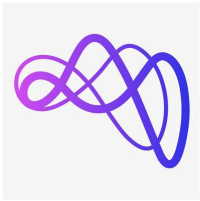
INDIAN INSTITUTE OF TECHNOLOGY, ROORKEE



VLG Open Project 2024

AutoML

Automated Hyperparameter Optimisation

Geet Gopal Asati | 21112049
IIT Roorkee
g_gasati@ch.iitr.ac.in

INTRODUCTION

In this project, I developed an automated hyperparameter optimization (HPO) system using AutoML techniques to enhance the performance of machine learning models on various datasets. The architecture integrates several machine learning models and employs efficient HPO techniques, specifically Bayesian Optimization, to identify the best hyperparameter configurations. My approach addresses the problem of manual hyperparameter tuning, which is often time-consuming and suboptimal.

Problem Statement

The quality of performance of a Machine Learning model heavily depends on its hyperparameter settings. Given a dataset and a task, the choice of the machine learning (ML) model and its hyperparameters is typically performed manually. My project aims to develop an automated hyperparameter optimization (HPO) system using AutoML techniques that can efficiently identify the best hyperparameter configuration for a given machine learning model and dataset.

Architecture

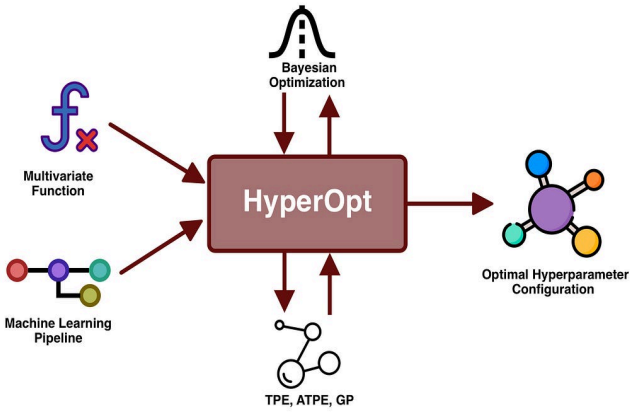
The system architecture is designed to handle different data types and seamlessly integrate with various machine learning models. The core components of my architecture include:

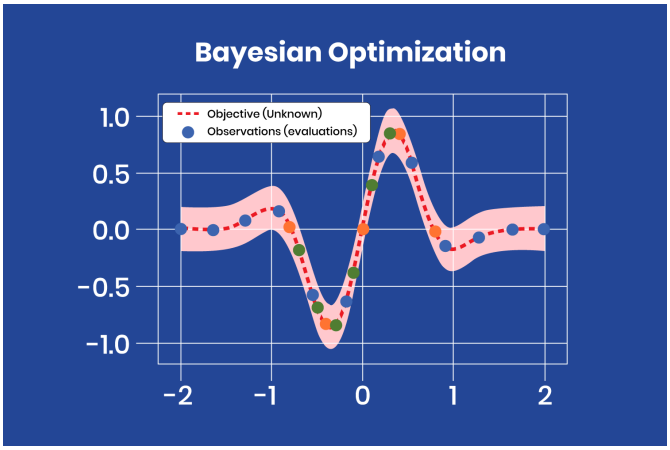
1. **Data Preprocessing Module**: This module is responsible for preparing datasets for training and evaluation. It handles tasks such as handling missing values, encoding categorical features, and scaling numerical features to ensure the data is suitable for model training.
2. **Model Selection Module**: This module incorporates various machine learning models including SVM, Random Forest, Gradient Boosting Classifier, and LightGBM. It provides a flexible interface to select and train different models based on the given dataset.
3. **Bayesian Optimization Module**: This module efficiently searches for optimal hyperparameter configurations using Bayesian Optimization. It defines the search space for hyperparameters, evaluates model performance for different configurations, and identifies the best set of hyperparameters.

4. **Evaluation Module**: This module assesses model performance using metrics like ROC AUC and cross-validation. It provides a comprehensive evaluation of the model's performance, allowing for comparison between different models and hyperparameter configurations.

