Introduction: For this exercise, I took a look at the Wine Quality data set to perform hyperparameter optimization to get a better understanding of hyperparameter tuning process. Optimizing hyperparameters can help improve the model performance and produce more optimal results.

Data Set Description: For this analysis, I used the Red Wine Quality Data. Here, In this data set, wine quality is measured by twelve features including fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol, and quality. For each of these features, the number of observations was 1599. Regarding preprocessing, I checked for null values using a isnull().values.any() function. There were not any null values. In this exercise, hyperparameter optimization was performed to improve the predicting model. The feature chosen to predict was quality.

Experimental Setup: To work with hyperparameter optimization, two ML algorithms were chosen to optimize: Random Forest and SVC.

Beginning with Random forest, the pandas package was imported to aid in data analysis and importation of the dataset. Further, the sklearn package was used to utilize machine learning models such as Random Forest Regressor and SVC. Additionally, the sklearn package was used to implement randomly splitting the Red Wine Quality dataset into train and test subsets. For hyperparameter optimization, Bayesian Optimization was utilized from the package bayes_opt. For each Bayesian Optimization, 35 iterations and 5 steps of random variations (init_points) was ran. The function to be optimized was designated as "f" with boundaries represented as the parameter boundaries.

Once packages were loaded, the Red Wine Quality Data set was downloaded via the pandas package and checked for null values using the isnull().values.any() function. Since there were none, the input features (x) and output targets (y) were defined accordingly. Next, the dataset was split into training and testing data using the train_test_split function from sklearn. Here, I chose the test size to be 20% and the train size to be 80%.

Before optimization, I ran the Random Forest Regression with the default parameters. Once the data was loaded, I continued with Bayesian Optimization. First, I did not know what hyperparameters were included in the Random Forest algorithm so I used a get_param_names() function to look at hyperparameters. Then, to further understand what each hyperparameter measured, I went to the sklearn information page about Random Forest before picking the hyperparameters to tune. From the sklearn library page, I chose the hyperparameters with integer values. Once I looked at these, I chose to optimize n_estimators(number of trees), max_depth(tree depth maximum), min_samples_split (number of samples required for splitting), max_features(number of features required for the best split), max_leaf_nodes(tree leaf node maximum), max_samples(maximum number of samples), min_impurity_decrease(threshold for when nodes are split), min_samples_leaf(minimum samples needed to form a node), min_weight_fraction_leaf(minimum weight calculated required to be a node), n_jobs(number of jobs running in parallel), and verbose(controls expressivity during fitting and predicting) . Next, I defined an objective function using the parameters I chose from the list I generated. When defining my objective function, I defined my model to be Random Forest Regressor and listed all of the hyperparameters as integer values. Then, I added a line to return the negative mean cross validation score. Here, nested resampling was setup using the cross_val_score function where the number of folds was set to 7. The estimator was set to be the Random Forest Regressor model and the X and y features/targets were set to be the x_train and y_train data respectively. This was the score I was trying to improve upon.

Next for Random Forest, I defined the parameter bounds and then ran the Bayesian Optimization. When optimization was complete, I returned the best hyperparameters by using the opt.max['params'] function. Then, I went on to train the final model using the hyperparameters that were optimized using Bayesian optimization. The model was fit and a R^2 Test score was generated.

Next, I ran a Bayesian Optimization on a SVC algorithm. For this portion, the sklearn package was utilized for importation of SVC. The pandas package was used for data importation and the bayes_opt was used to run Bayesian Optimization.

Just as in Random Forest Regressor, data was downloaded using pandas and the input and output were set accordingly. The data was then split into train and test data subsets using the train_test_split function.

Then, I defined an objective function and set the model to be SVC. Before I listed hyperparameters I used get_param_names() function to look at hyperparameters. Then, I included every hyperparameter in my objective function and ended with a return of the negative mean cross validation score. The hyperparameters initially included were C(regularization), kernel(kernel type), degree(degree of kernel polynomial), gamma(kernel coefficient), coef0(independent kernel variable), shrinking (determines whether to use shrinking heuristics), probability (whether to use probability estimates), tol (tolerance for stopping criterion), cache_size(size of kernel cache), class_weight(weight of C for kernel with class i), verbose (enables expressivity), max_iter(maximum iterations), decision_function_shape(returns one v. rest or one v. one), and break_ties (breaks ties in accordance to decision function confidence values). Here, nested resampling was setup using the cross_val_score function where the number of folds was set to 7. The estimator was set to be the SVC model and the X and y features/targets were set to be the x_train and y_train data respectively. This was the score I was trying to improve upon.

