Pipeline Optimization

Mohammad Irfan Uddin

Introduction:

In this experiment, we aim to develop and optimize machine learning models to predict the quality of wines based on various features. The problem involves a classification task, where the quality of wine is the target variable, and we will explore the use of different machine learning algorithms and hyperparameter tuning techniques to achieve optimal predictive performance.

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 experiment is conducted using the Python programming language. We utilized popular machine learning libraries, including scikit-learn for building models, Optuna for hyperparameter optimization, and TPOT for automated machine learning.

Automated Machine Learning with TPOT: In addition to manual hyperparameter tuning with Optuna, we explore automated machine learning using TPOT. TPOT is configured to search for the best model among Decision Tree, Random Forest, and Gradient Boosting Classifier. The fit method is employed on the training data, and the resulting pipeline is evaluated on the test set.
The final accuracy of the TPOT-selected model is printed, and the best pipeline is exported to a Python script file (best_model_pipeline.py).

Data Preprocessing: The dataset is loaded using the pandas library, and features and target variable are separated. A standard train-test split with a ratio of 80:20 is performed using the train_test_split function from scikit-learn.

Preprocessing Steps:

Imputation: For numerical features, missing values are imputed using the mean strategy, while for categorical features, the most frequent strategy is employed.

Scaling: Numerical features are standardized using the StandardScaler to ensure that all features have the same scale.

One-Hot Encoding: Categorical features are one-hot encoded using the OneHotEncoder to convert them into a format suitable for machine learning algorithms.

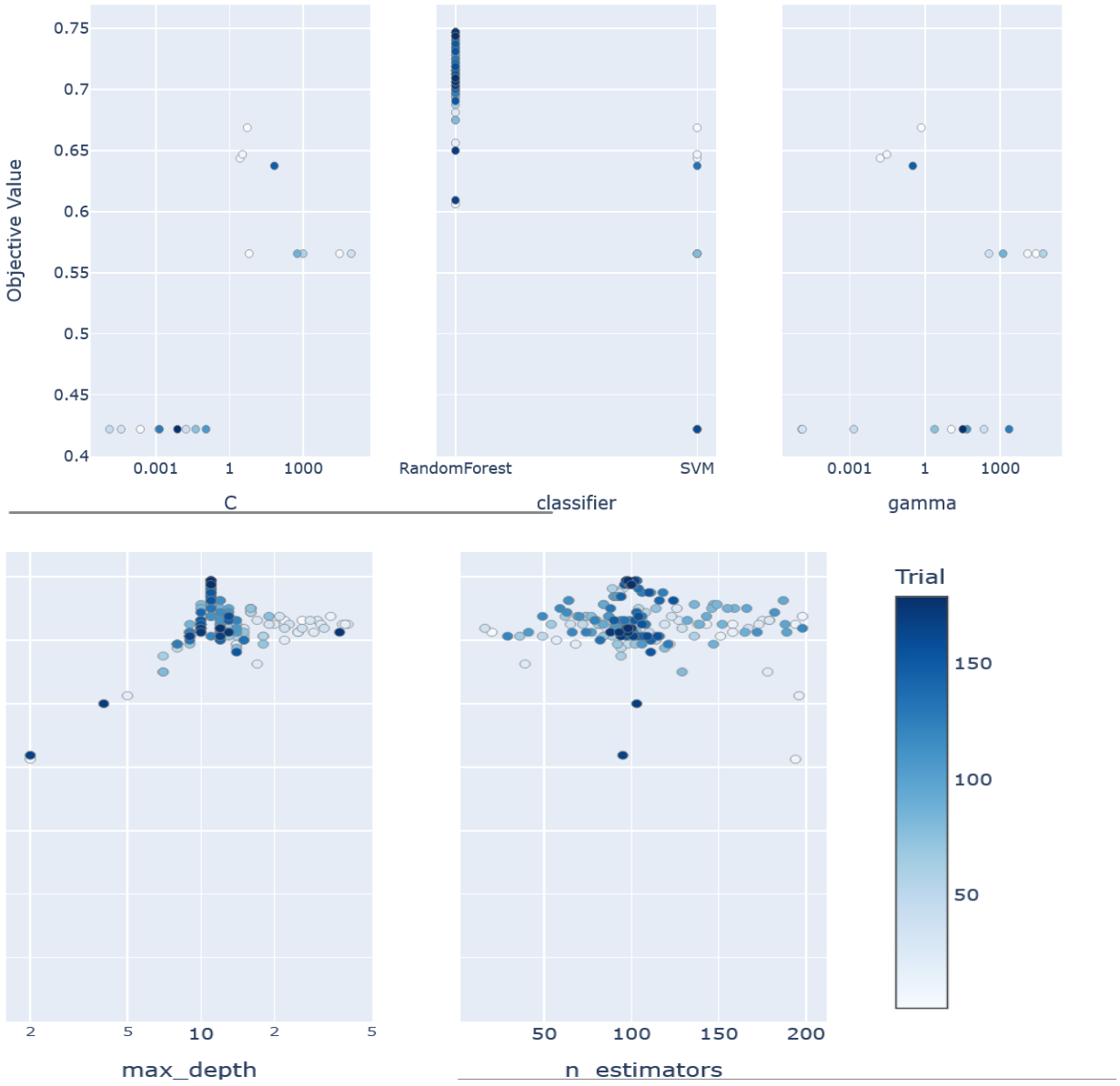
These preprocessing steps are implemented using scikit-learn's Pipeline and ColumnTransformer.

