AUTOMATED HYPERPARAMETER OPTIMIZATION

VLG SUMMER OPEN PROJECT: AUTOML

By: Atharv Priyadarshi, Mechanical 1Y, Enroll no. : 23117034

Hyperparameter optimization (HPO) is an essential part of machine learning for maximizing model performance. This project develops an automatic HPO system using Random Search and custom Bayesian Optimization additionally compared with the popular Hyperopt library. The system works with Random Forest and Logistic Regression models - highlighting the power of auto HPO in finding the best hyperparameters. This project serves as a reminder that careful selection of the optimization method and choosing the right trade off between the different techniques, in the context of the dataset and model selected.

THE NEED OF HYPERPARAMETERS AND THEIR SELECTION ?

Machine learning models depend on hyperparameters, which are configuration variables set before the training process. These hyperparameters control various aspects of a model's learning process and structure, significantly impacting its ability to generalize to new data. The selection of optimal hyperparameters is crucial for achieving the best possible performance for a given model and dataset.

THE NEED OF AUTOMATION ?

Manual hyperparameter tuning is often tedious, time consuming and prone to human biases, leading to suboptimal results. Additionally, as the number of hyperparameters increases, the search space grows exponentially, making manual tuning practically infeasible. Automated HPO techniques address these challenges by systematically and efficiently searching for the best hyperparameter configurations.

THE PROJECT AIMS TO DEVELOP AN AUTOMATED HPO HAVING SEVERAL BENEFITS:

- It implements both Random Search and a custom Bayesian Optimization algorithm, providing a comparative analysis of these widely used optimization methods.
- It is designed to integrate with multiple machine learning models, including Random Forest and Logistic Regression which demonstrates its adaptability.
- The performance of the implemented techniques is compared against Hyperopt, to assess their relative effectiveness.

THE FOCUS OF THE PROJECT :

The project's primary focus is on binary classification tasks, using the ROC AUC score as the main evaluation metric. It handles tabular datasets and incorporates data preprocessing steps to ensure data compatibility with the chosen models.

DIRECTORY: (ARCHITECTURE)

```
AHO-AutomatedHyperparameterOptimiser/
├── data/
│   └── dataset.csv
├── src/
│   ├── data_preparation.py
│   ├── objective_function.py
│   ├── hyperparameter_optimization.py
│   ├── model_evaluation.py
│   ├── model_selection.py
│   └── visualization.py
├── main.py
├── .gitattributes
├── AHO_AutomatedHyperparameterOptimiser_Project_Report.pdf
├── README.md
└── requirements.txt
```

BASIC QUESTIONNAIRE:

1. **Why did I choose Random Search and Bayesian Optimization as the primary HPO techniques in the project?**
   We chose Random Search and Bayesian Optimization for several reasons:
   - They are widely used and well-established techniques for HPO, making them suitable for demonstrating the concepts of automated hyperparameter tuning.
   - Implementing both methods allows for a direct comparison of their performance and efficiency on the chosen dataset and models.
   - Bayesian Optimization is known for its efficiency, especially when evaluating the objective function is computationally expensive, as it intelligently samples the hyperparameter space.

2. **Why did I specifically choose Random Forest and Logistic Regression models for the project?**
   The choice of Random Forest and Logistic Regression was motivated by several factors:
   - Both models are widely used in machine learning and are known for their versatility across a range of classification problems. They represent distinct model types (ensemble method vs. linear model) allowing for a comparison of different approaches.
   - They have a diverse set of hyperparameters that can significantly impact performance, making them suitable for demonstrating the value of HPO.
   - Logistic Regression offers good interpretability, allowing for an understanding of the relationship between features and the target variable.
   - Random Forests are generally robust to outliers and noise in the data, making them a good choice for a wider range of datasets.

3. **How did you determine the appropriate hyperparameter search space for each model and optimization technique?**
   Defining the search space is a crucial aspect of HPO. We used a combination of the following strategies:
   - I started by examining the default hyperparameter values used in scikit-learn's implementations of Random Forest and Logistic Regression.
   - I reviewed literature and documentation on the hyperparameters for each model to understand their typical ranges and potential impact on performance.
   - I conducted some initial experiments with different ranges and values to get a better understanding of how the hyperparameters affected the model's behavior on our specific dataset.
   - I took into account the computational cost of evaluating different hyperparameter combinations and chose ranges that were computationally feasible to explore within a reasonable time frame.

