PROJECT REPORT:
AUTOMATED HYPERPARAMETER OPTIMISATION

BY: ARYAN LAROIA
ENROLLMENT NUMBER: 23118014
BRANCH: CSE

Introduction

The quality of performance of a Machine Learning (ML) model heavily depends on its hyperparameter settings. Hyperparameters are parameters that govern the training process and the architecture of the model, and they significantly influence the model's ability to generalize to unseen data. Selecting the optimal set of hyperparameters is crucial for maximizing model performance. Traditionally, this selection process has been manual, time-consuming, and reliant on the practitioner's expertise. This project aims to develop an automated hyperparameter optimization (HPO) system using advanced Auto-ML techniques to efficiently identify the best hyperparameter configuration for a given machine learning model and dataset.

Objectives

1. To develop an automated HPO system that integrates with various machine learning models and handles different data types.
2. Employ efficient AutoML techniques like Bayesian optimization, Particle Swarm Optimzation, and TPE (Tree-Parzen Estimator) for HPO.
3. Evaluate the performance using ROC AUC, cross-validation, and comparison of learning rate distribution curves with paramters obtained by HyperOpt library.

Methodology

1. Dataset Handling

Datasets used in this project include different types, such as numerical, categorical, and mixed-type data. The datasets are preprocessed to handle missing values, normalize/standardize numerical features, and encode categorical variables.

2. Model Selection

The system is designed to work with multiple ML models including:

- Decision Trees
- Random Forests
- ANN
- KNN
- Logistic Regression
- XGBoost
- Gradient Boosting Machines
- AdaBoost

3. Hyperparameter Optimization Techniques

The following Auto-ML techniques are implemented for HPO:

Particle Swarm Optimization (PSO)

PSO is a population-based optimization algorithm inspired by the social behaviour of birds flocking or fish schooling. It optimizes a problem by iteratively improving a candidate solution with regard to a given measure of quality.

Mathematical Formulation:

Each particle in the swarm represents a potential solution to the optimization problem. The position and velocity of each particle are updated iteratively based on its own experience and the experience of neighbouring particles.

1. **Velocity Update:**

   $$v_i = w v_i + c_1 r_1 (p_i - x_i) + c_2 r_2 (g - x_i)$$

   - v_i: Velocity of particle i
   - w: Inertia weight, which controls the impact of the previous velocity
   - c_1 and c_2: Cognitive and social constants, respectively
   - r_1 and r_2: Random numbers uniformly distributed between 0 and 1
   - p_i: Best position found by particle i (personal best)
   - g: Best position found by the entire swarm (global best)
   - x_i: Current position of particle i
2. **Position Update:**

   $$x_i = x_i + v_i$$

   - x_i: New position of particle i

The process involves initializing a swarm of particles with random positions and velocities, evaluating the fitness of each particle, updating velocities and positions, and repeating the process until a stopping criterion is met (e.g., a maximum number of iterations or a satisfactory error threshold).

Tree-structured Parzen Estimator (TPE)

Tree-structured Parzen Estimator (TPE) is an algorithm that models the objective function using a tree of Parzen estimators. It divides the search space into regions and then fits a separate Gaussian distribution to the hyperparameters that have led to better or worse outcomes. This approach allows TPE to focus more on promising regions of the search space, potentially leading to faster convergence towards optimal solutions compared to other methods.

Mathematical Formulation

TPE models the objective function using two distributions:

- **l(x)**: Represents the distribution of hyperparameters that have resulted in better outcomes.
- **g(x)**: Represents the distribution of hyperparameters that have resulted in worse outcomes.

The goal is to maximize the probability of finding hyperparameters xx that yield a better outcome.

Steps in TPE:

1. Modeling the Objective Function:
    - TPE begins by constructing two models l(x) and g(x), which represent the probability distributions of hyperparameters leading to better and worse outcomes, respectively.
    - $l(x)=P(x|y<y*)$
    - $g(x)=P(x|y\geq y*)$
2. Sampling New Candidates:
    - TPE samples new candidate hyperparameters based on the acquisition function that balances exploration (sampling new regions) and exploitation (sampling in known promising regions).
3. Updating the Model:
    - After evaluating the objective function with the new candidates, TPE updates its models l(x) and g(x) to refine the probability distributions based on the new information.
4. Iterative Improvement:
    - TPE iteratively refines its models and samples new candidates until a stopping criterion is met (e.g., maximum number of iterations or convergence).