Next for SVC, I ran the SVC model with no optimization of hyperparameters. Here, data was standardized using StandardScaler. I defined the parameter bounds and then ran the Bayesian Optimization. When optimization was complete, I returned the best hyperparameters by using the opt.max['params'] function. Then, I went on to train the final model using the hyperparameters that were optimized using Bayesian optimization. The model was fit and a R^2 test score was generated.

Results: Beginning with Random Forest this is what I observed. As I was using Bayesian Optimization, many parameters came back with an error either explaining that the hyperparameter had to be boolean or fall within a specific range of numbers. When I observed the ”needs to be boolean” argument, I eliminated this hyperparameter from the optimization. When I observed the ”needs to fall in this range” argument, I adjusted the parameter boundaries. In the end, the hyperparameters I optimized included n_estimators (int range of $[1,\infty)$), max_depth (int range of $[1,\infty)$), min_samples_split(range of $[2,\infty)$), max_features (int range of $[1,\infty)$, float range of $(0,1]$, or str among log2, sqrt), max_leaf_nodes(int range of $[2,\infty)$), max_samples(float in the range of $(0,1]$ or int in range $[1,\infty)$), min_decrease(float range of $[0,\infty)$), min_samples_leaf(int range of $[1,\infty)$ and float range $(0,1)$), and min_weight_fraction_leaf (float range $[0,0.5]$). Before optimization, the R^2 value was 0.900601162250618 and the mean squared error was 0.0647842401500937 (Table 1). Once optimized, the final R^2 score was -0.00159927923320313 and mean squared error was 0.6528028869272 for Random Forest (Table 2).

For SVC, many parameters came back with an error either explaining that the hyperparameter had to be boolean or fall within a specific range of numbers. This resulted in C(float range of $(0,\infty)$), cache_size(float range of $(0,\infty)$), coef0 (range of $(-\infty, \infty)$), and gamma (float range of $[0,\infty]$) being tuned. Before optimization, the mean accuracy score was 0.671669793621013 (Table 3). Once optimized, the final mean accuracy score was 0.908692933083177 for SVC (Table 4).

Of these two models after Bayesian Optimization, SVC appears to work better for the Red Wine Quality Data set due to greater improvement in the mean accuracy score. The negative R^2 test score for Random Forest implies the fit was not very good and or did not fit the shape of the data well.

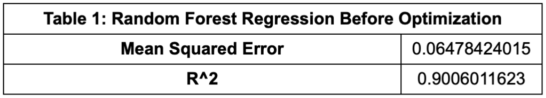

Table 1: Random Forest Regression Before Optimization	
Mean Squared Error	0.06478424015
R^2	0.9006011623

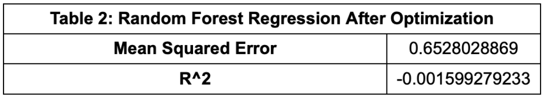

Table 2: Random Forest Regression After Optimization	
Mean Squared Error	0.6528028869
R^2	-0.001599279233

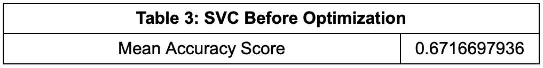

Table 3: SVC Before Optimization	
Mean Accuracy Score	0.6716697936

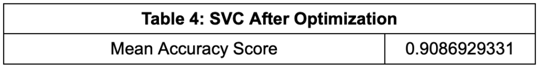

Table 4: SVC After Optimization	
Mean Accuracy Score	0.9086929331

References:

1. Lee, Dr. Ernesto. "Step-by-Step Guide: Bayesian Optimization with Random Forest." Medium, Medium, 31 Aug. 2023, drlee.io/step-by-step-guide-bayesian-optimization-with-random-forest-fdc6f329db9c.

2. "Sklearn.Ensemble.Randomforestregressor." Scikit, scikit-learn.org/stable/modules/generated/sklearn .ensemble.RandomForestRegressor.html. Accessed 25 Mar. 2024.

3. "Sklearn.Svm.SVC." Scikit, scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. Accessed 25 Mar. 2024.

4. "Get_all_params - Catboostregressor." CatBoost, catboost.ai/en/docs/concepts/python-reference _catboostregressor_get_all_params. Accessed 25 Mar. 2024.

5. GfG. "Random Forest Regression in Python." GeeksforGeeks, GeeksforGeeks, 6 Dec. 2023, www.geeksforgeeks.org/random-forest-regression-in-python/.