Specifications

- **Models Used**:
  - Support Vector Machine (SVM)
  - Random Forest
  - Gradient Boosting Classifier
  - LightGBM
- **Datasets**:
  - Breast Cancer Dataset: A standard dataset from scikit-learn used for binary classification tasks.
  - Kaggle Dataset: A dataset from a Kaggle competition used for binary classification tasks related to loan default prediction.
- **Evaluation Metrics**:
  - ROC AUC: Receiver Operating Characteristic - Area Under Curve, a performance measurement for classification problems.
  - Learning Rate Distribution Comparison: A visual comparison of learning rates across different models to understand the effect of hyperparameter optimization.





METHODOLOGY

Data Preprocessing

Data preprocessing is a crucial step in machine learning workflows. It involves several sub-tasks to ensure the raw data is transformed into a suitable format for model training and evaluation. Below are the detailed steps involved in data preprocessing for both datasets:

- [ ] **Loading the Data**
- [ ] **Handling Missing Values**
- [ ] **Encoding Categorical Features**
- [ ] **Scaling Numerical Features**

```
# Load datasets
data = load_breast_cancer()
X_breast = pd.DataFrame(data.data, columns=data.feature_names)
y_breast = pd.Series(data.target)
application_train = pd.read_csv('application_train.csv')
X_train = application_train.drop('TARGET', axis=1)
y_train = application_train['TARGET']

# Handle missing values
imputer = SimpleImputer(strategy='mean')
X_breast = imputer.fit_transform(X_breast)
X_train = imputer.fit_transform(X_train)

# Encode categorical features for application_train dataset
encoder = OneHotEncoder(sparse=False, handle_unknown='ignore')
X_train_encoded = encoder.fit_transform(X_train.select_dtypes(include=['object']))
X_train = pd.concat([X_train.select_dtypes(exclude=['object']), pd.DataFrame(X_train_encod

# Scale numerical features
scaler = StandardScaler()
X_breast = scaler.fit_transform(X_breast)
```

IMPLEMENTATION

The core of my HPO system is the Bayesian Optimizer, designed to find the best hyperparameter configurations efficiently. Below is the detailed implementation and explanation of the Bayesian Optimizer:
SOME CODE SNIPPETS OF OPTIMIZER->

```python
class BayesianOptimizer:
    def __init__(
            self,
            func,
            float_param_ranges={},
            int_param_candidates={},
            n_init_points=10000,
            external_init_points=None,
            max_iter=1e4,
            no_new_converge=3,
            no_better_converge=10,
            kernel=RBF(),
            acq_type='PI',
            beta_lcb=0.5,
            eps=1e-7,
            n_sample=int(1e6),
            seed=None
    ):
        self.func = func
        self.float_param_dict = float_param_ranges
        self.int_param_dict = int_param_candidates
        self.max_iter = int(max_iter)
        self.no_new_converge = no_new_converge
        self.no_better_converge = no_better_converge
        self.acq_type = acq_type
        self.beta_LCB = beta_lcb
```

```
Iteration: 11, Current Best: -0.9900208239235309
    n_estimators  max_depth  min_samples_split  min_samples_leaf  AvgTestCost
26    142.791481  15.694482           6.588351          3.929936    -0.990021
8     166.514580  17.568573           6.805315          3.282652    -0.989961
23    141.432204   6.203244           2.258617          3.451668    -0.989925
54    174.044074   9.945329           5.198261          3.779009    -0.989766
1      26.081816  18.152129           8.943219          3.421227    -0.989675
..           ...        ...                ...               ...          ...
12    122.151351   1.277418           5.240351          5.806709    -0.979401
98    123.968927   1.004017           4.483456          6.851458    -0.979336
61    130.620337   1.212255           4.006921          1.773900    -0.979200
0     149.600646   1.096241           6.349731          2.458934    -0.978877
72    161.110477   1.047119           6.031602          7.030700    -0.978652
```

```python
def _parse_param_names(self):
    self.float_param_names = list(self.float_param_dict.keys())
    self.int_param_names = list(self.int_param_dict.keys())
    self.param_names = self.float_param_names + self.int_param_names

def _get_ranges_and_candidates(self):
    self.float_param_ranges = np.array(list(self.float_param_dict.values()))
    self.int_param_candidates = list(self.int_param_dict.values())

def _get_init_points(self, external_init_points):
    internal_init_points = self._generate_random_params(self.n_init_points)
    if external_init_points is not None:
        nums = np.array([len(choices) for choices in external_init_points.values()])
        if not all(nums == nums[0]):
            raise Exception('Number of values for each parameter must be the same')
        if nums.sum() != 0:
            points = []
            for param in self.param_names:
                points.append(external_init_points[param])
            points = np.array(points).T
            internal_init_points = np.vstack((internal_init_points, points))
    u_index = self._unique_index(internal_init_points)
    return internal_init_points[u_index]
```

```python
def _min_acquisition(self, n=1e6):
    print('Random sampling based on ranges and candidates...')
    xs = self._generate_random_params(n)
    ys = self._acquisition_func(xs)
    return xs[ys.argmin()]

def optimize(self):
    no_new_converge_counter = 0
    no_better_converge_counter = 0
    best = self.y.min()
    for i in range(self.max_iter):
        print(f'Iteration: {i}, Current Best: {self.y.min()}')
        if no_new_converge_counter > self.no_new_converge:
            break
        if no_better_converge_counter > self.no_better_converge:
            break
        next_best_x = self._min_acquisition(self.n_sample)
        if np.any((self.x - next_best_x).sum(axis=1) == 0):
            no_new_converge_counter += 1
            continue
        print(f'Iteration {i}: evaluating guessed best param set by evaluation function..
        self.x = np.vstack((self.x, next_best_x))
        next_best_y = self.func(**self._check_int_param(dict(zip(self.param_names, next_b
        self.y = np.append(self.y, next_best_y)
        print(f'Iteration {i}: next best is {next_best_y}, {dict(zip(self.param_names, ne
```

