Hyperparameter Optimization

Mohammad Irfan Uddin

Introduction:

The objective of this report is to explore Hyperparameter Optimization (HPO) with the goal of improving the predictive performance of machine learning models. To accomplish this, we apply different machine learning algorithms and conduct a meticulous tuning of their hyperparameters to identify the most accurate predictive models.

Dataset Description:

The dataset under consideration, the Wine Quality Dataset, is composed of 11 features and contains 1599 samples. The target variable is wine quality, which is a categorical attribute. One notable aspect of this dataset is the absence of missing values, simplifying the preprocessing phase. From dataset, we establish a clear understanding of the dataset's characteristics and set the stage for subsequent model selection.

Experimental Setup:

The practical implementation of the HPO approach begins with the selection of programming languages and libraries. In this case, Python is the chosen language, and we utilize essential libraries such as Pandas for data manipulation, Scikit-learn for machine learning, SciPy for statistical analysis, GridSearchCV for finding the optimal parameter values from a given set of parameters in a grid and Warnings for managing alerts. The dataset is loaded and subsequently divided into training and testing sets through an 80-20 split.

Four machine learning algorithms are considered for evaluation: Random Forest, Support Vector Machine (SVM), Logistic Regression, and Decision Tree. To assess their performance, we employ a 5-fold cross-validation strategy, with the primary evaluation metric being accuracy. Furthermore, statistical tests are used to compare the performance of these models to select the most suitable algorithms for further analysis.

Results:

The results of the model evaluation are summarized as follows:

Model	Accuracy		
	Before_HPO	Bayesian_HPO	GridSearch_HPO
Random Forest	0.68	0.69	0.73
SVM	0.5	0.6	0.64
Logistic Regression	0.62	0.62	0.63
Decision Tree	0.64	0.64	0.66

SVM - Tuned Hyperparameters: ([('C', 50603.554533845774), ('kernel', 'rbf')])

Decision Tree - Tuned Hyperparameters: ([('max_depth', 24), ('min_samples_leaf', 2), ('min_samples_split', 11)])

Random Forest - Tuned Hyperparameters: ([('max_depth', 21), ('min_samples_leaf', 2), ('min_samples_split', 4), ('n_estimators', 46)])

For three out of the four models, the accuracy has increased whereas the accuracy for logistic regression has remained almost same. The accuracy for SVM has increased by more than 26% after HPO.