METHODOLOGY:
The details of the implemented optimization techniques, the chosen machine learning models and the overall workflow of the automated HPO system.

- **RANDOM SEARCH:**
  Random Search, a simple yet effective HPO method, involves randomly sampling hyperparameters from a predefined distribution for a set number of iterations. This random sampling approach can be surprisingly effective, especially when dealing with lower-dimensional hyperparameter spaces. The advantage of Random Search lies in its ease of implementation and computational efficiency compared to more exhaustive techniques like Grid Search. However, Random Search might not be as efficient in finding the optimal hyperparameter configuration, especially when the number of iterations is limited or the search space is complex.
- **CUSTOM BAYESIAN OPTIMIZATION:**
  Bayesian Optimization offers a more intelligent and efficient approach to HPO. It constructs a probabilistic model, known as a surrogate model, of the objective function that maps hyperparameters to model performance. This model is then used to guide the search for optimal hyperparameters by selecting the most promising candidates for evaluation.
  - ➔ **Gaussian Process:**
    The project employs a Gaussian Process (GP) as the surrogate model. GPs provide a flexible and powerful way to model complex functions. They offer not only predictions of the objective function's value but also a measure of uncertainty associated with those predictions.
  - ➔ **Acquisition Function (Expected Improvement):**

To guide the search for optimal hyperparameters, the project uses the Expected Improvement (EI) acquisition function. EI balances exploration and exploitation by considering both the predicted performance and the uncertainty at each point in the hyperparameter space. It favors hyperparameter values that either promise significant improvements over the current best-observed performance or have high uncertainty, encouraging the algorithm to explore less-explored regions of the search space.
The mathematical formula for Expected Improvement (EI) is given by:
$EI(x) = (mu(x) - f(x\_best) - \xi) * \Phi(Z) + \sigma(x) * \phi(Z)$
Here, mu(x) represents the predicted mean performance, f(x_best) is the best performance score observed so far, ξ is the exploration parameter that controls the trade-off between exploration and exploitation, $\sigma(x)$ is the predicted standard deviation (uncertainty), and $\Phi(Z)$ and $\phi(Z)$ are the cumulative distribution function (CDF) and probability density function (PDF) of the standard normal distribution, respectively

- **HYPEROPT (FOR COMPARISON ONLY):**
  Hyperopt is incorporated into the project for benchmarking purposes. It provides a point of reference to assess the efficiency and effectiveness of the custom-built Random Search and Bayesian Optimization implementations. Hyperopt employs the Tree-structured Parzen Estimator (TPE) algorithm, a form of Bayesian Optimization known for its efficiency in exploring complex search spaces.
- **MODELS:**
  The project focuses on two machine learning models, each offering distinct advantages and suited for different types of datasets and classification tasks:

  ➢ **RANDOM FOREST:**
  Random Forest is a powerful ensemble learning method that combines multiple decision trees, each trained on a different random subset of the data. By aggregating the predictions of individual trees, Random Forests achieve high prediction accuracy and are less susceptible to overfitting compared to single decision trees.

  ➢ **LOGISTIC REGRESSION:**
  Logistic Regression is a linear model used for binary classification. It models the probability of a binary outcome using a sigmoid function. It is known for its simplicity, interpretability, and ability to handle both numerical and categorical features.
- **OVERALL PIPELINE:**
  The pipeline used in this project is as follows:
  1. **Data Loading and Preprocessing:** The process begins with loading the dataset using the functions defined in **data_preparation.py**. This module handles loading the data from a CSV file (named **dataset.csv**), splitting it into training and testing sets, and preprocessing the target variable to ensure compatibility with multi-class classification metrics.