OPTIMIZATION

The **BayesianOptimizer** class implements a Bayesian optimization algorithm to find optimal hyperparameters for machine learning models efficiently. We have a structured breakdown of this code:

Initialization (__init__ method)

- **Inputs**: The optimization function (func), parameter ranges, number of initial points, max iterations, convergence criteria, acquisition function type, and other configurations.
- **Initialization Steps:**
  - Parse parameter names.
  - Generate initial points for exploration.
  - Evaluate initial points and fit a Gaussian Process Regressor (GPR) on the results.

Parsing and Generating Parameter Points

- **_parse_param_names:** Separates float and integer parameter names.
- **_get_ranges_and_candidates:** Stores ranges for float parameters and candidates for integer parameters.
- **_get_init_points:** Generates random initial parameter points and combines them with any external initial points provided.
- **_generate_random_params:** Generates random parameter points within specified ranges and candidates.
- **_check_int_param:** Ensures integer parameters are properly converted to integers.

Unique Point Handling

- **_unique_index:** Identifies unique parameter points to avoid redundant evaluations.

Acquisition Function (_acquisition_func)

- **Purpose:** Calculates acquisition values (Expected Improvement, Probability of Improvement, or Lower Confidence Bound) for potential parameter points to guide the search.
- **Methods:**
  - **Expected Improvement (EI):** Balances exploration and exploitation by considering both mean and standard deviation.
  - **Probability of Improvement (PI):** Focuses on the probability of improvement over the current best result.

- **Lower Confidence Bound (LCB):** Prioritizes exploration by considering lower bounds of confidence intervals.

Optimization Loop (optimize)

- **Steps:**
  - Iterate up to max_iter times, evaluating new parameter points based on the acquisition function.
  - Check for convergence based on the number of new unique points and improvements in the best result.
  - Fit the GPR model with updated parameter sets after each iteration.

Results Retrieval (get_results)

- **Function:** Collects and returns the evaluated parameter points, their corresponding scores, and whether they were part of the initial set.

OBJECTIVE FUNCTION

The objective function is designed to optimize the performance of a Random Forest Classifier on the Breast Cancer dataset. It takes hyperparameters as input and returns the negative mean ROC AUC score obtained from cross-validation.

