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.

Results:

The R2 Score result of the ML algorithm used for this exercise is tabulated below in a descending oreder.

The R2 score before hyperparameter optimization is tabulated below.

S.N.	ML Algorithm	R2 Score
1	Random Forest (RF)	0.9412
2	Neural Network (NN)	0.7595

Table 1: R2 score before hyperparameter optimization

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)	
		Bayesian Optimization	Random Search	Bayesian Optimization	Random Search
1	Mean Squared Error (MSE)	95356.3260	88282.4204	201110.0312	240081.1906
2	Mean Absolute Error (MAE)	221.2872	215.2299	343.7699	381.5491
3	R2 Score	0.9525	0.9561	0.9000	0.88064

Table 2: MSE, MAE and R2 score before and after hyperparameter optimization

The metrics before and after hyperparameter optimization are tabulate in table 1 and table 2 respectively. Lets' compare them with respect to the R2 score values as we have R2 score 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 0.7595 to 0.9000 after Bayesian optimization. From table 2, we can conclude that Bayesian Optimization is better than Random Search method for NN because there is a significant difference in R2 score. On the other hand, for Random Forest R2 score values do not have much difference. Therefore, both of them can be applied for RF algorithm for this specific task and dataset. There is improvement in R2 score for both of the ML algorithm. This suggests that, it is useful to use hyperparameter optimization to improve performance of any of the ML models.

In RF, Bayesian Optimization has less R2 score than Random search even though, there is not much difference. Additionally, in terms of execution time, Bayesian optimization took for more time than Random search which might due the range of values it is evaluating. Still, Random search seems to be better for RF algorithm and Bayesian Optimization seems to be better for NN algorithm based upon the performance metrics generated for this specific case.

The ranges of the tuned hyperparameters for random search for Neural Network are:

```
Best Hyperparameters: {'max_iter': 100, 'learning_rate_init': 0.01, 'hidden_layer_sizes': (100, 100, 100)}
```

The ranges of the tuned hyperparameters for Bayesian optimization for Neural Network are:

```
Best Hyperparameters: {'hidden_size1': 132.76893381154952, 'hidden_size2': 86.30261322586993, 'hidden_size3': 39.29205805008313, 'lr': 0.01665194604315761, 'num_epochs': 82.1321555727439}
```

I have used lr=0.001, num_epochs = 20 and sigmoid is used as activation function as the default values because I did not consider all the hyperparameters. I tried to include more hyperparameters during HPO and the best values are listed above for random search and Bayesian Optimization.

The ranges of the tuned hyperparameters for random search for Random Forest are:

```
Best Hyperparameters: {'max_depth': 29.533322771623297, 'min_samples_leaf': 1.6662665625678468, 'min_samples_split': 2.1499900648821173, 'n_estimators': 117.02540120580396}
```

The ranges of the tuned hyperparameters for Bayesian optimization for Random Forest are:

```
Best Hyperparameters: {'n_estimators': 50, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_features': 'log2', 'max_depth': None, 'bootstrap': False}
```

I have used n_estimators=10, random_state=0 as the default values because I did not consider all the hyperparameters. I tried to include more hyperparameters during HPO and the best values are listed above for random search and Bayesian Optimization.

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.

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.

