hyper-parameters
K-Nearest Neighbors	$\approx 58\%$	Neighbors : 720 Weights : Distance Algorithm : Auto Leaf Size : 48994 p : 7429 Score : $\approx 66\%$
Decision Tree Classifier	$\approx 57\%$	Splitter : Best Criterion : Gini Max. Depth : 289 Min. Samples Leaf : 30 Min. Samples Split : 27 Min. Weight Fraction Leaf : ≈ 0.28 Max Features : None Score : $\approx 67\%$
Logistic Regression	$\approx 59\%$	Solver : LBFGS C : ≈ 325.40 Max Iterations : 1137 Penalty : None Score : $\approx 71\%$
SVM Classifier	$\approx 63\%$	C : ≈ 13.22 Coef0 : ≈ 0.75 Degree : 10 Gamma : Scale Kernel : RBF Tolerance : ≈ 0.17 Score : $\approx 68\%$
Random Forest	$\approx 63\%$	Estimators : 196 Criterion : Gini Max Depth : 35 Min. Samples Split : 8 Min. Samples Leaf : 6 Min. Weight Fraction Leaf : ≈ 0.01 Max. Features : Sqrt Score : $\approx 69\%$

Table 8: Bayesian Optimization with Hyperband

Model	Mean Model Accuracy Score	Best Hyper-parameters
Logistic Regression	$\approx 59\%$	Solver : LBFGS C : ≈ 2.26 Max Iterations : 2288 Penalty : L2 Score : $\approx 71\%$
SVM Classifier	$\approx 63\%$	C : ≈ 16.92 Coef0 : ≈ 0.18 Degree : 3 Gamma : Auto Kernel : RBF Tolerance : ≈ 0.36 Score : $\approx 69\%$

The following figures display the performance estimates (accuracy scores) for each model, after each hyper-parameter optimization session. The performance estimates correspond to 10 different evaluations due to 10-fold cross validation for the outer sampling.

