Pipeline Optimization

Ali Torabi

1. Introduction

Machine learning is changing our lives swiftly. While a simple task like image recognition seems so trivial for humans, there is a lot of work to do in the machine learning pipeline, such as data cleaning, feature engineering, finding which models best fit for the specific problem, and hyperparameters tuning, among many other tasks. A lot of these tasks are still not automated and need an expert to do some of these trials and errors. Automated Machine Learning (AutoML) provides a process to automatically discover the best machine learning model for a given task according to the dataset with very little expert need. In this experiment, instead of optimizing a single component, we want to optimize the entire ML approach, including any preprocessing. For this experiment, again the White Wine dataset has been used to predict the quality of wines. The classifiers that has been used for this experiment are the Gradient Boosting, Random Forest, and K-Neighbors Classifier. Also, the Bayesian Optimization is used as hyperparameter optimization.

2. Dataset Description

The dataset chosen for this experiment is the Wine Quality dataset [1]. It is related to white wine samples from the north of Portugal. It comprises 1599 instances and 11 different features. The label is the quality of the wine that makes the data suitable for Classification and Regression tasks. Each of these algorithms would detect the quality of wine ranges from poor to excellent. As mentioned in the description of the dataset itself, it has no missing value. The features are: fixed_acidity, volatile_acidity, citric_acid, residual_sugar, chlorides, free_sulfur_dioxide, total_sulfur_dioxide, density, pH, and sulphates. The output variable is quality which is scored between 0 and 10. We can use the Pandas describe method to show some of the main properties of the dataset.

	fixed acidity	volatile acidity	citric acid	residual sugar	pH	sulphates	alcohol	quality
count	4898	4898	4898	4898	4898	4898	4898	4898
mean	6.854788	0.278241	0.334192	6.391415	3.188267	0.489847	10.51427	5.877909
std	0.843868	0.100795	0.12102	5.072058	0.151001	0.114126	1.230621	0.885639
min	3.8	0.08	0	0.6	2.72	0.22	8	3
25%	6.3	0.21	0.27	1.7	3.09	0.41	9.5	5
50%	6.8	0.26	0.32	5.2	3.18	0.47	10.4	6
75%	7.3	0.32	0.39	9.9	3.28	0.55	11.4	6
max	14.2	1.1	1.66	65.8	3.82	1.08	14.2	9

3. Experimental Setup

First, we need a library like Pandas to import and manipulate the dataset. We import CSV data into the Dataframe using pandas. To understand the structure of the data, we can use some of the descriptor methods used in Pandas, like Shape, Info, and Describe. Then, the dataset also split into Train, Validation and Test sets. After splitting data, the preprocessing pipeline has been constructed. The first part of the pipeline uses imputer as a tool for the imputation of missing values if any exist. The imputer replaces nan values (missing values) with the mean of the column (imputer is used as a general case for implementing pipeline even though for this specific dataset there is no missing values). Then it uses the Standard Scaler as a way to standardize features by removing the mean and scaling to unit variance.

$$Z = \frac{x - \mu}{\sigma}$$

Where μ is mean and σ is standard deviation. As a categorial transformer, the imputer and on-hot-encoding have been used.

First, All classifiers go through the simple training phase without any hyperparameter optimization in order to having a baseline. In comparison, the pipeline created and Bayesian Optimization tries to tune whole hyperparameters related to preprocessing and the model.

In training phase, I also setup the nested resampling. To do this, for each classifier the hyperparameters will be tuned using Bayes Optimization. Nested resampling uses an additional layer of resampling that separates the tuning activities from the process used to estimate the efficacy of the model. An *outer* resampling scheme is used and, for every split in the outer resample, another full set of resampling splits are created on the original analysis set. For example, if 10-fold cross-validation is used on the outside and 5-fold cross-validation on the inside, a total of 500 models will be fit. The parameter tuning will be conducted 10 times and the best parameters are determined from the average of the 5 assessment sets. This process occurs 10 times [5]. The outer cross-validation as Stratified K-Fold with number of splits equal to 5 has been used. The number of inner fold using during hyperparameter optimization with each iteration of the outer loop is 3. In each iteration in outer cross-validation's split, the Bayes Optimization will be applied on each inner_cv and the computed accuracy will be appended to the outer_fold_accuracies.
Then after training the model, the accuracy and time is reported with/without using hyperparameter optimization technique.
The whole hyperparameters and their values for preprocessing pipeline are as follows:

Hyperparameter	Ranges	Desc.
preprocessor__pca__n_components	[None, 5, 10, …, 30]	PCA for feature selection
preprocessor__num__numeric__scaler	StandardScaler, PassthroughScaler	Using standard or not using any transformers
preprocessor__num__imputer__strategy	Mean, median, most_frequent	Strategies to handle missing values (Not applicable here)
preprocessor__num__scaler__with_mean	True, False	Standardization by mean
preprocessor__num__scaler__with_std	True, False	Standardization by standard deviation
preprocessing__categoricals__onehot__handle_unknown	Ignore, error	Missing values with unknown categories as ignore or error

The hyperparameters and their values to be searched for whole pipeline in Gradient Boosting Classifier are as follows:

Hyperparameter	Values	Desc.
n_estimators	[50, 100, 200, 300 ,…, 600]	Number of Trees
learning_rate	0.01, 0.1, 0.2, 0.3	It determines the step size at each iteration while moving toward a minimum of a loss function
max_depth	[3, 4, 5, …, 15]	Depth of the tree. stopping criteria that restrict the growth of the tree
min_samples_split	[2, 4, .., 20]	It controls the minimum number of samples required to split a node and the minimum number of samples at the leaf node
min_samples_leaf	[1,2,3,4,…, 20]	It sets the number of samples in the leaf node

Also, the same hyperparameters are used for Random Forest Classifier except for learning_rate which it's not defined. So, in the table below the hyperparameters and their values for Random Forest Classifier has been shown:

Hyperparameter	Values	Desc.
n_estimators	[50, 100, 200, 300 ,…, 600]	Number of Trees
max_depth	[3, 4, 5, …, 15]	Depth of the tree. stopping criteria

		that restrict the growth of the tree
min_samples_split	[2, 4, .., 20]	It controls the minimum number of samples required to split a node and the minimum number of samples at the leaf node
min_samples_leaf	[1,2,3,4,…, 20]	It sets the number of samples in the leaf node

The hyperparameters that has been used for K-Neighbors Classifier are as follows:

Hyperparameter	Values	Desc.
n_neighbors	Range[3, 20]	Number of neighbors
Weights	[uniform, distance]	Uniform: all points in each neighborhood are weighted equally Distance: closer neighbor will have greater influence
Algorithm	[auto, ball_tree, kd_tree, brute]	Different algorithm to search

After going through each of these hyperparameters using Bayesian Optimization with the same number of iteration (n_iter) equal to 100, the best hyperparameters have been printed for Gradient Boosting Classifier as follows:

- Preprocessor_pca_components: None
- Preprocessor_Num_Scaler: StandardScaler
- Preprocessing_Onehot_handle_Unknown: ignore
- preprocessor__num__imputer__strategy: Mean

- preprocessor__num__scaler__with_mean: True
- preprocessor__num__scaler__with_std: False
- Classifier_n_estimators: 200
- Classifier_min_samples_split: 10
- Classifier_min_samples_leaf: 1
- Classifier_max_depth: 5
- Classifier_learning_rate: 0.2

And for Random Forest Classifier we have:

- Preprocessor_pca_components: None
- Preprocessor_Num_Scaler: StandardScaler
- Preprocessing_Onehot_handle_Unknown: ignore
- preprocessor__num__imputer__strategy: Mean
- preprocessor__num__scaler__with_mean: True
- preprocessor__num__scaler__with_std: False
- Classifier_n_estimators: 200
- Classifier_min_samples_split: 4
- Classifier_min_samples_leaf: 1
- Classifier_max_depth: 13