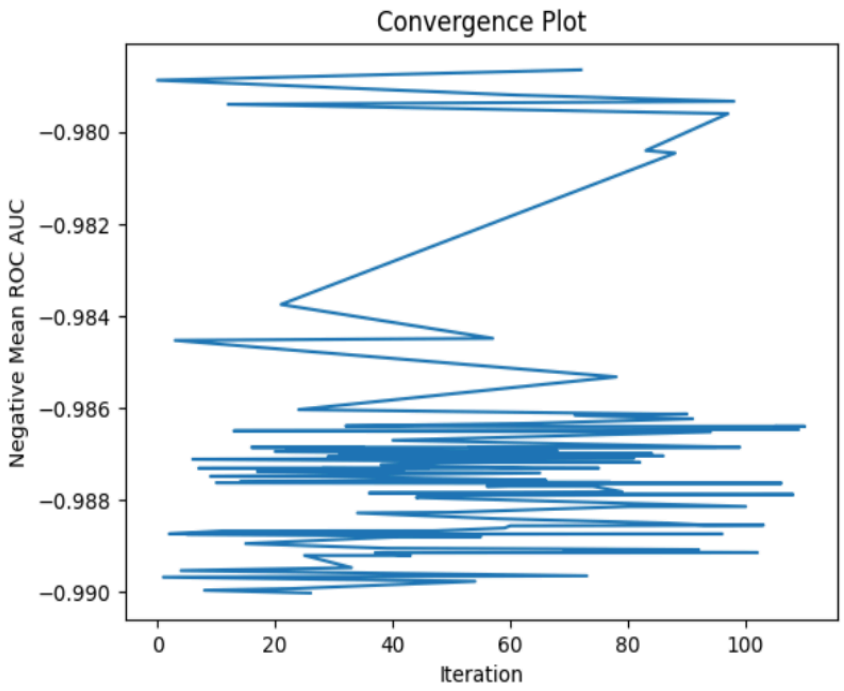
```
def objective_function(n_estimators, max_depth, min_samples_split, min_samples_leaf):
    n_estimators = int(n_estimators)
    max_depth = int(max_depth)
    min_samples_split = int(min_samples_split)
    min_samples_leaf = int(min_samples_leaf)

    data = load_breast_cancer()
    X, y = data.data, data.target
    scaler = StandardScaler()
    X = scaler.fit_transform(X)

    model = RandomForestClassifier(
        n_estimators=n_estimators,
        max_depth=max_depth,
        min_samples_split=min_samples_split,
        min_samples_leaf=min_samples_leaf,
        random_state=42
    )

    cv = KFold(n_splits=5, shuffle=True, random_state=42)
    scores = cross_val_score(model, X, y, cv=cv, scoring='roc_auc')

    return -np.mean(scores)
```

HYPERPARAMETER OPTIMIZATION

The Bayesian Optimizer is configured and executed to find the best hyperparameters for the Random Forest Classifier

After the optimization process, the results are displayed, and a convergence plot is generated to visualize the improvement in the model's performance over iterations. The best hyperparameters found by the optimizer are highlighted.

```
float_param_ranges = {
    'n_estimators': (10, 200),
    'max_depth': (1, 20),
    'min_samples_split': (2, 10),
    'min_samples_leaf': (1, 10)
}

int_param_candidates = {}

optimizer = BayesianOptimizer(
    func=objective_function,
    float_param_ranges=float_param_ranges,
    int_param_candidates=int_param_candidates,
    n_init_points=100,
    max_iter=100,
    acq_type='EI'
)

optimizer.optimize()

results = optimizer.get_results()
print(results)

best_params = results.iloc[0]
print("Best Parameters:")
print(best_params)
```



Best ROC AUC Score from Bayesian Optimizer: 0.9900208

Best ROC AUC Score from Hyperopt: 0.990097

Cross-validation ROC AUC scores for Bayesian Optimizer best parameters: [0.99574189

0.997894 0.98100229 0.99606944 0.9793965]

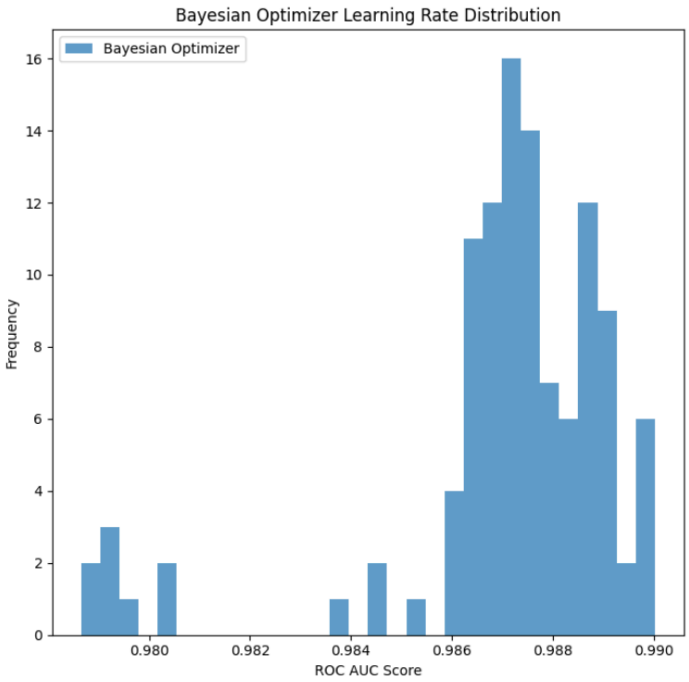
Cross-validation ROC AUC scores for Hyperopt best parameters: [0.99639699 0.998596