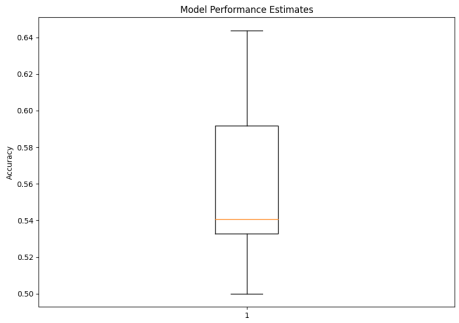


Figure 1: DTC Random Search

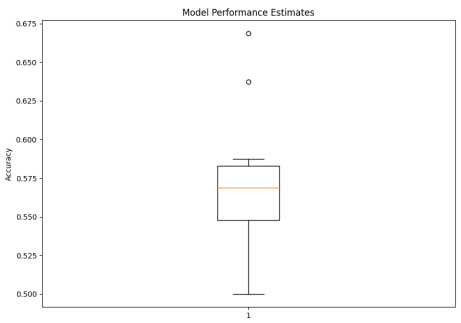


Figure 2: DTC Bayesian Search

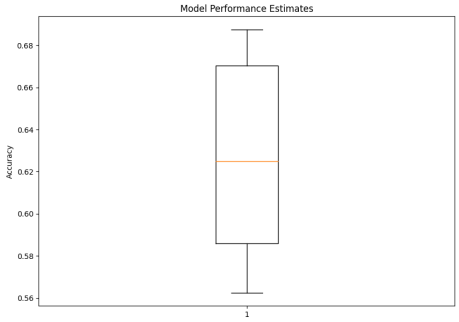


Figure 3: KNN Random Search

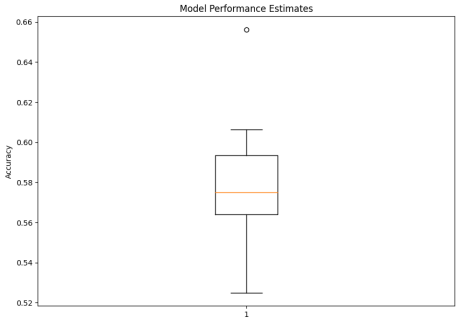


Figure 4: KNN Bayesian Search

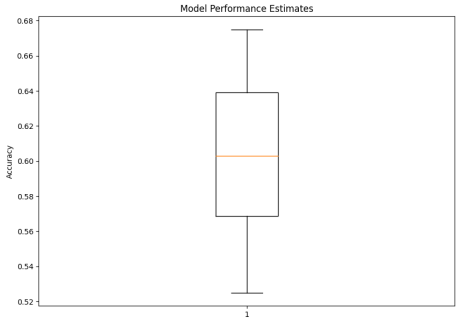


Figure 5: RF Random Search

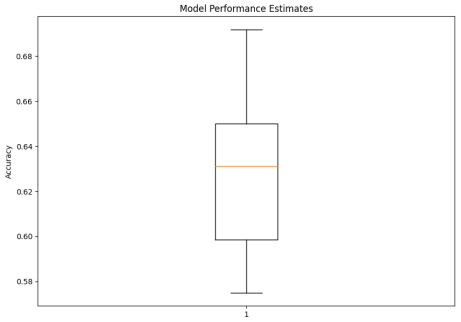


Figure 6: RF Bayesian Search

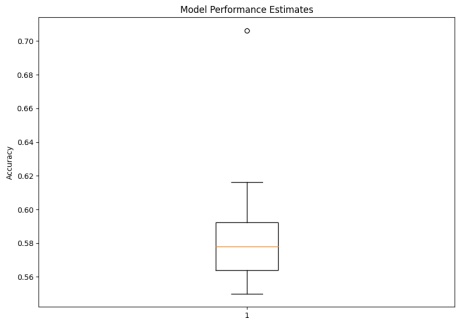


Figure 7: LR Bayesian Search

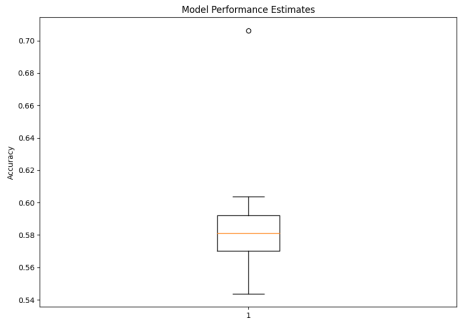


Figure 8: LR BOHB Search

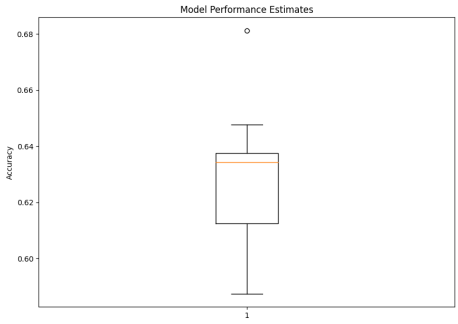


Figure 9: SVM Bayesian Search

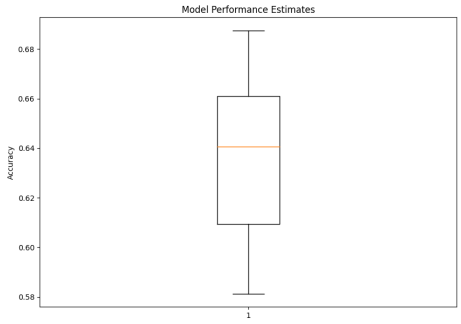


Figure 10: SVM BOHB Search

Using the best average accuracy score for each model, the following table compares the performance of the non-optimized predictive models with the optimized predictive models :

Table 9: Non-optimized vs. Optimized Model Comparison

Model	Non-optimized Accuracy Score	Optimized Accuracy Score
K-Nearest Neighbors	$\approx 51\%$	$\approx 63\%$
Random Forest	$\approx 59\%$	$\approx 63\%$
Decision Tree Classifier	$\approx 56\%$	$\approx 57\%$
Logistic Regression	$\approx 58\%$	$\approx 59\%$
SVM Classifier	$\approx 59\%$	$\approx 63\%$

NOTE : Non-optimized models were evaluated again, with normalized data to allow for unbiased comparisons with the optimized models. Thus, the results are different from the previous report. These updated results can be found in the folder containing all the code files.

From our results, there are three main observations. Firstly, we can observe that the Bayesian Optimization method consistently found hyper-parameter configurations with a better accuracy score, than the Random Search method; this was the result despite more iterations being allowed for Random Search (50:30). Secondly, we notice fairly similar accuracy scores from the hyper-parameter configurations when comparing between the Bayesian Optimization and BOHB methods. Finally, in terms of model performance, we notice that the highest-performing optimized models, on average, are : K-Nearest Neighbors, Random Forest, and the SVM Classifier, which are all tied at an accuracy score of approximately 63%. Despite that, we also observed that the best performing hyper-parameter configuration was from the Logistic Regression model, which had an accuracy score of 71%.