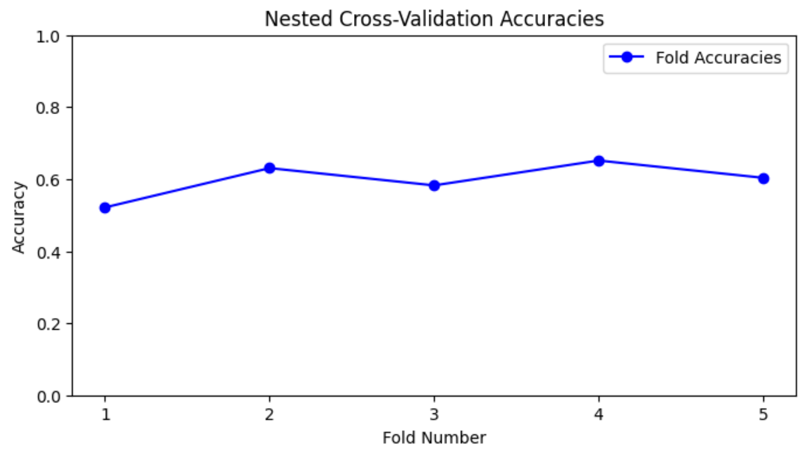
And for K-Neighbors we also have:

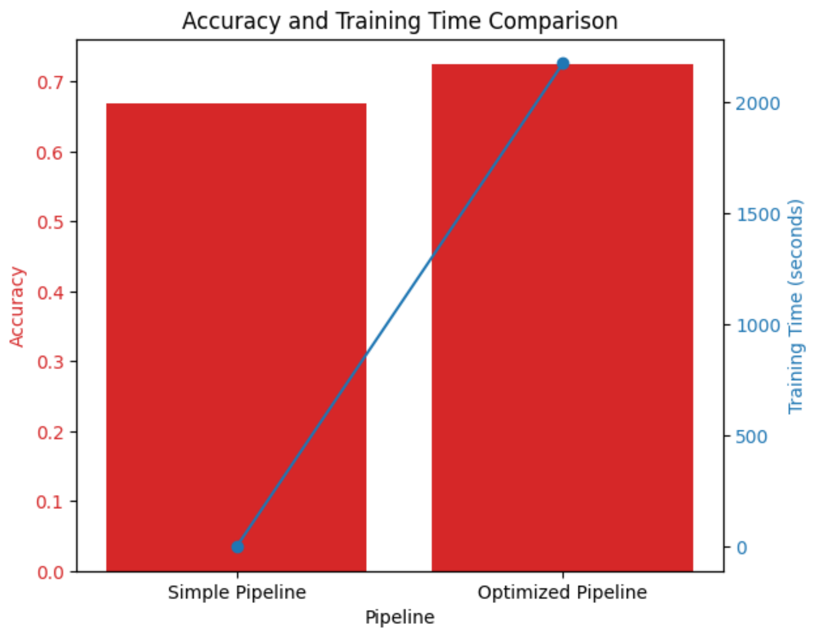
- Preprocessor_pca_components: None
- Preprocessor_Num_Scaler: StandardScaler
- Preprocessing_Onehot_handle_Unknown: ignore
- preprocessor__num__imputer__strategy: Mean
- preprocessor__num__scaler__with_mean: True
- preprocessor__num__scaler__with_std: False
- Classifier_weights: distance
- Classifier_n_neghbors: 9
- Classifier_algorithm: brute

In this experiment, both time and accuracy are computed as comparison metrics.

