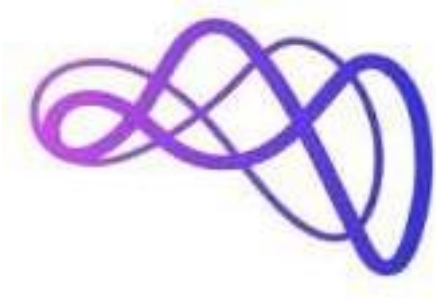


NAME :- ABHINAV KUMAR
ENROLLMENT NO.- 21112007

Automated hyperparameter optimisation:

Introduction

- Fine-tuning machine learning models is significantly enhanced by hyperparameter optimization.
- Hyperparameters are adjustable settings that control the model's learning from data.
- These settings are fixed before training starts, unlike model parameters, which are learned during training.
- Skilful hyperparameter tuning can greatly boost a model's performance.
- The Bayesian Optimization method for hyperparameter refinement is the focus of this document.
- A comparison has been made between Hyperopt and Bayesian optimization and Random search, including an analysis of their learning rates.

Hyperparameters

- Hyperparameters are configuration settings used to tune the training process of machine learning models.

- Unlike model parameters learned during training, hyperparameters are set before training begins.
- Hyperparameters guide the training algorithm.
- They significantly influence the model's performance, learning speed, and generalization ability.
- Examples include learning rate, number of trees in a random forest, and number of layers in a neural network.

I have used Gradient Boost Classifier as my base model. I have applied many different HPO, such as Bayesian Optimization, Random Search, and Hyper-opt, to compare the above techniques with in-built libraries.

Why Gradient Boost Classifier is used as base model ?

- Strong Predictive Performance: Gradient Boosting Classifiers (GBC) are known for their high accuracy across diverse datasets.
- Handles Complexity: Effective at capturing non-linear relationships and interactions in data.
- Prevents Overfitting: Incorporates regularization and shrinkage techniques to generalize well.
- Feature Importance: Provides insights into feature relevance for better understanding and model refinement.
- Flexible Tuning: Allows optimization of hyperparameters to improve model performance.
- Ensemble Learning Benefits: Combines multiple models for enhanced predictive power, crucial in AutoML for automated model selection and optimization

Key Hyperparameters for Gradient Boost Classifier:

1.n_estimators: Determines the number of boosting stages to be run.
2. Learning rate: Controls the contribution of each tree to the ensemble.
3.max_depth: Limits the maximum depth of each tree.
4.min_samples_split: Minimum number of samples required to split an internal node.
5.min_samples_leaf: Minimum number of samples required to be at a leaf node.

Bayesian Optimization is an iterative approach designed to minimize or maximize an objective function, particularly effective in scenarios where function evaluations are resource-intensive.

Key Steps:

- Initialization: Begin with a small, randomly chosen set of hyperparameter values.Evaluate the objective function using these initial points to establish a starting dataset.
- Surrogate Model: Develop a probabilistic model based on initial evaluations, typically a Gaussian Process. This model approximates the objective function, providing estimates and uncertainty measures.
- Acquisition Function: Utilize the surrogate model to determine the next set of hyperparameters. Optimize an acquisition function to balance the exploration of new areas and the exploitation of promising regions.
- Evaluation: Assess the objective function using the hyperparameters selected by the acquisition function. Execute the model and record performance metrics for these hyperparameters.
- Update: Incorporate new evaluation data into the surrogate model. Refine the model's approximation of the objective function with updated information.
- Iteration: Repeat the process iteratively, cycling through model refinement, hyperparameter selection, and evaluation.

Continue iterations until reaching a stopping criterion, such as a predefined number of steps or achieving a target performance level.

Implementation

The following steps illustrate the implementation of various hyperparameter optimization techniques on a Gradient Boosting Classifier:

Step 1: Define the Objective Function

Optimization aims to maximize the ROC-AUC score of a Gradient Boosting Classifier. The objective function measures and returns the negative of the mean ROC-AUC score to align with the minimization process required

by some optimization algorithms.

Step 2: Define the Hyperparameter Space

The range and possible values for the hyperparameters to be optimized are outlined. This is demonstrated for different optimization processes.