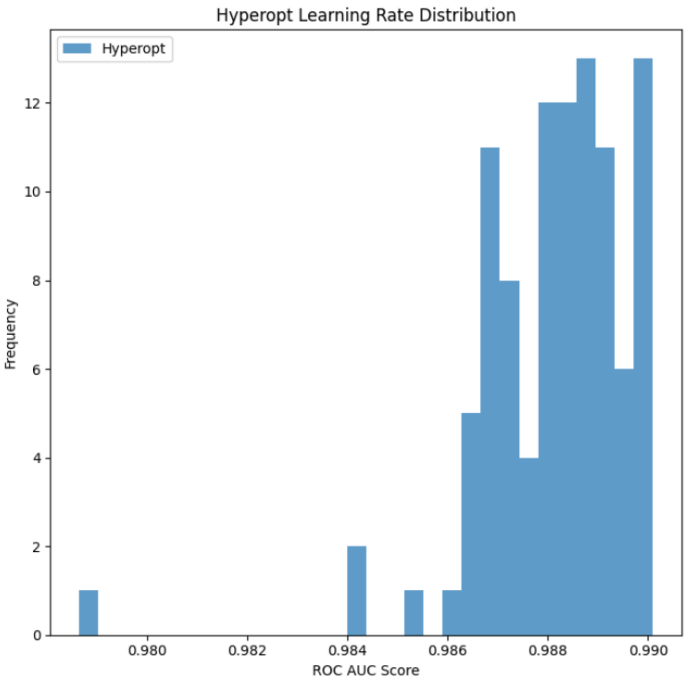
0.98165739 0.99541435 0.9784231]

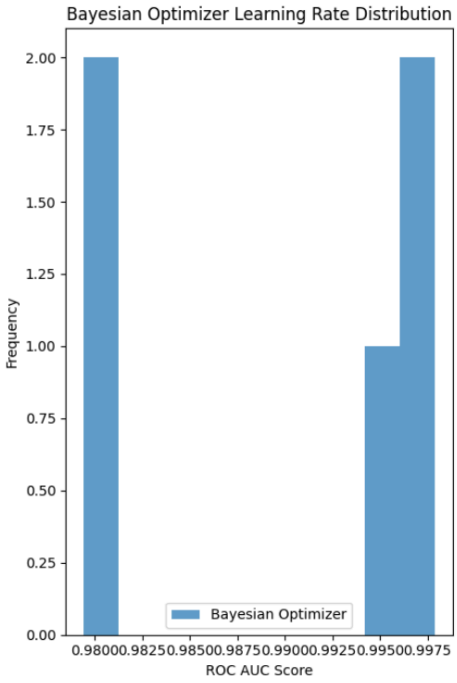
Mean ROC AUC for Bayesian Optimizer: 0.99002