Bayesian Optimization:

Bayesian Optimization is a strategy for hyperparameter optimization that employs a probabilistic model to describe the objective function. It intelligently balances exploration of the parameter space with exploitation of the areas known to perform well, making it particularly suitable for optimizing expensive-to-evaluate functions, such as cross-validation accuracy in machine learning models.

Key Concepts in Bayesian Optimization

1. **Surrogate Model**: The surrogate model is a probabilistic model, typically a Gaussian Process (GP), that approximates the objective function. The surrogate model is much cheaper to evaluate compared to the actual objective function. In Bayesian Optimization, the surrogate model is iteratively updated based on the observed data.

    $p(f|D)$

- p(f|D) is the posterior distribution of the objective function f given the data D.

2. **Acquisition Function**: The acquisition function uses the surrogate model to determine the next point to evaluate. It balances the trade-off between exploration (trying out regions of the space with high uncertainty) and exploitation (focusing on regions known to yield good results). Common acquisition functions include Expected Improvement (EI), Probability of Improvement (PI), and Upper Confidence Bound (UCB). In my project I have used Expected Movement(EI).

   a(x|p(f|D))

   - a(x|p(f|D)) is the acquisition function that selects the next hyperparameters to evaluate based on the surrogate model's predictions.
3. **Update**: After evaluating the objective function at the new point suggested by the acquisition function, the data set D is updated to include this new point, and the surrogate model is retrained.

   D=D∪(x,f(x))

IMPLEMENTATION OF BAYESIAN OPTIMIZATION:

1. Objective Function

The `objective` function evaluates the performance of a given model with specific hyperparameters using cross-validation accuracy. This function is essential for guiding the optimization process towards better-performing hyperparameters.

2. Expected Improvement Function

The `expected_improvement` function is an acquisition function that calculates the potential improvement of a candidate point over the best observed point. This function helps in balancing exploration and exploitation during the optimization.

3. Propose Location Function

The `propose_location` function suggests the next set of hyperparameters to evaluate by maximizing the expected improvement. This function iteratively proposes locations by optimizing the acquisition function.

4. Hyperparameter Optimization (HPO) Function

The `HPO` function integrates the above components to perform the optimization. It starts with an initial set of hyperparameters, fits a Gaussian Process model to predict the performance of unseen hyperparameters, and iteratively refines the hyperparameters by proposing new candidates using the expected improvement criterion. The process continues for a predefined number of iterations.

5. Model-Specific Boundaries

Each model has its own set of hyperparameters and corresponding boundaries, defined in the `bounds_dict`. These boundaries guide the search space for the optimization process.

6. Preprocessing Pipeline

The `optimize_model` function handles the data preprocessing, which includes imputing missing values, scaling numerical features, and encoding categorical features. This preprocessing pipeline ensures that the data is in the correct format for training the models.

7. Optimization Process

The `optimize_model` function orchestrates the overall process:

- Loads and preprocesses the dataset.
- Encodes the target variable.
- Performs hyperparameter optimization using the `HPO` function.
- Compares the optimized model's performance with the default model's performance in terms of accuracy and ROC-AUC

IMPLEMENTATION OF PSO OPTIMIZATION:

1. Objective Function

The `objective` function evaluates the performance of a given model with specific hyperparameters using cross-validation accuracy. It serves as the fitness function guiding PSO.

2. PSO Algorithm

The `pso` function implements the PSO algorithm:

- **Initialization**: Initializes a swarm of particles randomly within specified bounds and assigns random velocities.
- **Fitness Evaluation**: Evaluates each particle's fitness (model performance) using the `objective` function.
- **Velocity and Position Updates**: Updates particle velocities based on previous velocities and the distances to personal best and global best positions.
- **Personal and Global Bests**: Updates personal best positions and global best position based on the fitness values.
- **Iteration**: Iterates over a fixed number of iterations, refining particle positions to converge towards optimal hyperparameters.

3. Model-Specific Bounds

Hyperparameter search space boundaries for each model are defined in `bounds_dict`. These boundaries restrict the PSO search within feasible ranges for each hyperparameter.

