**VLG OPEN PROJECT REPORT**

Name: Sanya Jain

Enrollment number: 22115138

Branch: Electrical engineering

OBJECTIVE:

The objective of this project was to develop an automated Hyperparameter Optimization (HPO) system using AutoML techniques. This system aimed to efficiently identify optimal hyperparameter configurations for machine learning models, tailored to specific datasets and tasks. By leveraging algorithms like Bayesian Optimization and Random Search, the system navigates complex hyperparameter spaces to maximize model performance metrics such as accuracy and precision. Through iterative experimentation and adaptive learning, it dynamically refines its search strategy, ensuring robust and scalable optimization. The HPO system addresses the challenge of manual hyperparameter tuning, empowering practitioners to achieve superior model performance with reduced effort and time investment.

TECHNIQUE USED:

In this project, we employed Bayesian Optimization for hyperparameter tuning. Bayesian Optimization is a sequential model-based optimization technique that uses probabilistic models to predict the performance of different hyperparameter configurations.

Bayesian Optimization works by constructing a surrogate model of the objective function (e.g., model accuracy) that approximates the true performance based on observed hyperparameter values. It uses Bayesian inference to update this surrogate model iteratively as more evaluations are performed.

Advantages of Bayesian Optimization over other methods include:

1. **Efficiency**: It typically requires fewer evaluations of the objective function compared to exhaustive grid search or random search, as it intelligently selects promising hyperparameter configurations.
2. **Global Optima**: Bayesian Optimization focuses on finding the global optimum rather than getting stuck in local optima, making it suitable for complex and multimodal hyperparameter spaces.
3. **Adaptive**: It dynamically adjusts the search based on previous evaluations, effectively balancing exploration (searching new areas) and exploitation (exploiting known good areas).
4. **Handles Noise**: It can handle noisy evaluations of the objective function by modeling uncertainty, which makes it robust in real-world scenarios where objective function evaluations may vary.
5. **Flexibility**: Bayesian Optimization can accommodate different types of hyperparameters (continuous, discrete, categorical) and can handle constraints or dependencies between hyperparameters, making it versatile for various machine learning models.

Overall, Bayesian Optimization stands out as a powerful technique for hyperparameter tuning in machine learning, offering a principled and efficient approach to maximizing model performance with minimal computational resources.

EXPLANATION OF THE CODE:

Dataset used: https://www.kaggle.com/datasets/akshaydattatraykhare/diabetes-dataset

Data processing

The code defines a function load\_and\_preprocess\_data that streamlines data loading, preprocessing, and splitting, ensuring datasets are ready for model training and evaluation.**Loading Data**: It reads a dataset from a CSV file specified by data\_path using Pandas.

* **Handling Target Column**: If target\_column is not provided or not found in the dataset, it defaults to using the last column as the target variable.
* **Feature Extraction**: Separates the dataset into features (X) and the target (y).
* **Identifying Feature Types**: Identifies numeric and categorical features based on their data types.
* **Defining Transformers**: Sets up preprocessing pipelines for numeric (imputes missing values and scales) and categorical (imputes and encodes) features.
* **Applying Transformations**: Uses ColumnTransformer to apply the defined transformers to their respective feature types (num for numeric and cat for categorical).
* **Train-Test Split**: Splits the preprocessed data into training and testing sets using train\_test\_split.
* **Output**: Returns X\_train, X\_test, y\_train, and y\_test for further machine learning tasks.

Models and their hyperparameters space definition

The code defines a function define\_model\_and\_hyperparameters that sets up a specified machine learning model and its corresponding hyperparameter search space. This function streamlines the process of model selection and hyperparameter tuning for various machine learning tasks.

* Model Selection:The function takes an argument model\_type, which specifies the type of model to be created. It supports a variety of models, including logistic regression, decision tree, random forest, neural network, support vector machine (SVM), and gradient boosting, both for classification and regression tasks.
* Model Initialization:Based on the model\_type provided, the function initializes the appropriate model from scikit-learn. Each model type is associated with specific hyperparameters that need to be tuned for optimal performance.
* Hyperparameter Definition:For each model type, the function defines a dictionary of hyperparameters and their respective ranges or values to explore during hyperparameter tuning. These hyperparameters are critical for adjusting the model's behavior and improving its performance.
* Return Values:The function returns a tuple containing the initialized model and the hyperparameter dictionary. These can be used directly in model training and hyperparameter optimization processes, such as grid search or random search.