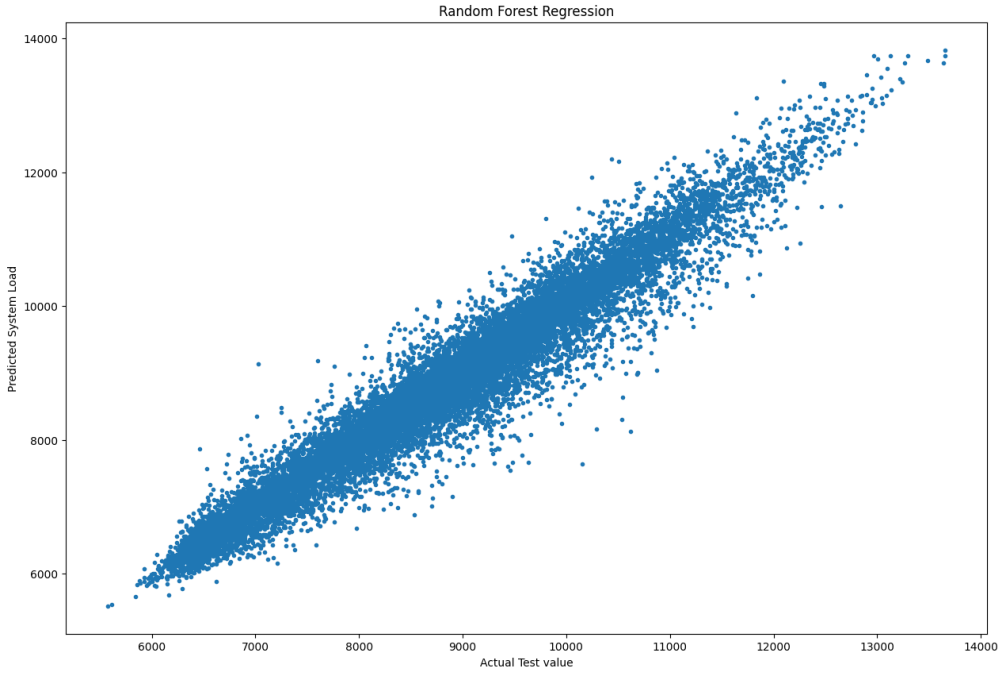

Fig1: Actual vs predicted value using RF regression before hyperparameter optimization