Hyperparameter Optimization: Optuna is used for hyperparameter optimization. The objective function is defined to train and evaluate the models based on the chosen hyperparameters. The optimization process aims to maximize the accuracy on the validation set.

Results:

I have tried to do the AutoML part with auto-sklearn but for some reasons I was facing an installation error in Google Collab platform. With TPOT, the best pipeline is exported to a Python script file (best_model_pipeline.py) is given below:

```
import numpy as np

import pandas as pd

from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split


# NOTE: Make sure that the outcome column is labeled 'target' in the data file

tpot_data = pd.read_csv('PATH/TO/DATA/FILE', sep='COLUMN_SEPARATOR', dtype=np.float64)

features = tpot_data.drop('target', axis=1)

training_features, testing_features, training_target, testing_target = \

        train_test_split(features, tpot_data['target'], random_state=0)


# Average CV score on the training set was: 0.6559957107843137

exported_pipeline = RandomForestClassifier()

# Fix random state in exported estimator

if hasattr(exported_pipeline, 'random_state'):

   setattr(exported_pipeline, 'random_state', 0)


exported_pipeline.fit(training_features, training_target)

results = exported_pipeline.predict(testing_features)
```



Code:

```
!pip install tpot
import pandas as pd
from sklearn.model_selection import train_test_split
from tpot import TPOTClassifier
from sklearn.metrics import accuracy_score

data = pd.read_csv('wineq.csv')
X = data.drop('quality', axis=1)
y = data['quality']
```

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

tpot = TPOTClassifier(
    generations=2,
    population_size=5,
    random_state=0,
    verbosity=3,
    config_dict={
        'sklearn.tree.DecisionTreeClassifier': {},
        'sklearn.ensemble.RandomForestClassifier': {},
        'sklearn.ensemble.GradientBoostingClassifier': {},
    },
)


tpot.fit(X_train, y_train)

y_pred = tpot.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

tpot.export('best_model_pipeline.py')
```

```
!pip install optuna
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
import optuna
import optuna.visualization as optuna_viz


def objective(trial):
    classifier_name = trial.suggest_categorical('classifier',
['RandomForest', 'SVM'])
    if classifier_name == 'RandomForest':
        n_estimators = trial.suggest_int('n_estimators', 10, 200)
        max_depth = trial.suggest_int('max_depth', 2, 40, log=True)
        clf = RandomForestClassifier(n_estimators=n_estimators,
max_depth=max_depth, random_state=42)
    else:
        C = trial.suggest_loguniform('C', 1e-5, 1e5)
```

```
        gamma = trial.suggest_loguniform('gamma', 1e-5, 1e5)
        clf = SVC(C=C, gamma=gamma, random_state=42)

    numeric_features = X.select_dtypes(include=['int64',
'float64']).columns
    categorical_features = X.select_dtypes(include=['object']).columns

    numeric_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='mean')),
        ('scaler', StandardScaler())
    ])

    categorical_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('onehot', OneHotEncoder(handle_unknown='ignore'))
    ])

    preprocessor = ColumnTransformer(
        transformers=[
            ('num', numeric_transformer, numeric_features),
            ('cat', categorical_transformer, categorical_features)
        ])

    pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                               ('classifier', clf)])

    pipeline.fit(X_train, y_train)

    y_pred = pipeline.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)

    return accuracy

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=200)

trial = study.best_trial
print('Accuracy: {}'.format(trial.value))
print("Best hyperparameters: {}".format(trial.params))

optuna_viz.plot_optimization_history(study)

optuna_viz.plot_slice(study)
```