COSC 4557/5557 Practical Machine Learning Spring 2024

Pipeline Optimization

Submitted by: Iqbal Khatoon

Introduction

In this report, we delve into the optimization of the entire machine learning (ML) pipeline for the 'winequality-red' dataset, rather than focusing solely on individual components. Our approach systematically adjusts hyperparameters across various stages of preprocessing and model training. This comprehensive strategy aims not only to enhance performance but also to gain a thorough understanding of how each component interacts and contributes within the pipeline. By integrating preprocessing steps with model parameter tuning, we seek to elucidate the dynamic interplay of factors that influence predictive accuracy and model robustness.

Wine Quality Dataset

ML algorithm selection exercise employs the "winequality-red" dataset, containing information on different attributes of red wines. These attributes encompass fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, and alcohol content. The dataset consists of eleven features. The target variable, denoted as "quality," assesses the wine's quality on a scale ranging from zero to ten (0-10). Based on our data analysis on the provided data set, we can ascertain that the dataset contains a total of 1599 entries, each with non-null values across all features and the label. This implies that there are no missing values present in the dataset, which is a positive aspect for our analysis. Furthermore, upon inspecting the data types assigned to each column, we observe that all features have been appropriately assigned the 'float64' data type, indicating numerical values. Similarly, the label class 'quality' comprises integer values exclusively, aligning with its assigned 'integer' data type.

Methodology

Data Preparation and Splitting

The dataset utilized in this study is the 'winequality-red' dataset, which includes both physicochemical properties and sensory quality ratings of red wine. Prior to any preprocessing or modeling, the dataset was divided into a training set and a test set. The split was performed with 80% of the data allocated for training and the remaining 20% reserved for testing, ensuring a representative distribution of the samples. This separation facilitates an unbiased evaluation of model performance on unseen data.

Preprocessing Steps

The **preprocessing hyperparameters that we have tuned in preprocessing pipelines** involve the activation states of specific transformers within the preprocessing pipeline. These transformers include logarithmic transformation, scaling, and one-hot encoding for categorical variables. The tunable hyperparameters for these preprocessing methods in your configuration are:

- **Logarithmic Transformation Activation:**
  This hyperparameter determines whether the logarithmic transformation (applied to numerical features) is active or not. The Function Transformer is used to apply a logarithm function to the data, and this can be turned on or off as a part of the hyperparameter tuning process.
- **Scaling Activation:**
  This controls whether scaling (standardization) is applied to the numerical features. The StandardScaler is utilized for this purpose, and its activation can also be toggled in the hyperparameter tuning.

- **One-hot Encoding Activation:**
  This parameter decides whether to apply one-hot encoding to categorical features through the OneHotEncoder. The activation state of this transformer is part of the hyperparameter tuning.

Feature Selection

Feature selection was conducted using the ANOVA F-test to retain the top 10 most significant features. This approach reduces the dimensionality of the data, which can decrease overfitting and improve model performance.

Model Training and Hyperparameter Tuning

The study evaluated four different classifiers:

- Random Forest
- Gradient Boosting
- KNN
- AdaBoost

Each model was subject to thorough hyperparameter tuning using two distinct strategies:

- **Randomized Search CV:** This method randomly selects combinations of parameters, providing a broad exploration of the parameter space. It offers a balance between exploration and computational efficiency.
- **Bayesian Search CV:** A more targeted approach, Bayesian Search CV uses past evaluation results to choose the next set of parameters to evaluate. This method is expected to find better parameters in fewer steps than random search.

The hyperparameters for each classifier were varied within specified ranges:

- **Random Forest:** Number of estimators, max depth, and min samples per leaf.
- **Gradient Boosting:** Number of estimators, learning rate, max depth, and min samples per leaf.
- **K-Nearest Neighbors (KNN):** Number of neighbors, weights, and algorithm used for nearest neighbors.
- **AdaBoost:** Number of estimators and learning rate.

Evaluation Metrics

The effectiveness of each model configuration was assessed using cross-validated accuracy on the training set and accuracy on the test set. This dual approach ensures a robust evaluation, emphasizing both the model's ability to generalize to new data and its performance consistency across multiple subsets of the training data.

Results

Hyperparameter Ranges and Optimized Values

The methods mentioned in the methodology steps efficiently navigate through an extensive parameter space and examine various configurations to pinpoint the most effective ones. The hyperparameters of the classifiers, their respective ranges, and the optimal values derived from both optimization strategies are detailed in Table 1.

Model	Hyperparameter	Range/Search Space	Random Search Best Value	Bayesian Search Best Value
RandomForest	n_estimators	100-1000 (Integer)	838	1000
	max_depth	10-100 (Integer)	44	100
	min_samples_leaf	1-20 (Integer)	1	1
	log_transform_active	True/False (Categorical)	TRUE	TRUE
	scaler_active	True/False (Categorical)	FALSE	FALSE
	onehot_active	True/False (Categorical)	TRUE	TRUE
GradientBoosting	n_estimators	50-300 (Integer)	135	245
	learning_rate	0.01-0.2 (Real)	0.0506	0.1912
	max_depth	1-10 (Integer)	7	7
	min_samples_leaf	1-10 (Integer)	3	1
	log_transform_active	True/False (Categorical)	FALSE	FALSE
	scaler_active	True/False (Categorical)	FALSE	FALSE
	onehot_active	True/False (Categorical)	FALSE	FALSE
KNN	n_neighbors	1-30 (Integer)	15	21
	weights	'uniform', 'distance' (Categorical)	'distance'	'distance'
	algorithm	'auto', 'ball_tree', 'kd_tree', 'brute' (Categorical)	'ball_tree'	'brute'
	log_transform_active	True/False (Categorical)	FALSE	TRUE
	scaler_active	True/False (Categorical)	TRUE	TRUE
	onehot_active	True/False (Categorical)	TRUE	TRUE
AdaBoost	n_estimators	50-100 (Integer)	72	50
	learning_rate	0.1-1.0 (Real)	0.1636	0.1273
	log_transform_active	True/False (Categorical)	FALSE	TRUE
	scaler_active	True/False (Categorical)	TRUE	FALSE
	onehot_active	True/False (Categorical)	FALSE	FALSE

