Practical Machine Learning

Hyperparameter Optimization

Sanjeeb Humagain

April 29, 2024

Introduction:

The main goal of this exercise is to use hyperparameter tuning for Machine Learning (ML) models which can have significant impact on the model's performance. In this report, Random Forest Regression and Neural Network Regression are optimized by implementing Bayesian Optimization and Random Search Optimization and the results are compared and finally suggesting the most appropriate optimization technique for this specific data and problem.

There are a significantly various methods which could be used for optimization of a ML. However, selection of appropriate technique for a particular dataset can be quite challenging task. After reviewing several resources, it was found that Bayesian optimization and random search optimization methods are widely used technique for hyperparmeter optimization. Therefore, those two methods were chosen in this task.

To select an appropriate ML algorithm and the optimization method, first we need to understand the problem. For instance, if the problem is related to classification we have to choose a different algorithm than an algorithm for prediction. For instance, Linear Regression is used for forecasting and Logistic Regression algorithm is used for classification problems.

Correlation of features were checked and there were some negative correlations as well. A negative correlation is a relationship between two variables that move in opposite directions. From the correlation plot of heatmap, it was fond that some features have negative correlation with system load indicating that system load decreases as the particular feature increases and vice versa. The one advantage of checking the correlation is that we could remove the features which has very less correlation or zero correlation with the data we want to forecast. Removing such features would reduce the complexity and might ultimately help to increase the model performance.

Following libraries are used for this task:

- pandas
- seaborn
- matplotlib.pyplot
- numpy
- train_test_split
- BayesianOptimization from bayes_opt
- mean_absolute_error, mean_squared_error, r2_score from sklearn.metrics
- RandomForestRegressor
- RandomizedSearchCV

Some of the libraries used are listed above. They are used for data preprocessing, numerical computation, forecasting, optimization and performance evaluation of the model.

While going through the process of coding and evaluation of different optimization techniques, it was found that one optimization technique is better than other. In Bayesian optimization, it was easier to define bounds for the hyperparameters to be tuned. We could define the range as a tuple with minimum and maximum value. However, random search optimization technique needed more lines of codes and defining the values of specific hyperparameter needed quite a large number of values listed as an array from which the model will random select the values and give the best hyperparameter. So, if we are not sure about the values to use in this case, it would be better to Bayesian Optimization because, it will allow you to choose a range of value (minimum and maximum) will do the rest of the task for you. On contrary, if your program to check hyperparameters among the exact values you used, random search would be a better option. Furthermore, in terms of execution time, Bayesian Optimization found to be taking a significant amount of time. Thus, random search might give you a solution in less amount of time.

In all ML algorithms used, roughly same steps are followed. They are, checking for the missing value, data preprocessing, clean up, scaling, split up, creating model, training the model and running the test, and hyperparameter optimization etc. After performing the hyperparameter optimization, the same model was run with the optimized set of values of the hyperparameters. Finally, the performance of the model and hyperparameters are summarized and discussed by comparing mean square error, mean absolute error and r2 score of each of the options considered in this experiment.

Dataset Description:

In this experiment, the dataset has 9 features which are used to predict the system load. The data set contains 87648 samples of data and the target system load from 2006 to 2011. In this exercise, the system load data is being used to compare machine learning algorithms and optimization methods.

Experimental Setup:

Python is used for this experiment because of the availability of crucial libraries for instance Pandas (for data manipulation), Scikit-learn (for ML) etc. The dataset imported from the csv file and split into training and testing sets in ratio of 80:20 in the same way as performed in previous exercises.

Two ML algorithms are used for evaluation namely, Random Forest (RF) and Neural Network (NN). Similarly, two optimization techniques, Bayesian optimization and Random Search optimization methods are implemented for hyperparameter optimization process. The performance of each model was assessed using mean square error, mean absolute error and r2 score. Finally, the most appropriate algorithm and optimization method was selected among each of two under the consideration.

Nested resampling is a process that provides reliable estimate of a model's performance and assists to prevent overfitting during the entire process of hyperparameter tuning. Specially, it is important when evaluating the performance of various models or even when comparing the different techniques for hyperparameter optimization. In the code, first the training and testing data are split

in ratio of 0.8/0.2 which works as outer loop for model evaluation. Hyperparameter tuning is performed on the training folds, while the validation fold is used to evaluate the performance of the model with the selected hyperparameters. Additionally, for each iteration of the outer loop, an inner loop is used to tune the hyperparameters of NN or RF. For instance RandomizedSearchCV is used as the hyperparameter tuning method, which randomly samples hyperparameters from the defined parameter grid ('param_grid') and evaluates the performance using cross validation on the training folds. Finally, the combination of hyperparameters that yields the best performance (i.e. lowest negative MSE) is selected as the best hyperparameters. 5 fold cross validation was used for both optimization technique for Random Forest as well as Neural Network method.

