Pipeline Optimization Report

Soudabeh Bolouri

Introduction:

This exercise involves a comprehensive approach to predicting wine quality utilizing multiple ML algorithms including SVM, Random Forest, KNN, and Logistic Regression. 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.

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. The dataset is then split into training and testing sets with an 80-20 ratio using the **train_test_split** function from Scikit-Learn. 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
- Logistic Regression
- K-Nearest Neighbors (KNN)

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

SVM:

Hyperparameter	Values	Description
classifier__C	[0.001, 0.01, 10, 100]	This parameter controls the trade-off between maximizing the margin and minimizing classification error.
classifier__gamma	[0.1, 0.01, 0.001, 0.0001]	This parameter defines the influence of a single training example's distance, affecting the reach of the kernel
classifier__kernel	['linear', 'rbf', 'poly']	Kernel specifies the type of kernel used in the SVM algorithm, determining the decision boundary shape in the feature space.Three kernel options are considered here: **'linear'**, **'rbf'** (Gaussian kernel), and **'poly'** (polynomial kernel).

Random Forest:

Hyperparameter	Values	Description
classifier__n_estimators	[5, 20, 50, 100]	number of trees in the rand om forest
classifier__max_features	['auto', 'sqrt']	number of features in consideration at every split
classifier__max_depth	[int(x) for x in np.linspace(10, 120, num = 12)]	maximum number of levels allowed in each decision tre e
classifier__min_samples_split	[2, 6, 10]	minimum sample number t o split a node
classifier__min_samples_leaf	[1, 3, 4]	minimum sample number that can be stored in a leaf node
classifier__bootstrap	[True, False]	method used to sample data points

Logistic Regression:

Hyperparameter	Values	Description
classifier__penalty	['l1', 'l2', 'elasticnet', 'none']	Regularization term, specifying the nor m used in the penalization
classifier__C	np.logspace(-4, 4, 20)	Inverse of regularization strength, a positive float value. It controls the regularization parameter, where smaller values specify stronger regularization
classifier__solver	['lbfgs','newton-cg','liblinear' ,'sag','saga']	Algorithm to use in the optimization pr oblem
classifier__max_iter	[100, 1000,2500, 5000]	Maximum number of iterations for the solvers to converge

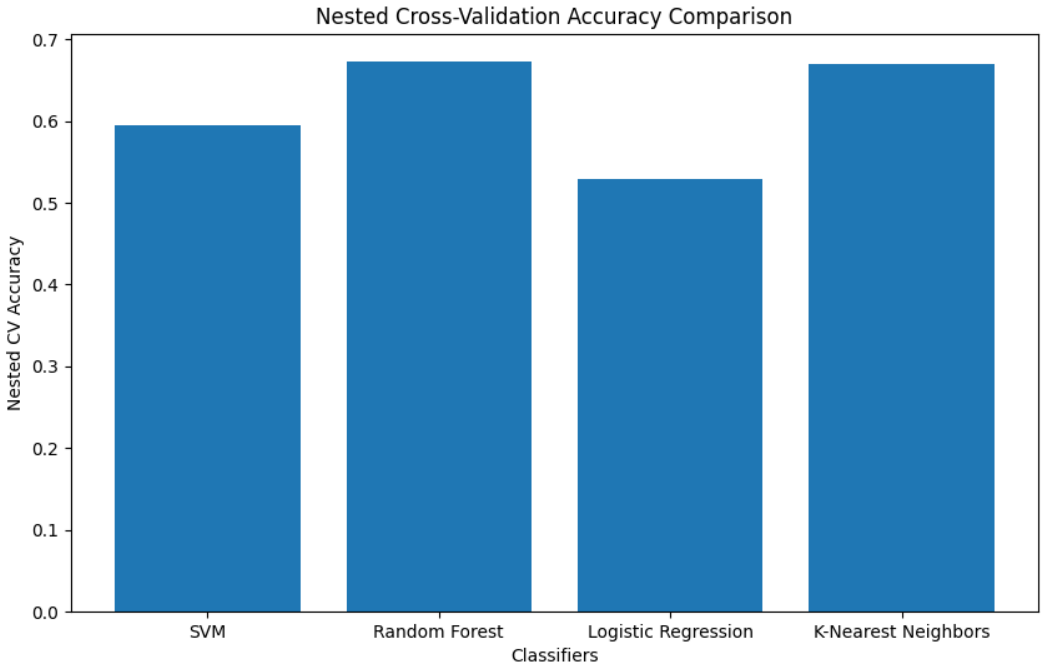
KNN:

Hyperparameter	Values	Description
classifier__n_neighbors	np.arange(2, 30, 1)	Number of neighbors to consider. It de fines the value of 'k' in the KNN algori thm.
classifier__weights	['uniform', 'distance']	The weight function used in prediction

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:

Based on the bellow bar chart showcasing the Nested Cross-Validation Accuracy Comparison, we can interpret the results as follows:



As a result, both **Random Forest** and **KNN** performed reasonably well; Random Forest, in particular, benefits from its ability to handle complex interactions in the dataset, while KNN performed well on a well-segmented dataset.

The accuracy of the SVM was moderate; hyperparameters and kernel used might have influenced its performance; considering the dataset's complexity, tuning the regularization and kernel parameters might have been worthwhile.

Logistic regression had the lowest accuracy, which might be due to its linear nature being less suited to capture complex relationships in the data compared to other models.

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