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 grid 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 grid search method.

- **Hyperparameters Range and discussion about default and best-obtained parameters**

Hyperparameters for Random forest classification using Bayesian optimization

The ranges of hyperparameters that were tuned using Bayesian optimization for random forest classification are listed below. The parameters that were tuned are a number of estimators, maximum depth, minimum samples leaf, minimum samples split, and maximum features. The tuned parameters gave better accuracy than the default.

```
'n_estimators': (10,300),

   'max_depth': (3,30),

   'min_samples_leaf': (1,4),

   'min_samples_split':(2,10),}

max_features={'sqrt', 'log2'}
```

From the optimization, the **best parameters** that were obtained are:

- N_estimators: 63
- Max_depth: 24
- Min_samples_leaf: 1
- Min_samples_ split: 3

The ranges of hyperparameters for support vector machine using BayesSearchCV are:

```
param_grid = {

   'C': Real(0.1, 10, prior='log-uniform'),

   'kernel': Categorical(['linear', 'rbf', 'poly', 'sigmoid'])

}
```

The **best** hyperparameters that were obtained from optimization are:

C: 4

Kernel: linear

The hyperparameters that were used for the random forest classifier and support vector machine during the grid search method are described below. The search space was made large enough initially but that made the simulation time much longer; the following values were tuned to obtain best parameters.

```
param_grids = {

   'Random Forest Regressor': {

      'n_estimators': [50, 100, 200, 300,400,500,600,700],
```

'max_depth': [None, 10, 20, 30],

'min_samples_split': [2, 5, 10],

'min_samples_leaf': [1, 2, 4]

The **best** hyperparameters that were obtained from the grid search are:

Max_depth: 20

Min_samples_split= 5

N_estimators= 200

Min_samples_leaf= 3

Max_features= sqrt

The hyperparameters that were tuned for support vector machine were:

'SVM': {

'C': [0.1, 1, 10],

'kernel': ['linear', 'rbf', 'poly', 'sigmoid'],

'gamma': ['scale', 'auto']

}

The grid search method hyperparameter optimization obtained the following **best** parameter for SVM:

C: 10

Kernel: rbf

Gamma: auto

- **Nested resampling**

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 Grid Search cross-validation to find the best hyperparameters for both the random forest and SVM classifier. The outer loop uses 5-fold for cross-validation. For each split in the outer loop, the inner loop performs a grid 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 3-fold cross-validation and the process is repeated for each split in the outer loop, allowing for robust estimation. After finding the best hyperparameters for each split, ‘cross_val_score’ trains the model with those hyperparameters on the training set of the split.

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 0.65 and 0.55 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 grid search method to find the best parameters, the accuracy score for random forest and SVM were increased to 0.66 and 0.60. Hence, we can conclude that the Bayesian optimization performed better for both classifiers than the grid search method.