Results:

The MSE, MAE and R2 score of both algorithms using both optimization methods are tabulated below.

S.N.	Metrics	ML algorithms					
		Random Forest (RF)			Neural Network (NN)		
		Before HPO	Bayesian Optimization	Random Search	Before HPO	Bayesian Optimization	Random Search
1	Mean Squared Error (MSE)	234563.13	95619.61	94267.17	524807.42	230356.23	225125.52
2	Mean Absolute Error (MAE)	368.17	228.13	226.52	589.29	371.77	369.21
3	R2 Score	0.8833	0.9524	0.9531	0.7391	0.8854	0.8880
4	Execution Time	3.9 sec	124 min	12 min	5.3 sec	136 min	20 min

Table 1: MSE, MAE, R2 score and execution time before and after hyperparameter optimization

Talking about the range of hyperparameters used, the values are tabulated in table 2 and table 3 for RF and NN respectively. We tried to include more values that covers the default hyperparameters as well. The primary objective of random search is to explore the hyperparameter space randomly in search of good configurations, rather than systematically exploring all values in between the specified ranges. The tuned values for each hyperparameter might not be the best solution because there are infinite possibilities of different combination of hyperparameter and finding that value may take forever. We could say that the tuned value is the best value on the range we defined. Also, few hyperparameters are considered in this exercise as listed in table 2 and 3 and the range we defined are not best but we considered what seems fair considering time

of execution. Moreover, we could consider wide range of hyperparameters for random search as it is taking considerably less amount of time than Bayesian Optimization.

S.N.	Hyperparameters	Random Forest (RF) ML algorithm					
		Random Search			Bayesian Optimization		
		Before HPO	Range defined	Tuned value	Before HPO	Range defined	Tuned value
1	n_estimators	20	(10, 200)	200	20	(10, 200)	173
2	max_depth	10	(1, 30)	30	10	(1, 30)	29
3	min_samples_split	5	(2, 10)	2	5	(2, 10)	2
4	min_samples_leaf	5	(1, 4)	1	5	(1, 4)	1
5	max_features	None	['sqrt', 'log2', None]	'log2'	None	('sqrt', 'log2', None)	'sqrt'

Table 2: Hyperparameter values before and after Optimization for RF

S.N.	Hyperparameters	Neural Network (NN) ML algorithm					
		Random Search			Bayesian Optimization		
		Before HPO	Range defined	Tuned value	Before HPO	Range defined	Tuned value
1	learning_rate_init	0.01	(1e-5, 1e-1)	0.1	0.01	(1e-5, 1e-1)	0.1
2	hidden_layer_sizes	64	(10, 200)	200	64	(10,200)	200
3	max_iter	30	(10,100)	100	30	(10,100)	100
4	activation	relu	['identity', 'logistic', 'tanh', 'relu']	'logistic'	relu	['identity', 'logistic', 'tanh', 'relu']	'logistic'

Table 3: Hyperparameter values before and after Optimization for NN

The metrics before and after hyperparameter optimization are tabulate in table 1. Lets' compare them with respect to the R2 score values before and after the hyperparameter optimization. We can see that the R2 score values for Random Forest is higher than that of Neural Network for both before and after HPO. However, there have been a significant improvement of R2 score value from

0.7391 to 0.8854 after Bayesian optimization in NN algorithm. From table 1, we can conclude that Random Search is better than Bayesian Optimization method for NN both in terms of execution time and R2 score. On the other hand, if we consider R2 score values only, then both of optimization technique can be used as there is not much difference in R2 score. Therefore, both of them can be applied for both algorithms for this specific task and dataset. There is improvement in R2 score for both of the ML algorithm after hyperparameter optimization. This suggests that, it is useful to use hyperparameter optimization to improve performance of any of the ML models.

In RF, Bayesian Optimization has more R2 score than Random search even though, there is not much difference. Additionally, in terms of execution time, Bayesian optimization took far more time than Random search. Execution time for random search is less in NN but, for Bayesian optimization, RF has less execution time. The presence of a ConvergenceWarning suggests, in my view, that additional iterations are necessary to reach an optimal solution. This indicates that Neural Network would take considerable amount of time and the model performance is considerably less than RF.