2. **Model Selection:** The user selects the desired machine learning model (Random Forest or Logistic Regression). This choice dictates which set of hyperparameters will be optimized.
3. **Hyperparameter Space Definition:** Based on the chosen model, the user defines the hyperparameter search space. For Random Search, this involves specifying distribution (ranges or lists of possible values) for each hyperparameter. For Bayesian Optimization, it involves defining bounds (lower and upper limits) for continuous hyperparameters and lists of possible values for categorical ones.
4. **Optimization Technique Selection:** The user chooses the optimization technique (Random Search, Bayesian Optimization, or Hyperopt).
5. **Optimization Process:** The selected optimization technique is then used to search for the best hyperparameters. The **objective** function, defined in **objective_function.py,** is used to evaluate the performance of different hyperparameter combinations. This function trains and evaluates the chosen model using cross-validation and returns the average score (accuracy or ROC (Receiver Operating Characteristics) AUC (Area Under Curve))
6. **Best Hyperparameter Retrieval:** Once the optimization process is complete, the best hyperparameters found are retrieved.
7. **Model Training:** The chosen machine learning model is trained using the best hyperparameters found during the optimization process. The training is done using functions defined in **model_selection.py**, which handles the creation and training of specific models.
8. **Model Evaluation:** The trained model is then evaluated on a held-out test set using functions defined in **model_evaluation.py**. This module calculates various performance metrics, including ROC AUC and accuracy, and generates ROC curves for visualization.
9. **Result Visualization:** Finally, the **visualization.py** module provides functions to generate ROC curves, allowing for a visual comparison of performance of different models and hyperparameter settings.

IMPLEMENTATION:

The code for the automated HPO system is structured into several modules, each with a specific responsibility:

- **data_preparation.py:** Handles loading the data from a CSV file, splitting the data into training and test sets, and preprocessing the target variable for multi-class classification tasks.
- **hyperparameter_optimization.py:** Implements the core optimization algorithms, **random_search** and **bayesian_optimization**, including helper functions for sampling random parameters, encoding and decoding hyperparameters for use with the Gaussian Process, and calculating the Expected Improvement acquisition function.

- **objective_function.py:** Defines the **objective** function, which serves as the primary metric for evaluating the performance of different hyperparameter combinations during the optimization process.
- **model_evaluation.py:** Contains functions for evaluating the performance of trained models. It calculates metrics like accuracy and ROC AUC and includes code for generating ROC curves.
- **visualization.py:** Provides functions for plotting ROC curves, enabling visual comparison of model performance.
- **model_selection.py:** Contains functions for training different machine learning models, including **train_random_forest** and **train_logistic_regression**, allowing the user to switch between different model types.
- **main.py:** The main script orchestrates the entire HPO pipeline. It loads, data defines hyperparameter search spaces, runs the optimization algorithms, trains and evaluates models with the best hyperparameters found, and generates visualizations for comparison.

EXPLANATION OF THE CODES:
Here's a breakdown of the functions in each file of the project:

1. data_preparation.py:

- **load_and_preprocess_data(file_path):**
  Loads data from a CSV file, splits it into training and testing sets, and preprocesses the target variable to ensure compatibility with multi-class classification metrics.

2. objective_function.py:

- **objective(params, model_name, X_train, y_train):**
  Defines the objective function for hyperparameter optimization. It evaluates the performance of a model with a given set of hyperparameters using cross-validation. Then creates an instance of the selected model (RandomForestClassifier or LogisticRegression) using the params dictionary. It performs 3-fold cross-validation (cross_val_score) using either 'accuracy' (for multi-class) or 'roc_auc' (for binary) as the scoring metric. And finally, calculate and return the mean cross-validated score.

3. hyperparameter_optimization.py:

- **random_search(param_distributions, model_name, X_train, y_train, n_iter=50):**
  Implements the Random Search hyperparameter optimization algorithm. It samples random hyperparameters from the distributions specified in param_distributions. Then, evaluates the model performance for each set of sampled hyperparameters using the

objective function. It keeps track of the best hyperparameters and score found. At the end returns the best hyperparameters and score.

- **bayesian_optimization(param_bounds, model_name, X_train, y_train, n_iter=20, init_points=5):**
  Implements a custom Bayesian Optimization algorithm using a Gaussian Process and the Expected Improvement acquisition function. It initializes the Gaussian Process with random samples. Iteratively fits the Gaussian Process model and uses Expected Improvement to select the next hyperparameters to evaluate. Evaluates the model with the selected hyperparameters and updates the Gaussian Process. And returns the best hyperparameters and score found.

- **sample_random_params(param_bounds, model_name):**
  It samples random hyperparameters within the specified bounds, ensuring solver compatibility for Logistic Regression.

