Pipeline Optimization Report

Soudabeh Bolouri

Introduction:

This exercise involves a comprehensive approach to predicting wine quality utilizing multiple ML algorithms including SVM, Random Forest, and KNN. This experiment optimizes the entire ML approach, including any preprocessing, rather than optimizing just one component. Using nested cross-validation, we were able to determine which model generalized the data the best. The goal was to assess the impact of hyperparameter tuning on the performance of three different ML classifiers. The experiment involved comparing the default accuracy of these models with their post-tuning accuracy.

Dataset Description:

The dataset that we utilized in this exercise is the "Wine Quality" dataset, precisely the "white" wine version. It contains data on different features of white wines, including fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, PH, sulphates, and alcohol. The dataset includes a totality of 4,898 rows and 12 features. The "quality" of the wine is the target variable, which we desire to predict. Also, we did not find any missing values in the dataset.

Experimental Setup:

We used the Python programming language for this exercise. First, we need to import our dataset "winequality-white.csv" using Panda. In the next step, the dataset is preprocessed to ensure its suitability for model training. Missing values are replaced with the mean of the column via an imputer (this is a general case for implementing pipelines even though there are no missing values for this specific dataset). As a way of standardizing features, it uses the Standard Scaler to remove the mean and scale to unit variance. Then feature scaling and one-hot encoding are performed using **StandardScaler** and **OneHotEncoder** respectively.

Four classifiers are assessed in this analysis:

- Support Vector Machine (SVM)
- Random Forest
- K-Nearest Neighbors (KNN)

The hyperparameters and their values to be searched for each model are as follows:

SVM:

Hyperparameter	**Values**	**Description**
preprocessor__num__imputer__strategy	['median']	it specifies the imputation strategy for numeric features (like filling missing values with the mean, median, etc.)
preprocessor__cat__imputer__strategy	['most_frequent']	it specifies the imputation strategy for categorical features (like filling missing values with the most frequent value, a constant value, etc.)
classifier__C	[1, 10, 100]	This parameter controls the trade-off between maximizing the margin and minimizing classification error.
classifier__gamma	[0.01, 0.1, ‘scale’]	This parameter defines the influence of a single training example's distance, affecting the reach of the kernel
classifier__kernel	['rbf']	Kernel specifies the type of kernel used in the SVM algorithm, determining the decision boundary shape in the feature space.

Random Forest:

Hyperparameter	**Values**	**Description**
preprocessor__num__imputer__strategy	['mean', 'median']	it specifies the imputation strategy for numeric features (like filling missing values with the mean, median, etc.)
preprocessor__cat__imputer__strategy	['most_frequent', 'constant']	it specifies the imputation strategy for categorical features (like filling missing values with the most frequent value, a constant value, etc.)
classifier__n_estimators	[100, 300, 500, 800, 1200]	number of trees in the random for est
classifier__max_features	['auto', 'sqrt', ‘log2’]	number of features in consideration at every split
classifier__max_depth	[5, 10,15, 20, 25, 30, None]	maximum number of levels allow ed in each decision tree

classifier__min_samples_split	[2, 5, 10, 15, 20]	minimum sample number to split a node
classifier__min_samples_leaf	[1, 2, 4, 6, 8]	minimum sample number that can be stored in a leaf node
classifier__bootstrap	[True, False]	method used to sample data points
classifier__criterion	['gini', 'entropy']	the function used to measure the quality of a split

KNN:

Hyperparameter	**Values**	**Description**
preprocessor__num__imputer__strategy	['mean', 'median']	it specifies the imputation strategy for numeric features (like filling missing values with the mean, median, etc.)
preprocessor__cat__imputer__strategy	['most_frequent', 'constant']	it specifies the imputation strategy for categorical features (like filling missing values with the most frequent value, a constant value, etc.)
classifier__n_neighbors	np.arange(2, 30, 1)	Number of neighbors to consider. It defines the value of 'k' in the K NN algorithm.
classifier__weights	['uniform', 'distance']	The weight function used in prediction

The hyperparameters for preprocessing steps (**imputer** strategies for both numerical and categorical features) were also included in the tuning process. In the preprocessing pipeline, I tuned the following hyperparameters:

- Numeric Features: Imputation strategy ('mean' and 'median').
- Categorical Features: Imputation strategy ('most_frequent' and 'constant').

Nested cross-validation was implemented with an outer 5-fold Stratified K-Fold and an inner 3-fold Stratified K-Fold. The inner loop involved hyperparameter tuning using **RandomizedSearchCV** with 10 iterations, optimizing the entire pipeline, including preprocessing and the classifier. The outer loop assessed the optimized model's performance.