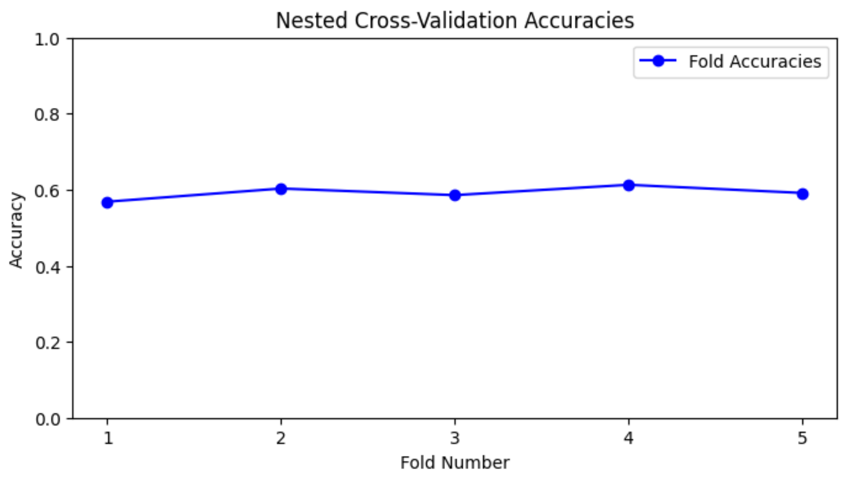
Model	Accuracy (percent)	Time (seconds)
Gradient Boosting Classifier without HPO	57.33%	6.3559
Gradient Boosting Classifier with HPO (Bayesian Optimization)	71.53%	2969.4558
Random Forest Classifier without HPO	68.25%	0.9876
Random Forest Classifier with HPO (Bayesian Optimization)	72.05%	2171.3234
K-Neighbors without HPO	54.22%	0.1171
K-Neighbors With HPO (Bayesian Optimization)	97.01%	17.9433

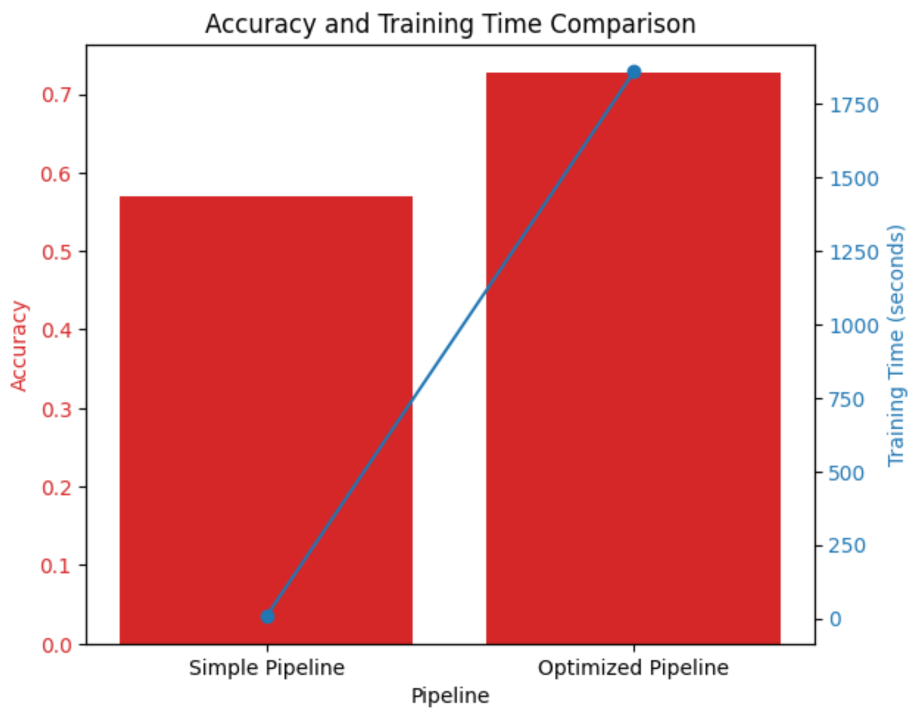
As you can see, hyperparameter optimization over the entire pipeline can lead to better results. First, you take into account the interdependencies and interactions between the preprocessing and modeling stages when you optimize the hyperparameters for the entire pipeline. This lets you identify a set of hyperparameters that function well together. This holistic approach is more likely to result in a well-tuned model that achieves better generalization and predictive accuracy. However, the particular situation, the resources at hand, and the trade-offs between computational cost and performance will determine whether to optimize the pipeline as a whole or its individual components.
In the diagrams below, we have nested cross-validation accuracies and the comparison between the Random Forest Classifier with and without HPO while indicating both accuracy and time.