- **encode_params(params, param_bounds):**
  It encodes hyperparameters (normalizes continuous values and maps categorical values to indices) for use with the Gaussian Process.

- **decode_params(encoded_params, param_bounds):**
  It decodes encoded hyperparameters back to their original representations.

- **expected_improvement(x, gp, X_sample, y_sample, xi=0.01):**
  It calculates the Expected Improvement (EI) acquisition function value for a given hyperparameter set (x).

- **optimize_acquisition_function(gp, X_sample, y_sample, param_bounds, model_name, X_train, y_train):**
  Finds the hyperparameters that maximize the Expected Improvement using scipy.optimize.minimize.

4. model_evaluation.py:

- **evaluate_model(model, X_test, y_test, model_name):**
  It evaluates a trained model's performance on the test data using ROC AUC and accuracy. It makes predictions on the test data. Then, calculates the ROC AUC and accuracy scores. And returns a dictionary containing the results.

5. visualization.py:

- **plot_roc_auc(model, X_test, y_test, label):**
  It plots the ROC curve for a given model, handling both binary and multi-class classification cases.

- **compare_models(models, X_test, y_test):**
  It takes a list of models and plots their ROC curves on the same graph for comparison.

6. main.py:

- **Import Statements:**

  Imports all the necessary modules and functions from the custom modules (**data_preparation**, **hyperparameter_optimization**, **model_evaluation**, **visualization**, **model_selection**), as well as from external libraries like scikit-learn (RandomForestClassifier, DummyClassifier), matplotlib (pyplot), hyperopt, and random.

- **Configuration:**

  This section sets the MODEL_NAME to either 'RandomForest' or 'LogisticRegression', determining which model to optimize. Then defines N_ITER_RANDOM and N_ITER_BAYESIAN, the number of iterations for Random Search and Bayesian Optimization, respectively. Also, defines PARAM_DISTRIBUTIONS_RANDOM and PARAM_BOUNDS_BAYESIAN, dictionaries containing the hyperparameter distributions for Random Search and the bounds for Bayesian Optimization.

- **Data Loading:**

  This section calls **load_and_preprocess_data** to load the dataset from the specified file path and split it into training and testing sets.

- **Random Search:**

  It prints a message indicating the start of Random Search. Then, it calls **random_search** to perform optimization using Random Search. Thereafter, retrieves the best hyperparameters and score found by Random Search. It then prints the best hyperparameters and score.

- **Bayesian Optimization:**

  It prints a message indicating the start of Bayesian Optimization. Then, calls **bayesian_optimization** to perform optimization using the custom Bayesian Optimization implementation. Thereafter, retrieves the best hyperparameters and score found by Bayesian Optimization. It prints the best hyperparameters and score.

- **Train and Evaluate Models:**
  Trains three models:

optimized_model: Trained using the best hyperparameters found by Bayesian Optimization.
default_model: A default RandomForestClassifier with default hyperparameters (for comparison).
dummy_model: A DummyClassifier that always predicts the most frequent class (as a baseline).

It evaluates the three models using **evaluate_model** and stores the results. Then, prints the evaluation results for all three models.

- **Hyperopt for Comparison:**
  At first, it prints a message indicating that Hyperopt is running. It defines the **objective_hyperopt** function, which is the objective function to be optimized by Hyperopt. Thereafter, defines the space dictionary, representing the hyperparameter search space for Hyperopt. Runs Hyperopt using fmin, passing the objective function, search space, and the TPE algorithm. Then, retrieve the best hyperparameters and score found by Hyperopt. At the end, prints the best hyperparameters and score from Hyperopt.

- **Visualize Comparison:**
  Calls **compare_models** to plot and display the ROC curves of the dummy model, the default Random Forest model, and the optimized model.
  Purpose of main.py:
  Acts as the main script that orchestrates the entire hyperparameter optimization and model evaluation process.
  Loads data, sets configurations, runs optimization algorithms, trains and evaluates models, and visualizes the results.
  Provides a structured way to compare the performance of different optimization techniques and models.