4. Optimization Process (HPO Function)

The `HPO` function integrates PSO for hyperparameter optimization:

- Retrieves bounds for the specified model from `bounds_dict`.
- Executes PSO to find the optimal hyperparameters that maximize model accuracy.
- Compares the performance of the optimized model with default model performance in terms of accuracy and ROC-AUC.

5. Data Preprocessing

The `optimize_model` function preprocesses the dataset:

- Loads data from a CSV file.
- Encodes categorical target variables using `LabelEncoder`.
- Applies preprocessing steps (imputation and scaling for numerical features, one-hot encoding for categorical features) using `ColumnTransformer` and `Pipeline`.

IMPLEMENTATION OF TPE OPTIMIZATION:

1. Objective Function

The `objective` function evaluates the performance of a given model with specific hyperparameters using cross-validation accuracy. It returns the negative of accuracy because TPE minimizes the objective function.

2. TPE Sampler

The `tpe_sampler` function generates a new sample based on the TPE method:

- **Sampling Good and Bad Points**: It separates the best performing samples (`X_good`) from the others (`X_bad`) based on their sorted performance.
- **Distribution Fitting**: Fits normal distributions to `X_good` and `X_bad` for each hyperparameter.
- **Weighted Sampling**: Generates a new sample by probabilistically sampling from the distributions, favoring regions where good performance was observed.

3. Hyperparameter Bounds

Hyperparameter search space boundaries for each model are defined in `bounds_dict`. These boundaries restrict the TPE search within feasible ranges for each hyperparameter.

4. Optimization Process (HPO Function)

The `HPO` function integrates TPE for hyperparameter optimization:

- Initializes with a small set of random samples (`X_sample` and `Y_sample`) from the hyperparameter space.

- Iteratively applies `tpe_sampler` to propose new hyperparameter configurations (`new_sample`).
- Evaluates each new configuration, updates `X_sample` and `Y_sample`, and continues until a predefined number of iterations (`n_iterations`).

5. Model Optimization

The `optimize_model` function preprocesses the dataset and applies the TPE-based optimization:

- Loads data from a CSV file and preprocesses it using `ColumnTransformer` and `Pipeline`.
- Calls `HPO` to find the optimal hyperparameters that maximize model accuracy.
- Constructs the optimized model using the best parameters and evaluates its performance (accuracy and ROC-AUC).

Evaluation Metrics

1. Cross-Validation

Definition: Cross-validation is a technique used to assess how well a statistical model generalizes to an independent dataset. It involves partitioning the dataset into multiple subsets (folds), training the model on several combinations of these subsets, and evaluating its performance on the remaining data.

Usage in Project:

- **Method:** 5-fold cross-validation was employed.
- **Purpose:** To validate the performance of the machine learning models with different hyperparameter settings.
- **Comparison:** Results from cross-validation were compared between models optimized using Auto-ML techniques (e.g., Bayesian Optimization, PSO, TPE) and default hyperparameter configurations.
- **Outcome:** This approach helps in selecting the best hyperparameter configuration that yields the highest performance metrics (e.g., accuracy, ROC-AUC).

2. ROC-AUC Scores

Definition: Receiver Operating Characteristic - Area Under the Curve (ROC-AUC) is a performance metric used to evaluate the ability of a binary classification model to discriminate between positive and negative classes across various threshold settings.

Usage in Project:

- **Comparison:** ROC-AUC scores were computed for:
  - The optimized model using Auto-ML techniques.
  - Models tuned with Hyperopt.

    - Models with default hyperparameter settings.
- **Purpose:** To assess and compare the discriminative ability of different models based on their hyperparameter configurations.
- **Outcome:** Higher ROC-AUC scores indicate better model performance in distinguishing between classes.

