Practical Machine Learning

Hyperparameter Optimization

Bimal Pandey

Introduction: This report talks about the hyperparameter optimization of different ml models. The performance of every ml model depends upon their hyperparameter selection. They control the learning algorithm or the structure of the model. First of all, different ml models are applied to predict the wine quality and after that, the hyperparameters of ml algorithms are tuned using Bayesian optimization and Randomized search to obtain the best parameters.

Dataset Description: The Red Wine dataset consists of 1599 observations and 12 characteristics, out of which 11 are input variables and the remaining one is output variable. Here, the data have only float and integer values (only for the target variable) and there are no null/missing values. The describe() function returns the count, mean, standard deviation, minimum, 25%, 50%, 75%, and maximum values and the qualities of data. The duplicate records are removed using data.drop_duplicates(inplace=True).

Input Variables:

- fixed Acidity
- volatile acidity
- citric acid
- residual sugar
- chlorides
- free sulfur dioxide
- total sulfur dioxide
- density
- pH
- sulphates
- alcohol

Output variable:

- quality

Used libraries and modules: To start with, I imported the following libraries and loaded the dataset, and the original data are separated by ";" in the given data set.

- Numpy: It will provide support for efficient numerical computation
- Pandas: It is a convenient library that supports data frames. Working with pandas will bring ease to many crucial data operations.
- Seaborn: It is a visualization library based on Matplotlib which provides a high-level interface for drawing attractive statistical graphics.
- Sklearn: It is a Python library for data mining, data analysis, and machine learning.
- Matplotlib: It provides a MATLAB-like plotting framework

Experimental Setup: Random Forest classifier and support vector machine algorithms are used for the predictions of the wine quality. The hyperparameters of both algorithms are tuned using Bayesian optimization and the randomized search method. **(In my first attempt, I used grid search for hyperparameter optimization but that didn't prove to be better due to small search space)**

Hyperparameters Range :

The hyperparameters that are tuned during Randomized Search for random forest classifier are 'n_estimators' with range (50, 400). The randint has been used for selecting randomly the integer values within the range. The range for 'max_depth' was (10,30). The range for 'min_sample_split' was (2,11) and 'randint' was used here as well. 'min_sample_leaf" was selected between (1,5). The 'randint' function was used for both of these features. Similarly, the ranges of hyperparameters that were tuned using Bayesian optimization for random forest classification were n_estimators= (50, 400), max_depth(10, 30), min_sample_split(2, 11) and min_samples_leaf (1, 5).

The hyperparameters for support vector machine using Randomized Search were 'c' in the range of (0.1, 10), 'kernel' with 'linear', 'rbf', 'poly', sigmoid' and 'gamma': ['scale', 'auto']

```
'SVM': {
    'C': stats.uniform(0.1, 10),
    'kernel': ['linear', 'rbf', 'poly', 'sigmoid'],
    'gamma': ['scale', 'auto']
  }
```

Similarly, the hyperparameters optimization for SVM using optuna was done. The range of hyperparameters that were tuned were:

```
param_grid = {
  'C': Real(0.1, 10, prior='log-uniform'),
  'kernel': Categorical(['linear', 'rbf', 'poly', 'sigmoid'])
}
```

Nested resampling and discussion of hyperparameter optimization procedure:

The outer loop is represented by the 'for clf_name, clf in classifiers.items(): loop. It iterates over each classifier that is defined under 'classifiers'. For each classifier, the outer loop sets the parameter grid for the particular classifier and performs the Randomized cross-validation to find the best hyperparameters for both the random forest and SVM classifier. The outer loop uses 5-fold for cross-validation. The 'outer_cv'

object (‘StratifiedKFold) splits dataset into 5 folds and ‘cross_val_score’ is applied with the ‘RandomizedSearchCV’ model and the outer cross-validation. The training set is used for hyperparameter tuning via randomized search. Within each iteration of the outer loop, the training set is further split into training and validation sets. For each split in the outer loop, the inner loop performs a randomized search cross-validation on the training set, exploring various hyperparameters to find optimal configurations which are listed/explained in the above discussion. The inner loop, which is performed by ‘cross_val_score’ function uses the 5-fold cross-validation and the process is repeated for each split in the outer loop, allowing for robust estimation. The ‘final_scores’ dictionary is used to store the test scores of each classifiers after the outer cross validation loop has been completed and over the iteration it only stores the best parameters score which means the best hyperparameters over the whole iteration are stored and displayed. After the outer cross-validation loop, the final accuracies for each classifier are printed on the test scores stored in ‘final_scores’. So, ‘final-scores’ is used to keep track of the test scores of each classifier from the best obtained hyperparameter over the iteration. The best hyperparameters selected in any fold of inner cross-validation by random search are stored in cv_scores and their selection is based on the average(np.mean(cv_scores):.4f) performance across all folds. Moreover, the outer cross-validation loop ensure that the chosen hyperparameters are not overfitted to any particular subset of the training data.

In summary, for randomized search method for random forest and svm, we have performed nested resampling where both inner and outer splits were equal to 5 for our dataset. Then, the random forest model and SVM were fitted on the input and target variables using the fit() function. We have then identified the best hyperparameters for our dataset and have used mean accuracy score {np.mean (cv_scores):.4f}") as the evaluation matric and values for hyperparameters and mean accuracy is mentioned in the result section. This way, the best hyperparameters chosen by **RandomizedSearchCV** across all folds of the inner cross-validation are used consistently throughout the outer cross-validation loop.

Similarly, for Bayesian Optimization of random forest, the objective function ‘rf_eval’ has been initialized and the search space for the hyperparameters have been defined as mention above. After this we have used maximize () function for bayesian optimization to maximize the objective function objective function and the number of iterations equal to 100 has been used. We have then identified the best hyperparameters for our dataset and have used accuracy score as the evaluation matric and values for hyperparameters and accuracy is mentioned in the result section.

Results and Discussion: The performance of both classifiers has been increased by using the tuned parameters obtained from the hyperparameter optimization. The accuracy scores for random forest and SVM with default hyperparameter/before hyperparameter optimization were 0.6 and 0.5 respectively. After using the tuned/best parameters obtained by Bayesian optimization the accuracy score increased to 0.67 and 0.6 respectively for both. Similarly, after the randomized search method to find the best parameters, the accuracy score for random forest and SVM were increased to 0.672 and 0.60.

ML model	Best Hyperparameters- Randomized Search	Best Hyperparameters- Bayesian Optimization
Random Forest	{'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_split': 4, 'n_estimators': 104}	{'max_depth': 27.67931192827918, 'min_samples_leaf': 2.124883215417278, 'min_samples_split': 5.409779166141391, 'n_estimators': 113.52513963120876}}

Support Vector Machine	'C': 8.424426408004217, 'gamma': ' auto', 'kernel': 'rbf'}	('C', 4.2149456283335), ('kernel', 'li near')])