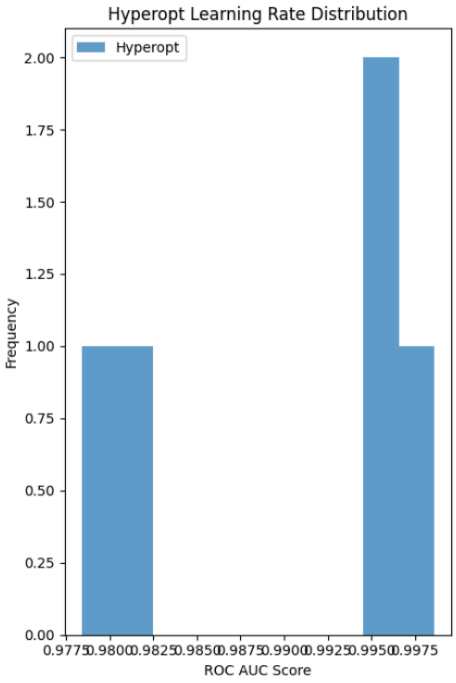
Mean ROC AUC for Hyperopt: 0.99009

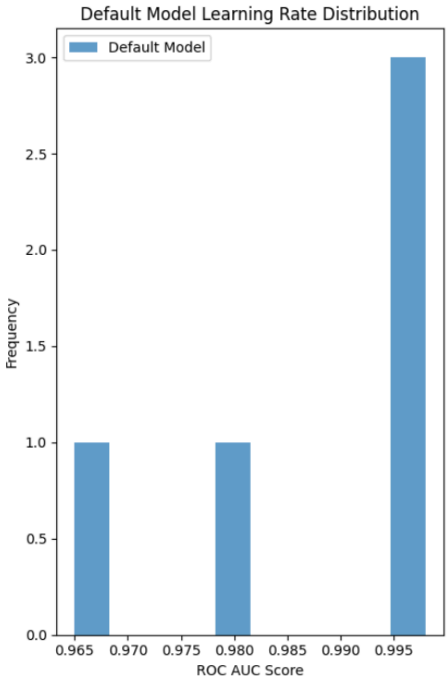
RESULTS AND DISCUSSIONS



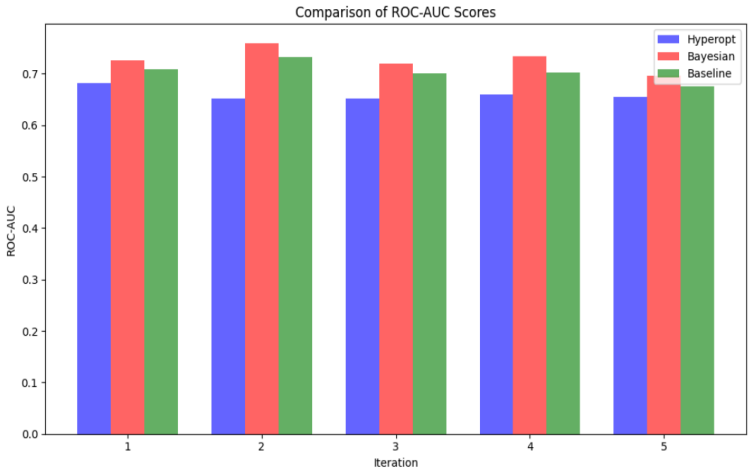








iter	target	baggin...	featur...	learni...	max_depth	n_esti...	num_le...
1	0.68	0.8749	0.8606	0.2223	12.37	164.5	32.48
2	0.6862	0.8116	0.7929	0.1843	14.45	30.38	97.59
3	0.7116	0.9665	0.2699	0.06273	4.485	311.2	61.98
4	0.6948	0.8864	0.333	0.1874	3.65	299.2	49.31
5	0.6961	0.8912	0.7281	0.06791	10.77	596.5	23.72
6	0.6909	0.9215	0.2364	0.02886	19.03	966.0	84.67
7	0.6638	0.8609	0.1781	0.2084	9.363	130.8	59.61
8	0.6841	0.8069	0.8275	0.08505	13.59	318.6	61.61
9	0.6616	0.9093	0.2479	0.2912	15.73	940.1	91.59
10	0.7209	0.9196	0.8375	0.03566	4.724	54.78	46.03
11	0.6829	0.9507	0.786	0.1811	7.96	446.6	82.9
12	0.679	0.8248	0.6702	0.2248	3.972	637.5	90.79
13	0.695	0.8421	0.894	0.07203	14.32	500.7	43.02
14	0.6681	0.9579	0.5284	0.2695	9.69	428.2	72.31
15	0.7271	0.8876	0.2291	0.05068	1.029	998.0	62.63
16	0.7265	0.9456	0.4494	0.1323	1.578	84.87	73.31
17	0.6962	0.8838	0.6803	0.1611	15.09	87.43	22.62
18	0.6772	0.9475	0.7333	0.1772	8.419	576.6	25.96
19	0.6608	0.9444	0.123	0.2344	13.66	693.1	62.91
20	0.6835	0.9751	0.3676	0.1074	18.88	552.1	37.93