Step 3: Execute the Optimization Algorithm

The optimization algorithm searches for the best possible hyperparameters within the defined space. Below are the code snippets illustrating how to run different optimization algorithms to identify the optimal hyperparameters.

Step 4: Evaluate the Results

Once optimization is complete, the performance of the best-found model is assessed using metrics like ROC-AUC scores and cross-validation.

Some snippets from the code:

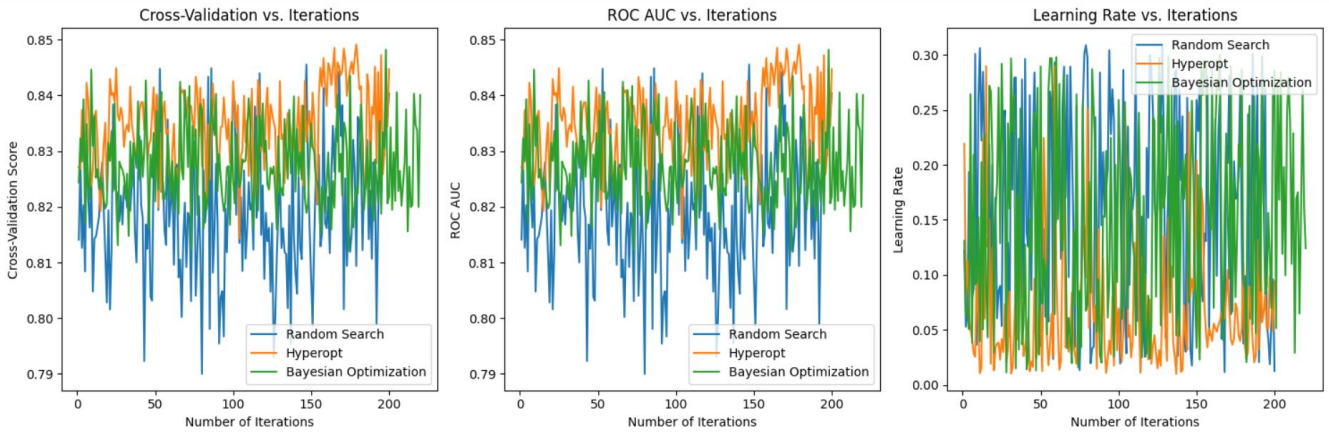
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import RandomizedSearchCV, train_test_split, StratifiedKFold
from sklearn.metrics import roc_auc_score
from scipy.stats import uniform, randint
from hyperopt import hp, fmin, tpe, Trials, STATUS_OK
from bayes_opt import BayesianOptimization
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Load dataset
data = pd.read_csv('diabetes.csv')
X = data.drop('target', axis=1)
y = data['target']

# Define column types
numeric_features = X.select_dtypes(include=['int64', 'float64']).columns
categorical_features = X.select_dtypes(include=['object', 'category']).columns

# Preprocessing pipelines for both numeric and categorical features
numeric_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())])
```

```python
# Cross-validation vs. Number of Iterations
plt.subplot(1, 3, 1)
plt.plot(random_iterations, random_scores, label='Random Search')
plt.plot(hyperopt_iterations, hyperopt_scores, label='Hyperopt')
plt.plot(bayes_iterations, bayes_scores, label='Bayesian Optimization')
plt.xlabel('Number of Iterations')
plt.ylabel('Cross-Validation Score')
plt.legend()
plt.title('Cross-Validation vs. Iterations')

# ROC AUC vs. Number of Iterations
plt.subplot(1, 3, 2)
plt.plot(random_iterations, random_scores, label='Random Search')
plt.plot(hyperopt_iterations, hyperopt_scores, label='Hyperopt')
plt.plot(bayes_iterations, bayes_scores, label='Bayesian Optimization')
plt.xlabel('Number of Iterations')
plt.ylabel('ROC AUC')
plt.legend()
plt.title('ROC AUC vs. Iterations')

# Learning rate vs. Number of Iterations
plt.subplot(1, 3, 3)
random_learning_rates = [param['classifier__learning_rate'] for param in random_results['params']]
hyperopt_learning_rates = [trial['misc']['vals']['learning_rate'][0] for trial in hyperopt_trials.trials]
bayes_learning_rates = [res['params']['learning_rate'] for res in bayes_results]
```