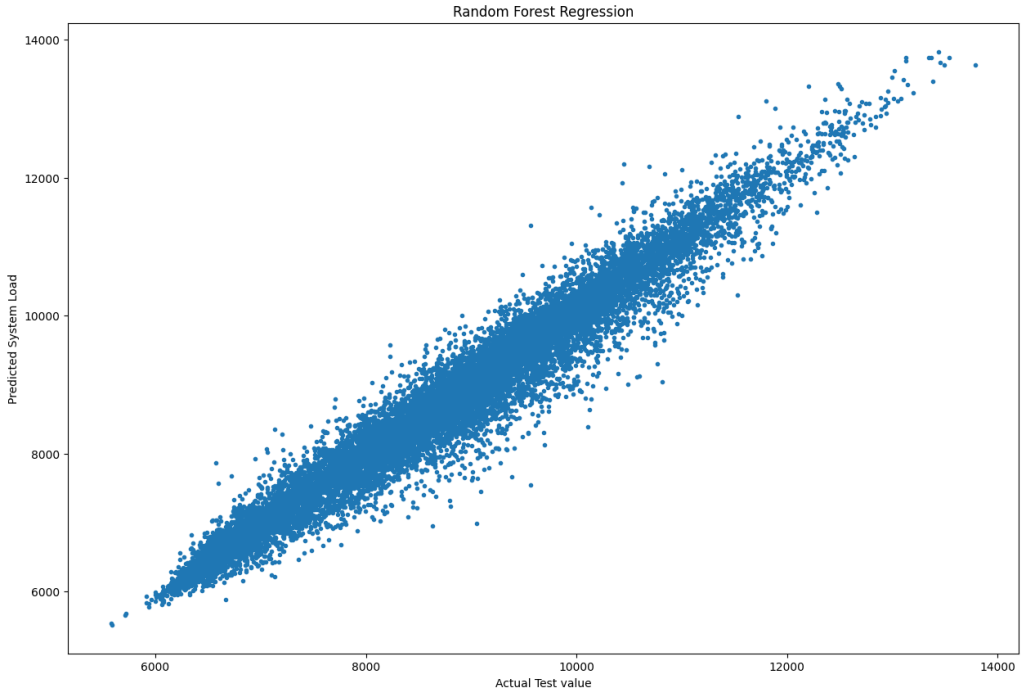

Fig2: Actual vs predicted value using RF regression after random search HPO

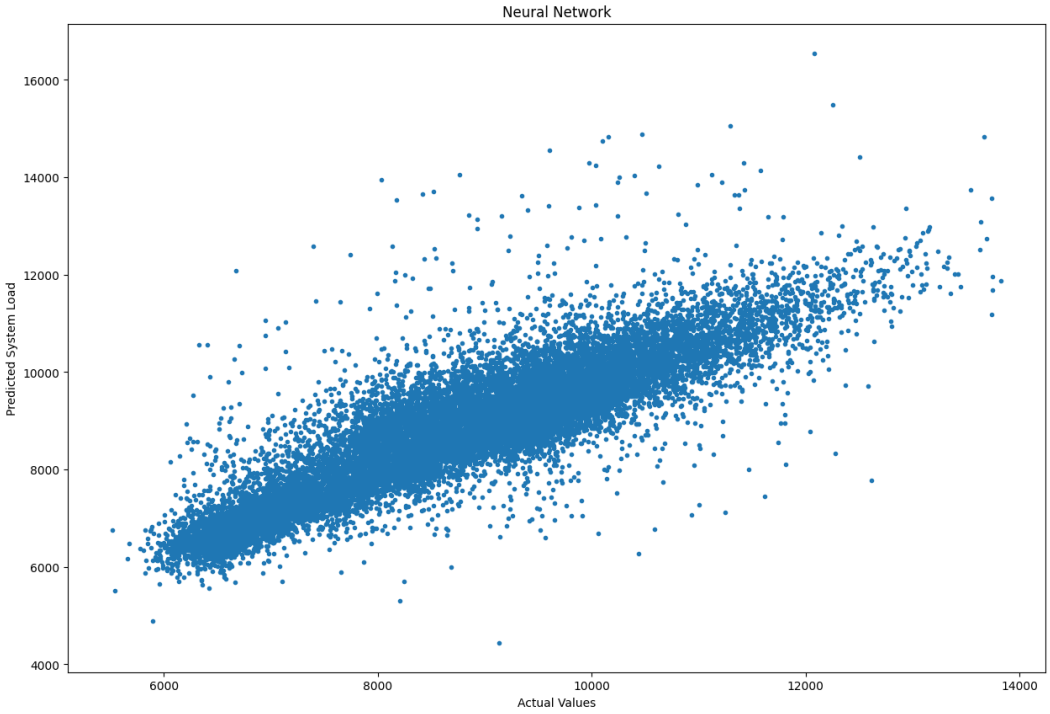

Fig3: Actual vs predicted value using NN regression before hyperparameter optimization