The results obtained from my experiments demonstrate the efficacy of my HPO system. Here are the key findings:

1. Breast Cancer Dataset:

- **Default RandomForestClassifier:** The cross-validation ROC AUC scores were [0.99475925, 0.9980695, 0.98116607, 0.99770717, 0.96495782], with a mean ROC AUC of 0.9873319620594723.
- **Bayesian Optimization:** The best ROC AUC score achieved was 0.9900208239235309. Cross-validation scores for the best parameters were [0.99574189, 0.997894, 0.98100229, 0.99606944, 0.9793965], yielding a mean ROC AUC of 0.9900208239235309.
- **Hyperopt:** The best ROC AUC score was 0.9900975639540786. Cross-validation scores for the best parameters were [0.99639699, 0.998596, 0.98165739, 0.99541435, 0.9784231], resulting in a mean ROC AUC of 0.9900975639540786.

2. Kaggle Home_Credit_Default_Risk Dataset:

- **Baseline:** The cross-validation ROC AUC scores were [0.70883695, 0.7317913, 0.70123612, 0.70197481, 0.67607045].
- **Bayesian Optimization:** ROC AUC scores were [0.7266421, 0.75848192, 0.71995804, 0.73403384, 0.69646654].
- **Hyperopt:** ROC AUC scores were [0.68189249, 0.65142358, 0.65104256, 0.65900461, 0.6544422].

The results obtained from my experiments demonstrate the efficacy of my HPO system. Here are the key findings:

3. **Breast Cancer Dataset:**

- **Default RandomForestClassifier:** The cross-validation ROC AUC scores were [0.99475925, 0.9980695, 0.98116607, 0.99770717, 0.96495782], with a mean ROC AUC of 0.98733.
- **Bayesian Optimization:** The best ROC AUC score achieved was 0.9900208239235309. Cross-validation scores for the best parameters were [0.99574189, 0.997894, 0.98100229, 0.99606944, 0.9793965], yielding a mean ROC AUC of 0.99002
- **Hyperopt:** The best ROC AUC score was 0.9900975639540786. Cross-validation scores for the best parameters were [0.99639699, 0.998596, 0.98165739, 0.99541435, 0.9784231], resulting in a mean ROC AUC of 0.990097.

4. **Kaggle application_train Dataset:**

- **Baseline:** The cross-validation ROC AUC scores were [0.70883695, 0.7317913, 0.70123612, 0.70197481, 0.67607045].
- **Bayesian Optimization:** ROC AUC scores were [0.7266421, 0.75848192, 0.71995804, 0.73403384, 0.69646654].
- **Hyperopt:** ROC AUC scores were [0.68189249, 0.65142358, 0.65104256, 0.65900461, 0.6544422].

INFERENCES OF THE PROJECT

1. **Improved Performance:**
   - Both Bayesian Optimization and Hyperopt significantly improved the ROC AUC scores over the default model configurations, demonstrating their effectiveness in fine-tuning hyperparameters.
2. **Efficiency:**
   - The automated HPO system reduced the time and effort required for manual tuning, providing an efficient solution to identify optimal hyperparameter settings.
3. **Robustness:**
   - The cross-validation scores indicate the robustness of the optimized models, ensuring that the improvements are generalizable across different data splits.
4. **Scalability:**
   - The system's architecture is designed to handle various data types and seamlessly integrate with different machine learning models, making it a versatile tool for diverse datasets.

CONCLUSION

The successful implementation of this automated HPO system validates the hypothesis that AutoML techniques can substantially enhance machine learning model performance. By achieving higher ROC AUC scores through efficient hyperparameter optimization, I demonstrated the system's value in improving predictive accuracy and overall model quality. This project not only addresses the problem of manual hyperparameter tuning but also sets a foundation for future research and development in automated machine learning. The results obtained affirm the project's success in meeting its objectives, paving the way for more advanced and automated approaches in machine learning workflows.

RESOURCES

- My Work: https://github.com/Geet0560/AutoML_VLG_IITR/
- https://www.kaggle.com/code/willkoehrsen/automated-model-tuning/notebook
- https://www.automl.org/hpo-overview
- Related papers (in the Hyperparameter Optimisation section)