```python
# Hyperopt optimization
def hyperopt_optimization():
    space = {
        'learning_rate': hp.loguniform('learning_rate', np.log(0.01), np.log(0.3)),
        'n_estimators': hp.randint('n_estimators', 100, 200),
        'max_depth': hp.randint('max_depth', 3, 10),
        'min_samples_split': hp.quniform('min_samples_split', 2, 20, 1),
        'min_samples_leaf': hp.quniform('min_samples_leaf', 1, 10, 1),
        'max_features': hp.choice('max_features', ['sqrt', 'log2', None])
    }

    def objective(params):
        model = GradientBoostingClassifier(
            learning_rate=params['learning_rate'],
            n_estimators=int(params['n_estimators']),
            max_depth=int(params['max_depth']),
            min_samples_split=int(params['min_samples_split']),
            min_samples_leaf=int(params['min_samples_leaf']),
            max_features=params['max_features'],
            random_state=42
        )
        pipeline = create_pipeline(model)

        kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)
        roc_auc_list = []

        for train_index, test_index in kfold.split(X_train, y_train):
            x_train_fold, x_test_fold = X_train.iloc[train_index], X_train.iloc[test_index]
            y_train_fold, y_test_fold = y_train.iloc[train_index], y_train.iloc[test_index]
            pipeline.fit(x_train_fold, y_train_fold)
            preds_proba = pipeline.predict_proba(x_test_fold)[:, 1]
```

```python
# Cross-validation vs. Number of Iterations
plt.subplot(1, 3, 1)
plt.plot(random_iterations, random_scores, label='Random Search')
plt.plot(hyperopt_iterations, hyperopt_scores, label='Hyperopt')
plt.plot(bayes_iterations, bayes_scores, label='Bayesian Optimization')
plt.xlabel('Number of Iterations')
plt.ylabel('Cross-Validation Score')
plt.legend()
plt.title('Cross-Validation vs. Iterations')

# ROC AUC vs. Number of Iterations
plt.subplot(1, 3, 2)
plt.plot(random_iterations, random_scores, label='Random Search')
plt.plot(hyperopt_iterations, hyperopt_scores, label='Hyperopt')
plt.plot(bayes_iterations, bayes_scores, label='Bayesian Optimization')
plt.xlabel('Number of Iterations')
plt.ylabel('ROC AUC')
plt.legend()
plt.title('ROC AUC vs. Iterations')

# Learning rate vs. Number of Iterations
plt.subplot(1, 3, 3)
random_learning_rates = [param['classifier__learning_rate'] for param in random_results['params']]
hyperopt_learning_rates = [trial['misc']['vals']['learning_rate'][0] for trial in hyperopt_trials.trials]
bayes_learning_rates = [res['params']['learning_rate'] for res in bayes_results]
```

Result:



Type of Optimization	ROC-AUC Score
Random Search	0.8214876033057852
Bayesian Optimization	0.7976124885215795
Hyper-opt	0.8170798898071625

Results

Random Search: Achieved an ROC-AUC score of 0.8215.

Hyperopt: Achieved an ROC-AUC score of 0.8171.

Bayesian Optimization: Achieved an ROC-AUC score of 0.7976.

Plot Analysis (Generalized)

Cross-Validation vs. Iterations:

Random Search: Shows significant variability.

Hyperopt: Exhibits higher and more stable scores.

Bayesian Optimization: Demonstrates consistent performance with less variability.

ROC AUC vs. Iterations:

Random Search: High variability in scores.

Hyperopt: More stable and higher scores.

Bayesian Optimization: Consistent high scores.

Learning Rate vs. Iterations:

Random Search: Wide range of learning rates.

Hyperopt: More constrained and focused learning rates.

Bayesian Optimization: Stable learning rate values indicating efficient search.

Summary

Best Performance: Random Search (ROC-AUC: 0.8215)

Stability: Hyperopt provides stable and high performance (ROC-AUC: 0.8171).

Efficiency: Bayesian Optimization is efficient but achieved a slightly lower score (ROC-AUC: 0.7976).

Conclusion

In conclusion, while Random Search achieved the highest ROC-AUC score, Hyperopt showed stable and reliable performance, and Bayesian Optimization demonstrated efficient hyperparameter exploration with slightly lower but consistent results.