Table 1. Hyperparameter ranges and optimized values for all four classifiers

Extensive testing and optimization were conducted on machine learning pipelines using the 'winequality-red' dataset, employing four different classifiers. The results shown in Figure 1 depict varied improvements across different optimization techniques and classifiers. Random Forest displayed robust performance with a Bayesian Search yielding the highest cross-validation accuracy of 69.28% and a Random Search best test accuracy of 65.94%. Gradient Boosting followed with a notable improvement from a default test accuracy of 61.56% to a Random Search test accuracy of 64.38%, and Bayesian optimizations slightly trailing at 63.12%. KNN, while starting with lower accuracies (56.87% test accuracy on default settings), showed significant gains with Bayesian Search reaching up to 66.87% on test accuracy. AdaBoost, however, had lesser improvements, with the highest test accuracy reaching only 53.75% through Bayesian Search, demonstrating that not all models reacted similarly to the optimization processes. These variations emphasize the importance of tailored hyperparameter optimization, which not only enhances model performance but also provides insights into the effective integration of preprocessing and model parameter tuning.

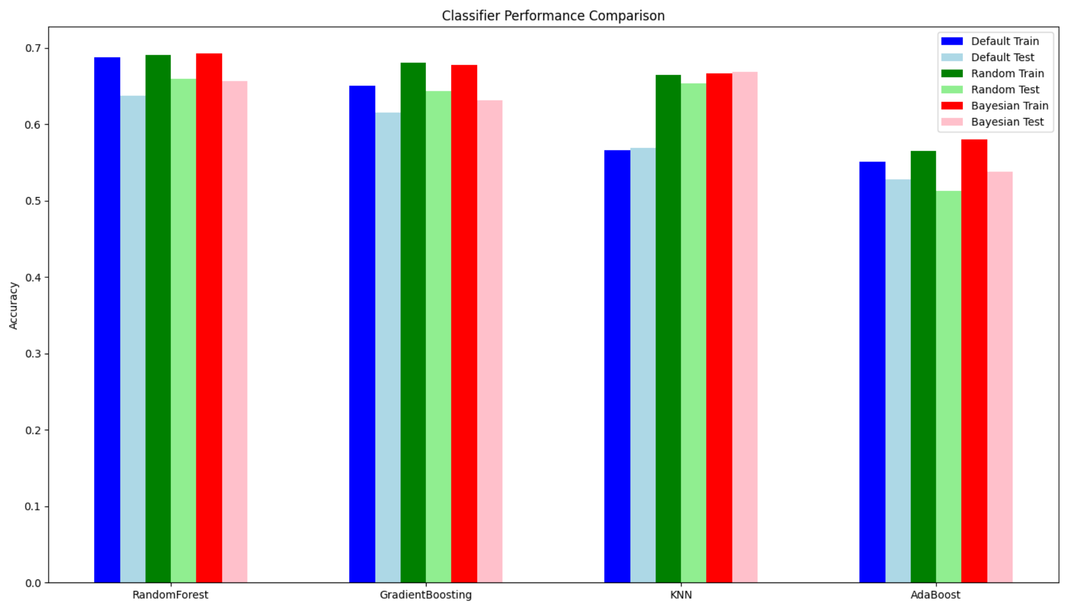

Figure 1: Before and after optimization accuracy comparison of all four classifiers.

Table 2 clearly illustrates the improvement trajectory from default configurations through to more advanced hyperparameter tuning methods, providing a quantitative comparison of how each classifier responds to optimization. This tabulation helps in understanding the effectiveness of hyperparameter optimization techniques on enhancing the predictive accuracy of the models across the 'winequality-red' dataset. It succinctly displays how each classifier's performance varies with different optimization strategies, illustrating the change in accuracy that can be achieved through systematic tuning.

Classifier	Metric	Default Accuracy	Random Search Accuracy	Bayesian Search Accuracy
RandomForest	Test Accuracy	63.75%	65.94%	65.62%
	Cross-validated Accuracy	68.73%	69.04%	69.28%
GradientBoosting	Test Accuracy	61.56%	64.38%	63.12%
	Cross-validated Accuracy	65.05%	68.02%	67.71%
KNN	Test Accuracy	56.87%	65.31%	66.87%
	Cross-validated Accuracy	56.61%	66.46%	66.62%
AdaBoost	Test Accuracy	52.81%	51.25%	53.75%
	Cross-validated Accuracy	55.05%	56.53%	58.02%

Table 2: The test and CV accuracies before and after optimization of all four classifiers.

Conclusion

This study highlights the importance of comprehensive optimization across the machine learning pipeline. Our results indicate that the effectiveness of hyperparameter optimization techniques can vary significantly between different models, with Random Search performing well for Random Forest and Bayesian optimization excelling with KNN. By adjusting both preprocessing and model parameters in a unified manner, the research sheds light on the synergistic effects within the ML pipeline, enhancing our understanding and the performance of predictive models. This integrated approach ensures that each component is optimized, leading to more effective and reliable machine learning outcomes.

References

[1] Wine Quality - UCI Machine Learning Repository
[2] **https://scikit-learn.org/stable/**
[3]**https://www.superannotate.com/blog/how-to-optimize-machine-learning-pipeline**
[4] Farshad’s and Iran’s GitHub repository for Pipeline Optimization