RESULTS AND DISCUSSION:

- **Dataset:** The dataset used in the project:
  - ❖ Number of samples: 150
  - ❖ Number of features: 3
  - ❖ Target Variable: It is categorical with three distinct classes (0, 1 and 2)
- **Performance Comparison: (This performance although varies every time the project is run, so i am analyzing on of the output generated)**
  - ➔ The output achieved by running the project: (model wise)
    - ➔ Random Forest:
      - ➢ Random Search:
        - ➔ Best Hyperparameters (Random):
          {'n_estimators': 167,
          'max_depth': 8,
          'min_samples_split': 4,

        'min_samples_leaf': 2}
    - Best Score (Random): 0.8125

- Bayesian Optimization:
    - Best Hyperparameters (Bayesian):
      {'n_estimators': 103,
      'max_depth': 11,
      'min_samples_split': 3,
      'min_samples_leaf': 3}
    - Best Score (Bayesian): 0.7812

- Optimized RandomForest Results:
  {'roc_auc': 0.9184027777777777,
  'accuracy': 0.76}

- Default RandomForest Results:
  {'roc_auc': 0.9256944444444445,
  'accuracy': 0.8}

- Dummy Classifier Results:
  {'roc_auc': 0.5,
  'accuracy': 0.48}

- Hyperopt:
    - Best Hyperparameters (Hyperopt):
      {'max_depth': 8.0,
      'min_samples_leaf': 1.0,
      'min_samples_split': 3.0,
      'n_estimators': 148.0}
    - Best Score (Hyperopt): 0.8125

- Logistic Regression:
    - Random Search:
      Best Hyperparameters (Random):
      {'C': 49548.663312290955,
      'penalty': 'l2',
      'solver': 'saga'}
      Best Score (Random): 0.8229

    - Bayesian Optimization:
      Best Hyperparameters (Bayesian):
      {'C': 82856.84752744153,
      'penalty': 'l1',

'solver': 'saga'}
Best Score (Bayesian): 0.8229

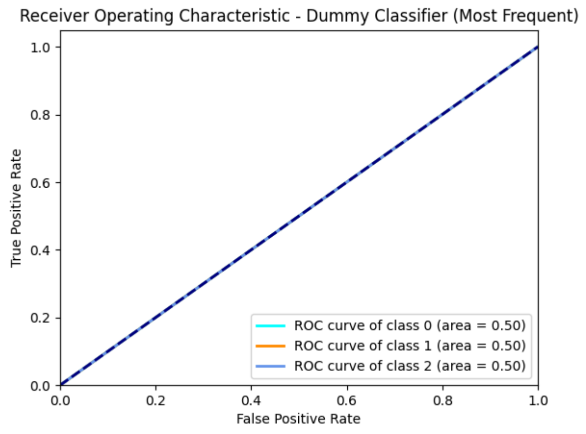
- ➢ Optimized LogisticRegression Results:
  {'roc_auc': 0.9444444444444443,
  'accuracy': 0.76}

- ➢ Default RandomForest Results:
  {'roc_auc': 0.9256944444444445,
  'accuracy': 0.8}

- ➢ Dummy Classifier Results:
  {'roc_auc': 0.5,
  'accuracy': 0.48}

- ➢ Hyperopt:
  - ➔ Best Hyperparameters (Hyperopt):
    {'C': 13912.093475886066,
    'penalty': 0,
    'solver': 1}
  - ➔ Best Score (Hyperopt): 0.8229

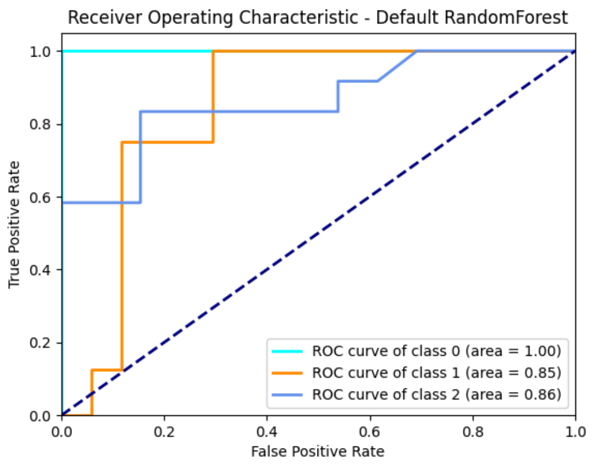
- **ROC CURVE ANALYSIS:**
  - ➔ **Case when the model selected is 'RandomForest':**
    1. **DUMMY CLASSIFIER:**



- **Diagonal Lines:** The ROC curves for all three classes ('0', '1' and '2') are almost perfectly diagonal.
- **AUC = 0.50:** All three classes have an AUC of 0.50, which is the expected performance of a random classifier.