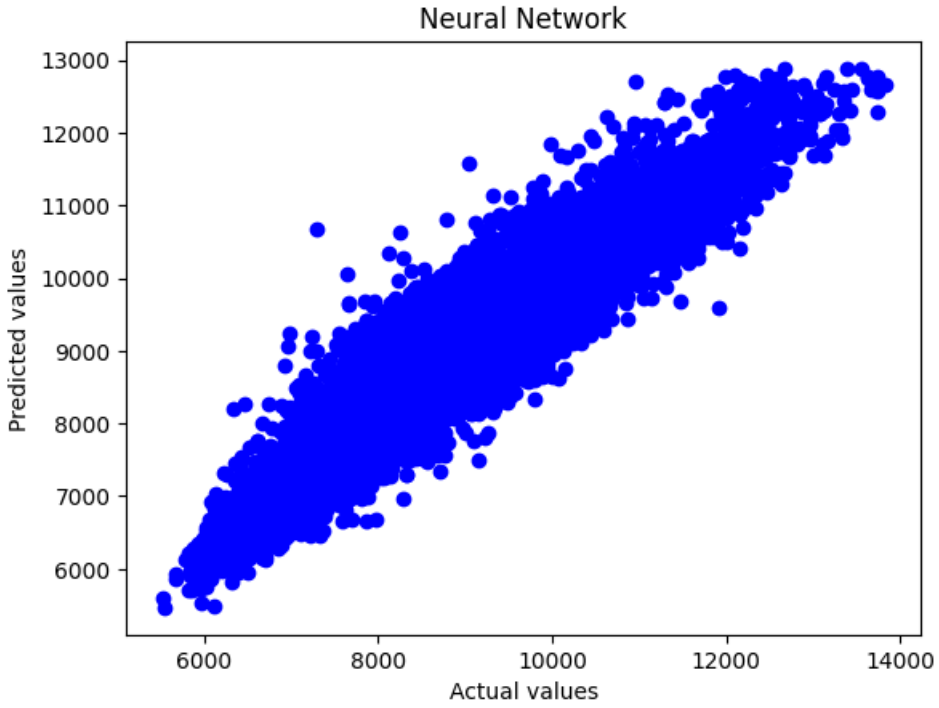

Fig4: Actual vs predicted value using NN regression after Bayes HPO

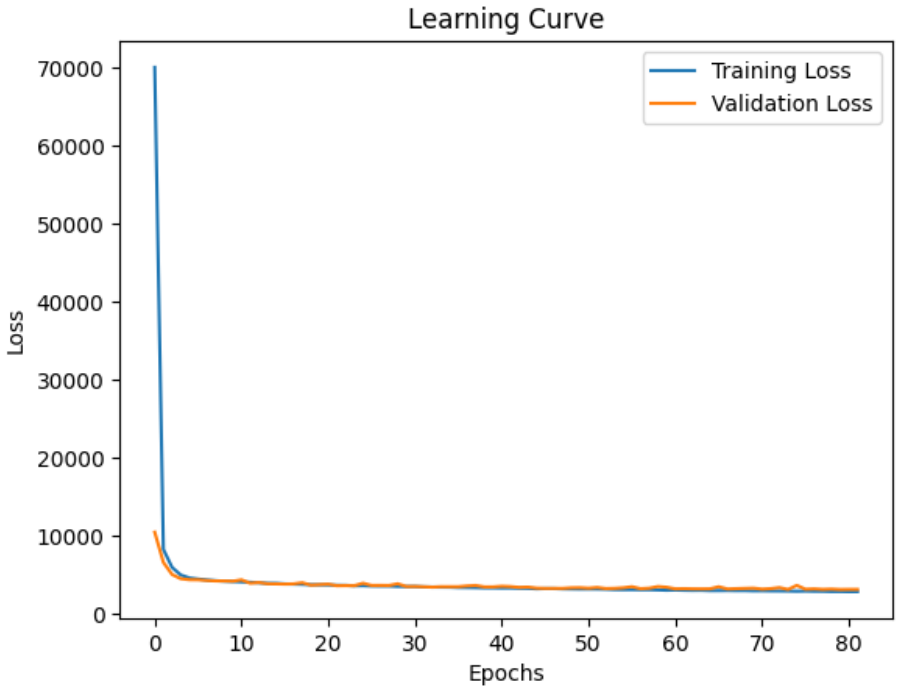

Fig5: learning curve for NN regression after Bayes HPO

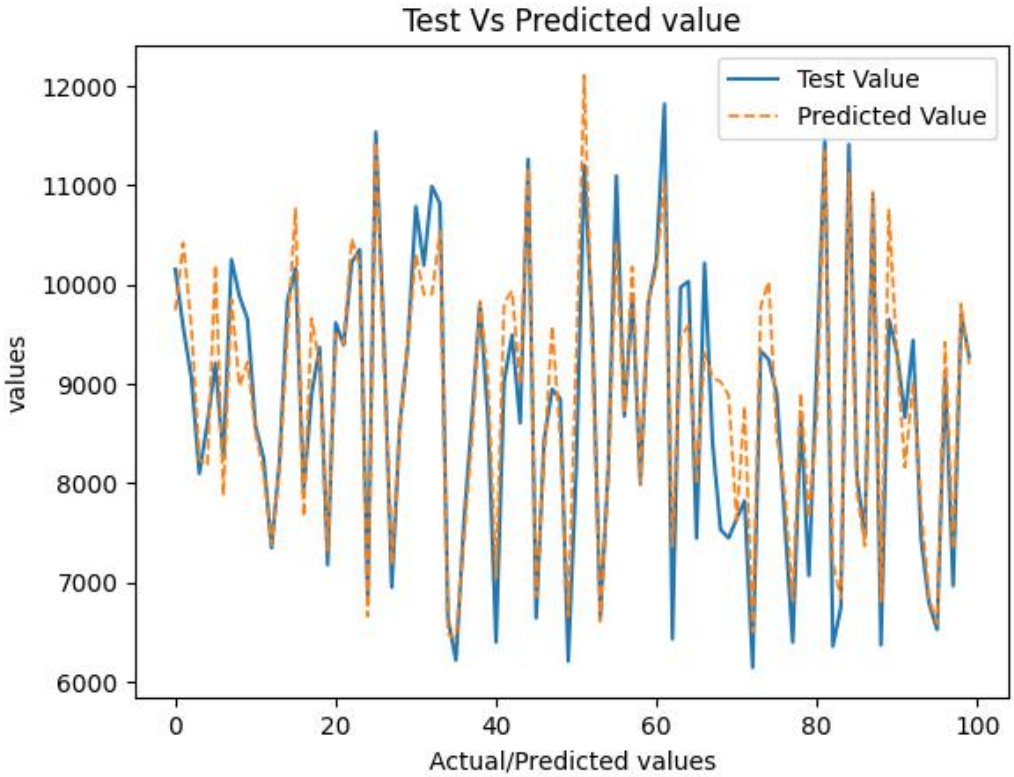

Fig6: line plot for NN regression after Bayes HPO (only 100 to 200 sample data were used)