Code:

```
!pip install scikit-optimize
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from skopt import BayesSearchCV
from skopt.space import Real, Categorical, Integer

data = pd.read_csv('wineq.csv')
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

# Outer loop for nested cross-validation
outer_cv = 8
inner_cv = 8
outer_scores = {}

# Algorithm Selection
models = [
    ('SVM', SVC()),
    ('Random Forest', RandomForestClassifier()),
    ('Logistic Regression', LogisticRegression()),
    ('Decision Tree', DecisionTreeClassifier())
]

# Hyperparameter Optimization using Bayesian Optimization
```

```
param_grids = {
    'SVM': {
         'C': Real(1e-5, 1e+5, prior='log-uniform'),
        'kernel': Categorical(['linear', 'rbf', 'poly', 'sigmoid'])
    },
    'Random Forest': {
        'n_estimators': Integer(10, 200),
        'max_depth': Integer(10, 30),
        'min_samples_split': Integer(1, 10),
        'min_samples_leaf': Integer(1, 10)
    },
    'Logistic Regression': {
        'C': Real(1e-6, 1e+6, prior='log-uniform'),
        'penalty': Categorical(['l1', 'l2']),
        'solver': Categorical(['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'])
    },
    'Decision Tree': {
        'max_depth': Integer(1, 100),
        'min_samples_split': Integer(2, 15),
        'min_samples_leaf': Integer(1, 7)
    }
}


for name, model in models:
    try:
        outer_scores[name] = []
        for outer_train_index, outer_test_index in StratifiedKFold(n_splits=outer_cv, shuffle=True,
random_state=0).split(X, y):
            X_outer_train, X_outer_test = X[outer_train_index], X[outer_test_index]
            y_outer_train, y_outer_test = y[outer_train_index], y[outer_test_index]
            # Inner loop for hyperparameter tuning
            opt = BayesSearchCV(
                model,
                param_grids[name],
                n_iter=10,
                cv=StratifiedKFold(n_splits=inner_cv, shuffle=True, random_state=0),
                scoring='accuracy',
                n_jobs=-1,
                random_state=0
            )
            opt.fit(X_outer_train, y_outer_train)
            best_model = opt.best_estimator_

            # Evaluate on the outer test set
            accuracy = best_model.score(X_outer_test, y_outer_test)
            outer_scores[name].append(accuracy)

            print(f"{name} - Tuned Hyperparameters: {opt.best_params_}, Accuracy: {accuracy}")

    except Exception as e:
        print(f"Bayesian Optimization for {name} raised an exception: {e}")

# Display results
for name, scores in outer_scores.items():
    mean_accuracy = np.nanmean(scores)  # Handling NaN values
    print(f"{name} Outer CV Mean Accuracy: {mean_accuracy}")

best_outer_model_name = max(outer_scores, key=outer_scores.get)
```

```
print("Best Outer Model:", best_outer_model_name,
np.mean(outer_scores[best_outer_model_name]))



from sklearn.model_selection import GridSearchCV

import warnings

from sklearn.exceptions import FitFailedWarning

data = pd.read_csv('wineq.csv')

X = data.iloc[:, :-1].values

y = data.iloc[:, -1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)



warnings.filterwarnings("ignore", category=FitFailedWarning)

# Hyperparameter Optimization for Logistic Regression

logreg_param_grid = {

    'C': [0.1, 0.5, 1, 5, 10],

    'penalty': ['15', 'l1', 'l2'],

    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']

}


logreg_optimized = GridSearchCV(LogisticRegression(max_iter=100), param_grid=logreg_param_grid,
cv=5, scoring='accuracy')

logreg_optimized.fit(X_train, y_train)

best_logreg_model = logreg_optimized.best_estimator_

logreg_accuracy = best_logreg_model.score(X_test, y_test)


# Hyperparameter Optimization for Decision Tree

dt_param_grid = {

    'max_depth': [None, 10, 20, 30, 50,100],

    'min_samples_split': [2, 5, 10, 15],
    'min_samples_leaf': [1, 2, 4, 7]

}


dt_optimized = GridSearchCV(DecisionTreeClassifier(), param_grid=dt_param_grid, cv=5,
scoring='accuracy')

dt_optimized.fit(X_train, y_train)

best_dt_model = dt_optimized.best_estimator_

dt_accuracy = best_dt_model.score(X_test, y_test)

# Hyperparameter Optimization for Random Forest

rf_param_grid = {

    'n_estimators': [10, 50, 100, 200],

    'max_depth': [None, 10, 20, 30],

    'min_samples_split': [2, 5, 10],

    'min_samples_leaf': [1, 2, 4]

}


rf_optimized = GridSearchCV(RandomForestClassifier(), param_grid=rf_param_grid, cv=5,
scoring='accuracy')

rf_optimized.fit(X_train, y_train)

best_rf_model = rf_optimized.best_estimator_

rf_accuracy = best_rf_model.score(X_test, y_test)

# Hyperparameter Optimization for SVM

svm_param_grid = {

    'C': [ 0.1, 0.8, 2, 10],

    'kernel': ['linear', 'rbf', 'poly', 'sigmoid']

}


svm_optimized = GridSearchCV(SVC(), param_grid=svm_param_grid, cv=5, scoring='accuracy')

svm_optimized.fit(X_train, y_train)

best_svm_model = svm_optimized.best_estimator_

svm_accuracy = best_svm_model.score(X_test, y_test)
```

```
final_scores = {
    'Random Forest': rf_accuracy,
    'SVM': svm_accuracy,
    'Logistic Regression': logreg_accuracy,
    'Decision Tree': dt_accuracy
}

best_model = max(final_scores, key=final_scores.get)

for name, accuracy in final_scores.items():
    print(f"{name} Accuracy: {accuracy}")

print("Best Model:", best_model, final_scores[best_model])
```