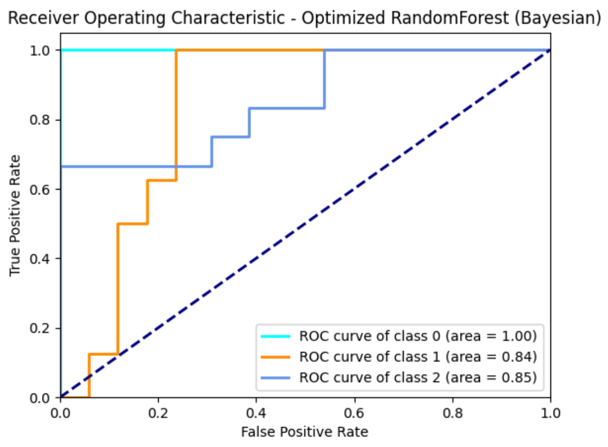
- **Interpretation:** This confirms that the dummy classifier, which simply predicts the most frequent class, does not have any real discriminatory power.

2. **DEFAULT RandomForest:**



- **Three Curves:** We have three ROC curves, one for each class (0, 1, and 2) in the dataset.
- **Class 0:** The curve for class 0 hugs the top-left corner, indicating excellent performance. The AUC for class 0 is 1.00, which means it perfectly separates this class from the others.
- **Class 1 and 2:** The curves for class 1 and 2 are also above the diagonal line (random classifier baseline), showing good performance. The AUC values are 0.85 and 0.86, respectively.
- **Overall:** The default RandomForest model seems to perform very well on the dataset, especially for class 0.

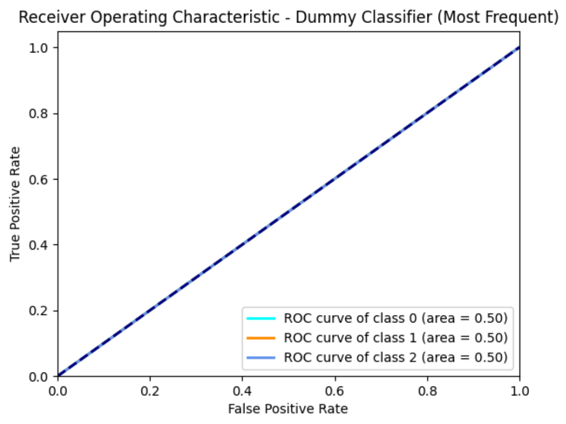
3. **Optimized RandomForest (Bayesian):**



- **Three Curves:** Similar to the default RandomForest, we have three curves, one for each class.
- **Class 0:** The curve for class 0 again shows perfect separation (AUC = 1.00).
- **Class 1 and 2:** The curves for class 1 and 2 are very close to the curves of the default RandomForest, with similar AUC values (0.84 and 0.85, respectively).
- **Comparison:** The optimized RandomForest (using Bayesian Optimization) has almost the same performance as the default RandomForest, suggesting that the Bayesian Optimization might not have found significantly better hyperparameters than the default settings.

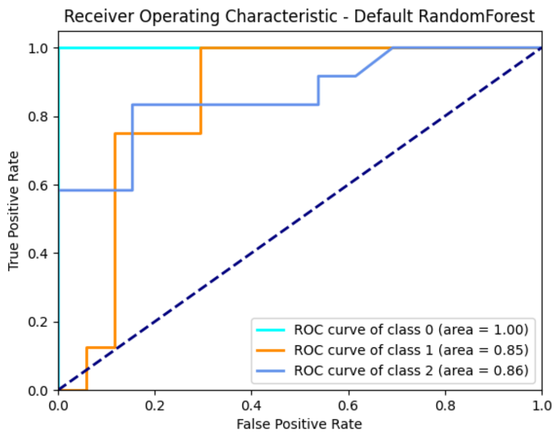
➔ **Case when the model selected is 'Logistic Regression':**.