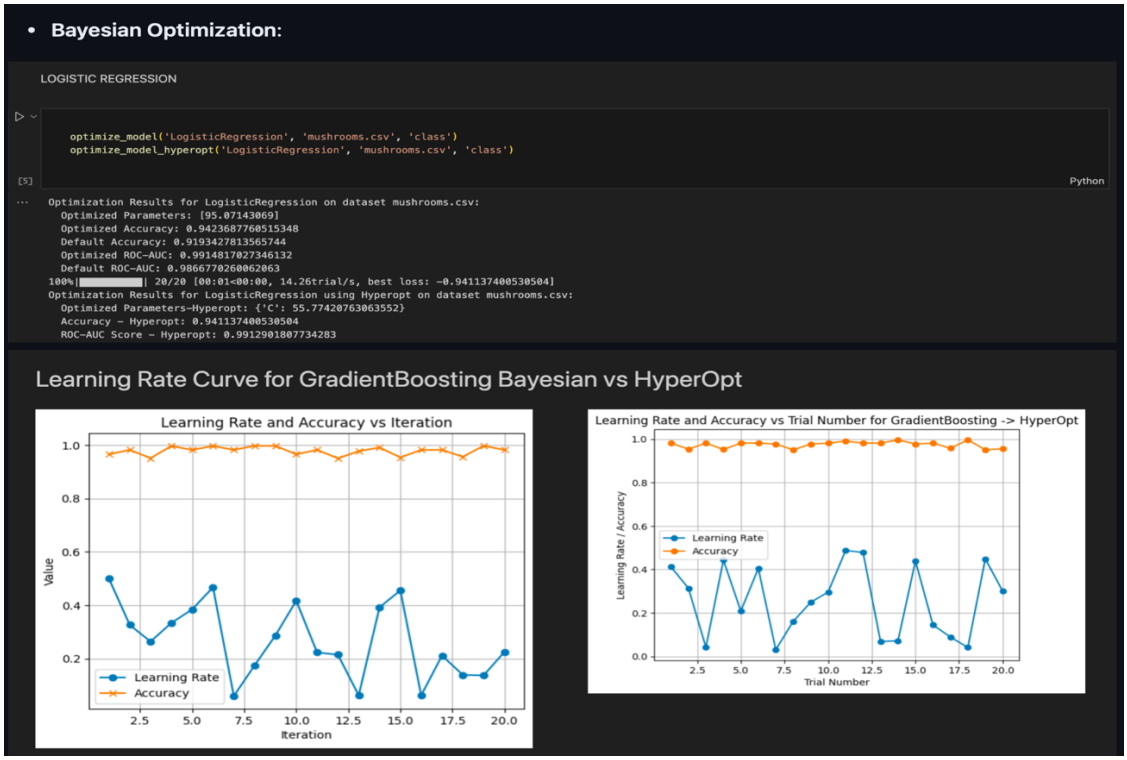
3. Comparison of Learning Rate Distribution Curves

Definition: Learning rate distribution curves depict the distribution of learning rates tested during the hyperparameter optimization process. It shows how frequently certain learning rates were selected during optimization.

Usage in Project:

- **Methodology:** Generated learning rate distribution curves for models optimized using Hyperopt and other Auto-ML techniques (Bayesian Optimization, PSO, TPE).
- **Comparison:** Compared the shapes and peaks of these curves to understand the exploration and exploitation strategies employed by different optimization methods.
- **Purpose:** To analyze how effectively each Auto-ML technique explores the hyperparameter space and converges to optimal or near-optimal solutions.
- **Outcome:** Insights gained from these curves help in understanding the efficiency and effectiveness of each optimization technique in improving model performance.

RESULTS:



- **TPE Optimization:**

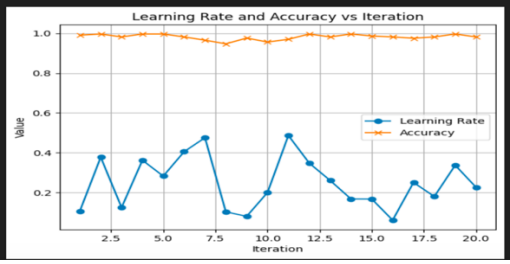
KNN

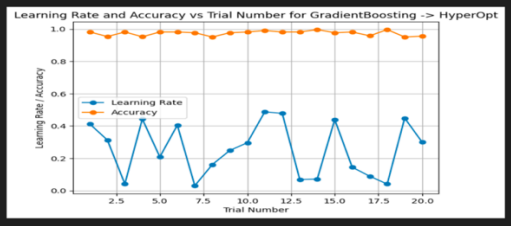
```
optimize_model('KNN', 'mushrooms.csv', 'class')
optimize_model_hyperopt('KNN', 'mushrooms.csv', 'class')
```