Results:

The following accuracies were observed before and after tuning:

Random Forest:

- Default Accuracy: 0.6784
- Tuned Accuracy: 0.6888
- Tuned Parameters:
    - 'preprocessor__num__imputer__strategy': 'mean',
    - 'preprocessor__cat__imputer__strategy': 'constant',
    - 'classifier__n_estimators': 1200,
    - 'classifier__min_samples_split': 10,
    - 'classifier__min_samples_leaf': 2,
    - 'classifier__max_features': 'sqrt',
    - 'classifier__max_depth': 25,
    - 'classifier__criterion': 'entropy',
    - 'classifier__bootstrap': False

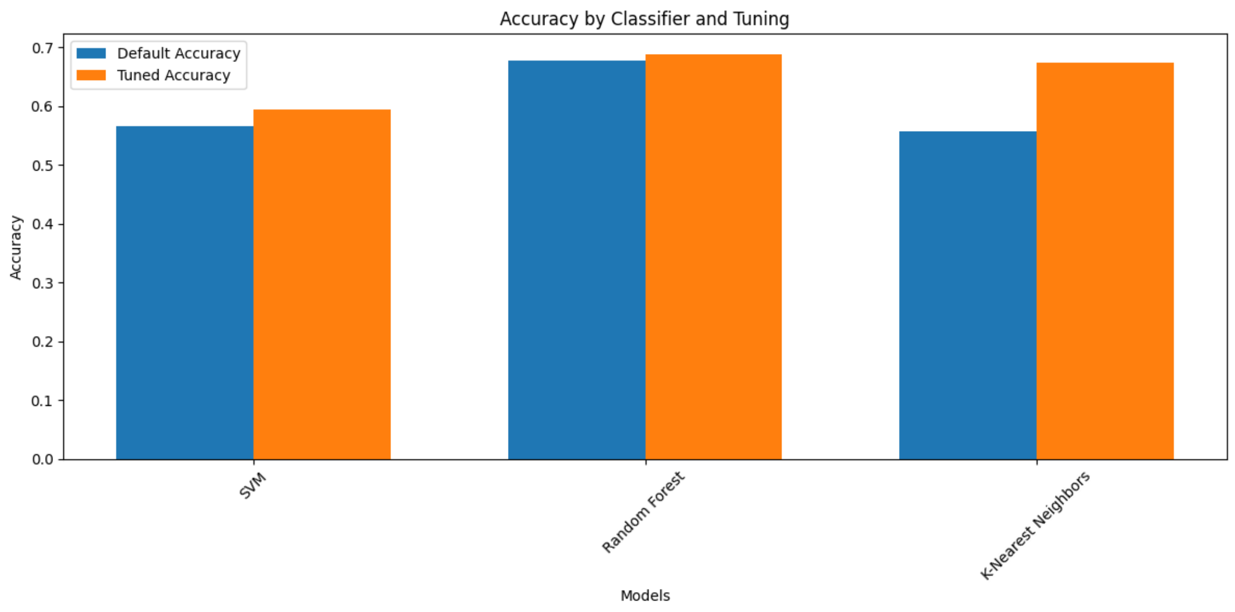
K-Nearest Neighbors (KNN):

- Default Accuracy: 0.5570
- Tuned Accuracy: 0.6744
- Tuned Parameters:
    - 'preprocessor__num__imputer__strategy': 'median',
    - 'preprocessor__cat__imputer__strategy': 'most_frequent',
    - 'classifier__weights': 'distance',
    - 'classifier__n_neighbors': 29

SVM:

- Default Accuracy: 0.5666
- Tuned Accuracy: 0.5945
- Tuned Parameters:
    - 'preprocessor__num__imputer__strategy': 'median',
    - 'preprocessor__cat__imputer__strategy': 'most_frequent',
    - 'classifier__kernel': 'rbf',
    - 'classifier__gamma': 0.1,
    - 'classifier__C': 10

Based on the bellow bar chart showcasing the Comparison of Model Performance Before and After Hyperparameter Tuning, we can interpret the results as follows:



Tuning does not significantly improve the performance of the Random Forest model. With its default hyperparameters, the Random Forest model already achieves a high degree of accuracy (0.6784), which indicates it is quite robust from the beginning. After tuning, the accuracy slightly increases to 0.6888.

By tuning hyperparameters, KNN shows an impressive improvement; the default accuracy of 0.5570 is significantly increased to 0.6744 after tuning. A marked improvement in model performance may have been attributed to the optimization of parameters like neighbor number and weighting method.

For the SVM model, the tuning process results in a moderate increase in accuracy, from 0.5666 to 0.5945.

References:

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

[2] https://medium.com/mlearning-ai/how-to-use-sklearns-pipelines-to-optimize-your-analysis-b6cd91999be

[3] https://scikit-learn.org/stable/modules/grid_search.html

[4] https://towardsdatascience.com/hyper-parameter-tuning-and-model-selection-like-a-movie-star-a884b8ee8d68

[5] https://towardsdatascience.com/nested-cross-validation-hyperparameter-optimization-and-model-selection-5885d84acda

[6] https://machinelearningmastery.com/nested-cross-validation-for-machine-learning-with-python/

[7] https://www.kaggle.com/code/arjunprasadsarkhel/simple-random-forest-with-hyperparameter-tuning

[8] https://www.kaggle.com/code/funxexcel/p2-logistic-regression-hyperparameter-tuning

[9] https://medium.com/@agrawalsam1997/hyperparameter-tuning-of-knn-classifier-a32f31af25c7