Considering the time of execution, Random search is doing better job for optimization in both RF and NN. Furthermore, RF has better model performance than NN which could be seen in table 1. All in all, RF is better more and Random search is the better optimization method for this specific dataset.

In conclusion, we cannot simply define which method and optimization technique is universally better or applicable in all the scenarios because, the cost, priority and performance of problems can change significantly for every ML problem we try to solve.

The Plots of all the algorithms used are listed below.

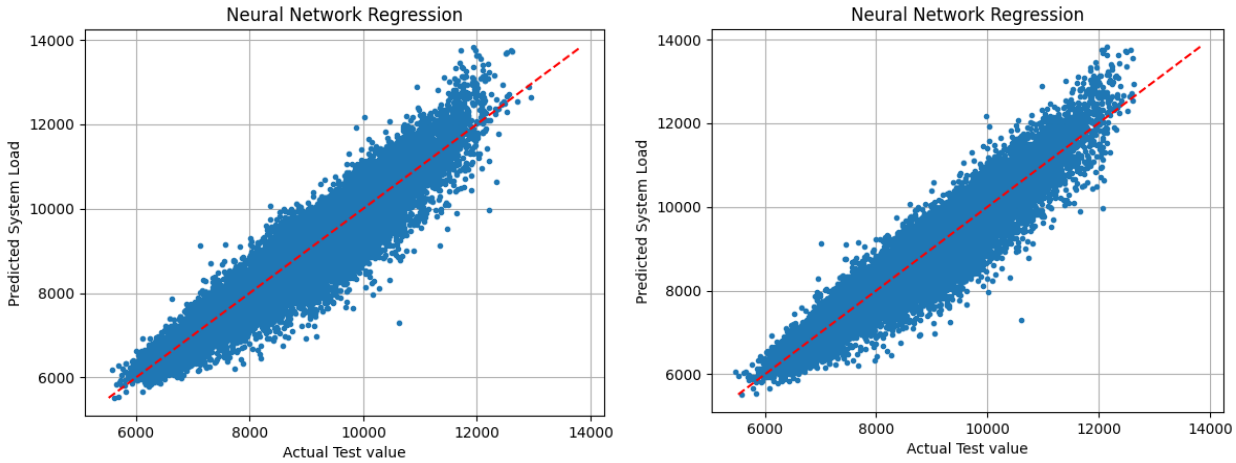

Fig1: Actual vs predicted value using NN Bayes HPO and Random Search HPO

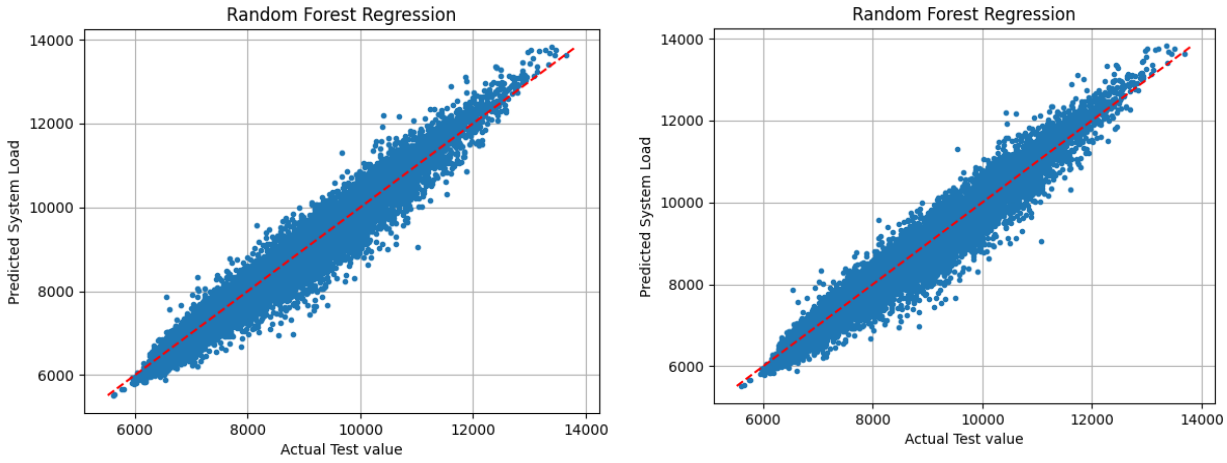

Fig2: Actual vs predicted value using RF Bayes HPO and Random Search HPO