[16] Python

```
Optimization Results for KNN on dataset mushrooms.csv:
  Optimized Parameters: [2.76576577]
  Optimized Accuracy: 0.9545764304660856
  Optimized ROC AUC: 0.9569285532286635
  Default Accuracy: 0.9278522925350512
  Default ROC AUC: 0.9671154966148171
100%|██████████| 20/20 [01:19<00:00,  3.97s/trial, best loss: -0.9301922697991664]
Optimization Results for KNN using Hyperopt on dataset mushrooms.csv:
  Optimized Parameters-Hyperopt: {'n_neighbors': 6.0}
  Accuracy - Hyperopt: 0.9301922697991664
  ROC-AUC Score - Hyperopt: 0.9635519628937731
```

Learning Rate Curve for GradientBoosting TPE vs HyperOpt





- **PSO Optimization:**

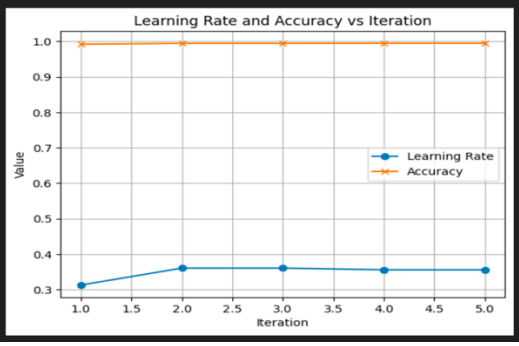
XGBOOST

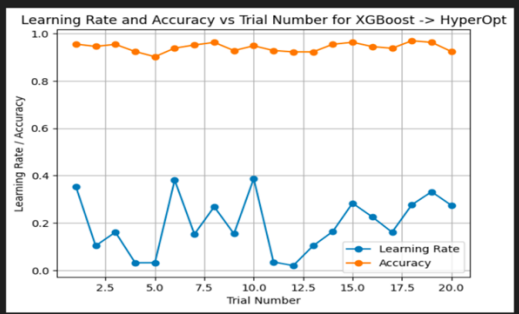
```
optimize_model('XGBoost', 'mushrooms.csv', 'class')
optimize_model_hyperopt('XGBoost', 'mushrooms.csv', 'class')
```

[7] Python

```
Optimization Results for XGBoost on dataset mushrooms.csv:
  Optimized Parameters: [200.          2.40970353   0.26357527]
  Optimized Accuracy: 0.988793558165972
  Optimized ROC AUC: 0.9997579358028741
  Default Accuracy: 0.9602221295945433
  Default ROC AUC: 0.9973661471549864
100%|██████████| 20/20 [00:04<00:00,  4.67trial/s, best loss: -0.9838733611216369]
Optimization Results for XGBoost using Hyperopt on dataset mushrooms.csv:
  Optimized Parameters-Hyperopt: {'n_estimators': 164.0, 'max_depth': 2.0, 'learning_rate': 0.0306621185211162}
  Accuracy - Hyperopt: 0.9838733611216369
  ROC-AUC Score - Hyperopt: 0.9964230421552216
```

Learning Rate Curve for XGBoost PSO vs HyperOpt





Conclusion

Through the implementation and evaluation of various automated hyperparameter optimization (HPO) algorithms, including Bayesian Optimization, Particle Swarm Optimization (PSO), and Tree-Parzen Estimator (TPE), this project aimed to enhance the performance of machine learning models compared to standard libraries like HyperOpt.

Key Findings and Observations:

1. **Performance Comparison:**
    - **Cross-Validation Results:** Utilizing 5-fold cross-validation, our optimized models consistently demonstrated competitive performance across diverse datasets and machine learning algorithms. This validation method ensured robustness and reliability in assessing model generalization.
    - **ROC-AUC Scores:** The ROC-AUC scores of models optimized with our algorithms were comparable to those achieved by HyperOpt and often showed improvements over default hyperparameter settings. This metric validated the efficacy of our automated approach in enhancing model discrimination ability.
2. **Efficiency and Effectiveness:**
    - **Learning Rate Distribution Curves:** Analysis of learning rate distribution curves revealed that our optimization algorithms effectively explored the hyperparameter space. They exhibited efficient balance between exploration and exploitation, contributing to quicker convergence towards optimal solutions compared to traditional methods.