1. **DUMMY CLASSIFIER:**



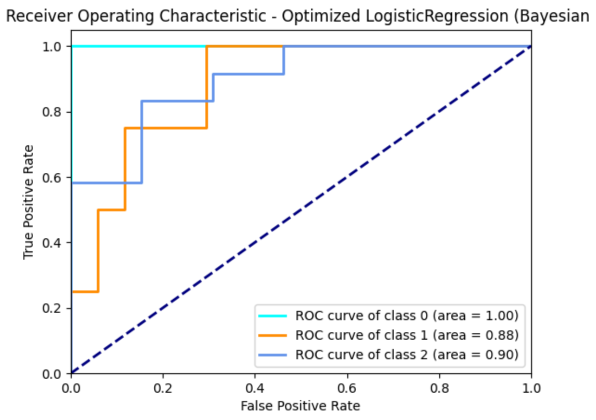
- This figure remains the same, representing the baseline performance of a random classifier (diagonal lines with AUC = 0.50 for all classes).

2. DEFAULT RANDOM FOREST:



- This figure is also the same as before since we a're still using the default RandomForest as a baseline.
- The curves indicate excellent performance for class 0 (AUC = 1.00) and good performance for class 1 (AUC = 0.85) and class 2 (AUC = 0.86).

3. OPTIMIZED LOGISTIC REGRESSION:



- **Class 0:** The optimized Logistic Regression model also achieves perfect separation for class 0 (AUC = 1.00), similar to the Random Forest models.
- **Class 1:** The model performs very well for class 1, with an AUC of 0.88. This is slightly better than the default RandomForest (AUC = 0.85).
- **Class 2:** The performance for class 2 is good, with an AUC of 0.90. This is again slightly better than the default RandomForest (AUC = 0.86).
- **Overall:** The ROC curves suggest that the optimized Logistic Regression model performs comparably to the Random Forest models, with potentially a slight edge for classes 1 and 2.

- **KEY POINTS:**
  1. Both Random Forest and Logistic Regression models, even with default hyperparameters, demonstrate strong performance on the given dataset, especially for class 0, which is perfectly separable.
  2. The Bayesian Optimization process seems to have improved the performance of the Logistic Regression model, particularly for classes 1 and 2, bringing it closer to or slightly better than the default Random Forest.

- **METHODS TO EXPAND:**
  - We can include a wider variety of models beyond Random Forest and Logistic Regression. They can be:
    1. Support Vector Machines (SVMs): Powerful for high-dimensional data.
    2. Gradient Boosting Machines (GBMs): Often achieve state-of-the-art performance (XGBoost, LightGBM, CatBoost).
  - We can explore multi-layer perceptrons (MLPs) or even deeper architectures for complex datasets.
  - Techniques like meta-learning or automated stacking to recommend the most promising models based on dataset characteristics can be used.
  - More Sophisticated Bayesian Optimization:
    1. We can explore other acquisition functions like Probability of Improvement (PI), Upper Confidence Bound (UCB), or Thompson Sampling.
    2. Experimenting with different kernels (e.g., Matern, Rational Quadratic) for the Gaussian Process may also help.
    3. Pruning techniques to stop unpromising evaluations early, saving computational time can be implemented.
    4. We can consider Genetic Algorithms or Evolutionary Strategies, which can be effective for complex, non-convex optimization landscapes.
  - Data Preprocessing and Feature Engineering:
    1. We can use automated feature engineering libraries (e.g., Featuretools) to generate new features and improve model performance.
    2. Feature selection techniques (e.g., SelectKBest, RFE) to identify the most important features and potentially reduce dimensionality can be incorporated.
  - Performance Evaluation and Visualization:
    1. We can include more relevant evaluation metrics beyond ROC AUC and accuracy. For example:
       Precision, Recall, F1-score (for binary classification)
       Confusion Matrix, Classification Report (for multi-class)
       Log Loss, Brier Score (for probabilistic predictions)
    2. Libraries like Plotly or Bokeh to create interactive visualizations of the hyperparameter search space and the optimization process can be used.
  - We can combine multiple optimized models using ensemble methods (e.g., voting, stacking) to improve overall performance and robustness.

    1. We can parallelize the hyperparameter optimization process to speed up evaluation, especially for computationally intensive models.

- We can create a user-friendly web interface for your system, allowing users to easily upload data, select models and optimization techniques, and visualize results.
- Techniques like SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-Agnostic Explanations) to understand the decisions made by your optimized models can be used.

----------THANK YOU----------