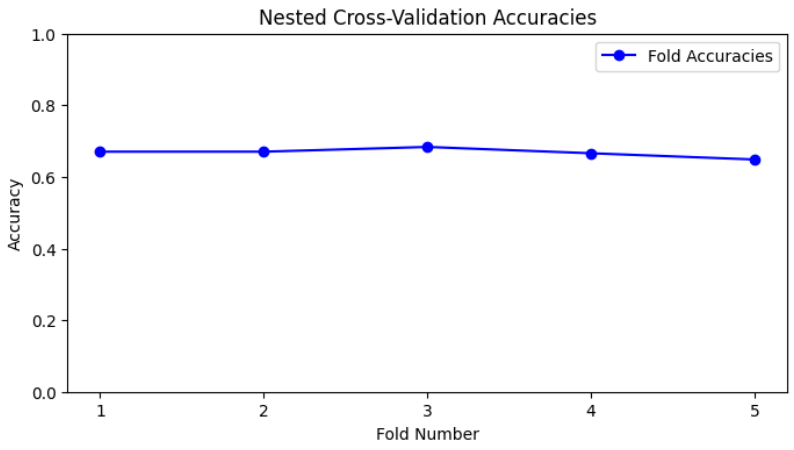


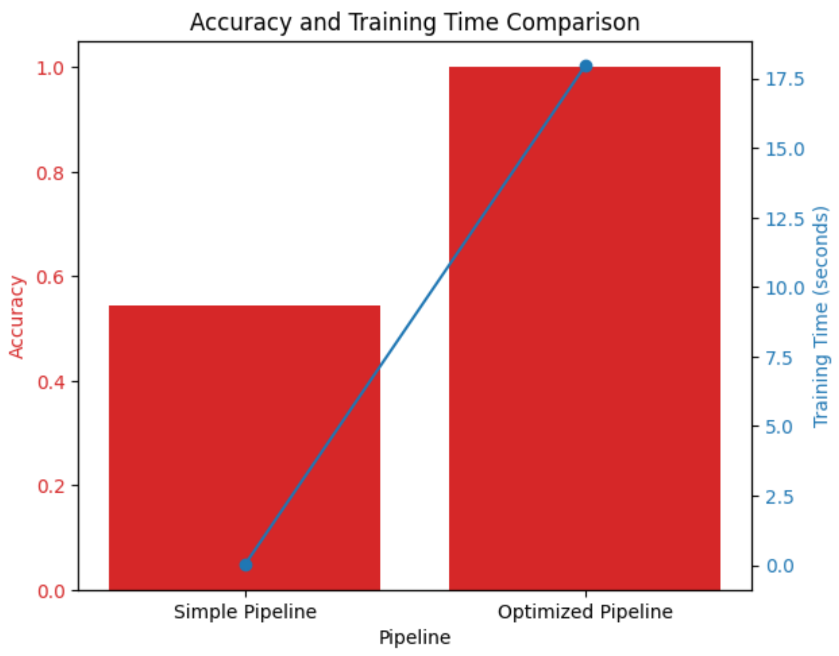
For Gradient Boosting Classifier with HPO comparing to the baseline, we have:





The nested cross-validation accuracies and comparison accuracy and time for K-Neighbor Classifier shown in bellow:





The simple pipeline means the one without hyperparameter optimization, while the optimized pipeline uses Bayesian Optimization as HPO. The left y-axis shows the accuracy scores and the right y-axis show the time takes in seconds. Also, the blue line shows the comparison between the training times that the two approaches take.

References:

1 - https://machinelearningmastery.com/modeling-pipeline-optimization-with-scikit-learn/

2 - https://www.youtube.com/watch?v=TLjMibCN6v4

3 - https://www.youtube.com/watch?v=w9IGkBfOoic

4 -https://towardsdatascience.com/step-by-step-tutorial-of-sci-kit-learn-pipeline-62402d5629b6

5 - https://www.tidymodels.org/learn/work/nested-resampling/#:~:text=Nested%20resampling uses an additional,on the original analysis set.