BAYESIAN OPTIMISATION IMPLEMENTATION

An objective\_function is defined that is used to optimize hyperparameters for various machine learning models. The optimization aims to maximize model accuracy by minimizing the negative cross-validation score. Here's a detailed breakdown of how the code works:

1. Imports:
   * numpy: For numerical operations.
   * scipy.stats.norm: Used for normal distribution functions.
   * scipy.optimize.minimize: For optimization routines.
   * GaussianProcessRegressor, Matern: For Gaussian process regression, which is often used in Bayesian optimization.
   * RandomForestClassifier: A machine learning model from scikit-learn.
   * cross\_val\_score: For performing cross-validation.
2. Objective Function:
   * The objective\_function takes a dictionary params that contains hyperparameters for a specific model.
   * The function then initializes a machine learning model based on the model\_type and the provided params.
3. Model Selection:
   * Depending on the model\_type, different models are initialized with the corresponding hyperparameters extracted from params:
     + RandomForestClassifier for 'random\_forest'.
     + LogisticRegression for 'logistic'.
     + DecisionTreeClassifier for 'decision\_tree'.
     + RandomForestRegressor for 'random\_forest\_regression'.
     + MLPClassifier for 'neural\_network'.
     + MLPRegressor for 'neural\_network\_regression'.
     + SVC for 'svm'.
     + SVR for 'svm\_regression'.
     + GradientBoostingClassifier for 'gradient\_boosting'.
     + GradientBoostingRegressor for 'gradient\_boosting\_regression'.

Each model type corresponds to a set of hyperparameters specific to that model.

1. Hyperparameters:
   * The hyperparameters are extracted from the params dictionary and used to initialize the model. These parameters vary based on the model type, such as criterion, max\_depth, max\_features, etc., for RandomForestClassifier.
2. Cross-Validation:
   * The cross\_val\_score function performs k-fold cross-validation (with cv=5 indicating 5 folds) on the initialized model using the training data (X\_train and y\_train).
   * The scoring='accuracy' parameter specifies that accuracy is used as the metric for evaluation.
3. Score Calculation:
   * The mean accuracy score from the cross-validation is computed.
   * The function returns the negative of this mean score (-score). This is done because the optimization routine (minimize) aims to minimize the objective function, and minimizing the negative accuracy is equivalent to maximizing the accuracy.

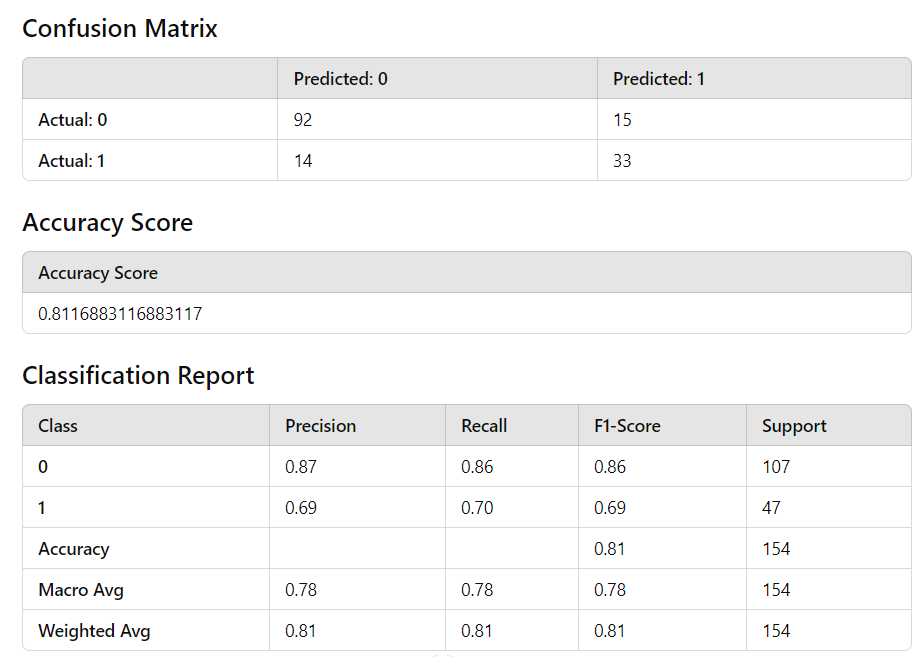
Next, it performs Bayesian optimization to find the best hyperparameters for a given machine learning model. The goal is to maximize model performance, in this case by minimizing the negative cross-validation accuracy score. Below is a detailed breakdown of how the code works:

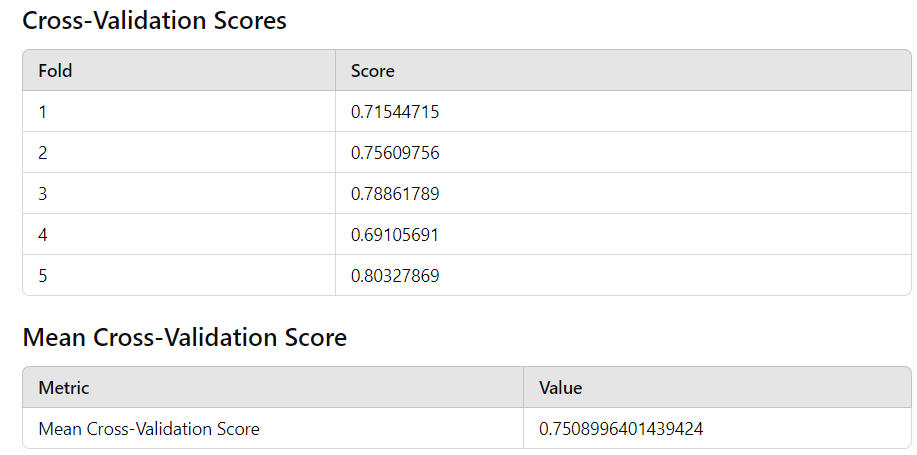
1. Encoding and Decoding Functions:
   * encode\_params(params, space): Converts hyperparameter values to their corresponding indices or values in a numerical array, which is necessary for optimization algorithms.
   * decode\_params(encoded\_params, space): Converts the numerical array back to the original hyperparameter values.
2. Sampling Hyperparameters:
   * sample\_space(space): Randomly samples hyperparameter values from the defined search space. It can handle lists, arrays, or tuples that define ranges for continuous values.
3. Bayesian Optimization Function:
   * bayesian\_optimization(objective\_function, space, n\_iter=100, n\_init=10, xi=0.01):
     + Initialization:
       - n\_iter: Number of iterations for optimization.
       - n\_init: Number of initial random samples.
       - xi: Exploration-exploitation trade-off parameter for the acquisition function.
       - Initially samples n\_init sets of hyperparameters using sample\_space and evaluates the objective\_function for each to generate initial data (X for hyperparameters, y for objective values).
     + Gaussian Process Model:
       - Uses a Gaussian Process Regressor with a Matern kernel to model the objective function.
       - ei(x, gp, y\_best, xi): Expected improvement acquisition function, guiding the selection of the next sample point by balancing exploration and exploitation.
     + Optimization Loop:
       - Iteratively updates the Gaussian Process model with new samples.
       - In each iteration, fits the GP model to current data (X and y).
       - Evaluates the acquisition function (ei) to determine the next point to sample.
       - Samples 1000 random points, evaluates ei, and selects the point with the highest ei value.
       - Evaluates the objective\_function at this new point and updates X and y with the new data.
     + Result:
       - After n\_iter iterations, finds and returns the best set of hyperparameters (best\_params) and the corresponding objective value (best\_value).

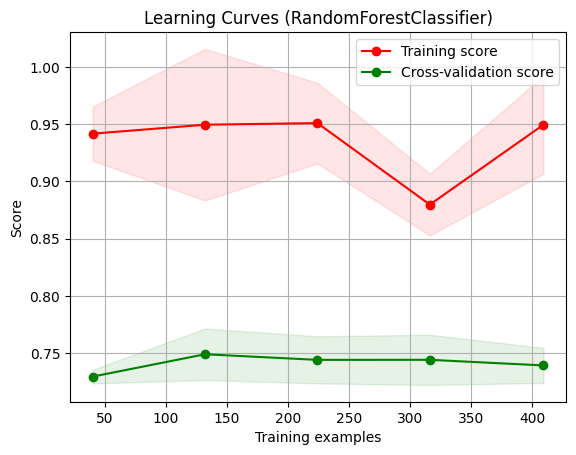
Key Components of Bayesian optimisation implementation

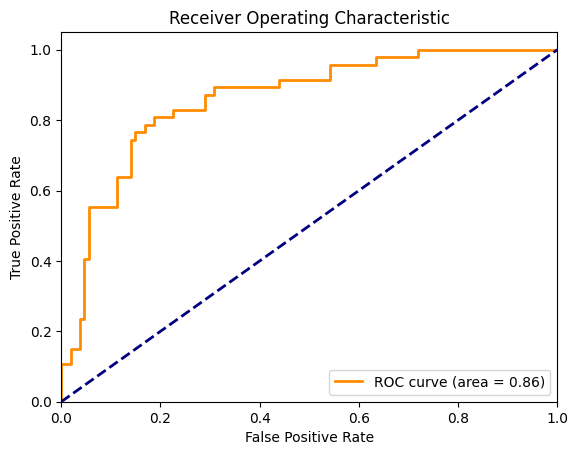
1. Initialization of Sample Points:The code starts by generating n\_init initial samples of hyperparameters and evaluating the objective function to create the initial dataset for the Gaussian Process model.
2. Gaussian Process Regression:A powerful tool for Bayesian optimization, it provides not only predictions but also uncertainty estimates for those predictions, which are used by the acquisition function.
3. Expected Improvement (EI):This acquisition function is used to decide where to sample next. It considers both the potential improvement over the current best result and the uncertainty of the predictions, allowing for a balance between exploring new areas of the search space and exploiting known good areas.
4. Bayesian Optimization Loop:The loop repeatedly fits the Gaussian Process to the current data, selects the next point based on the acquisition function, and updates the data with the new sample. This iterative process continues for n\_iter iterations, refining the hyperparameter search.
5. Finding the Best Parameters:After the optimization process, the best hyperparameters and their corresponding objective value are identified and returned.

COMPARITIVE ANALYSIS(RANDOM SEARCH VS BAYESIAN VS HYPEROPT SCORES)

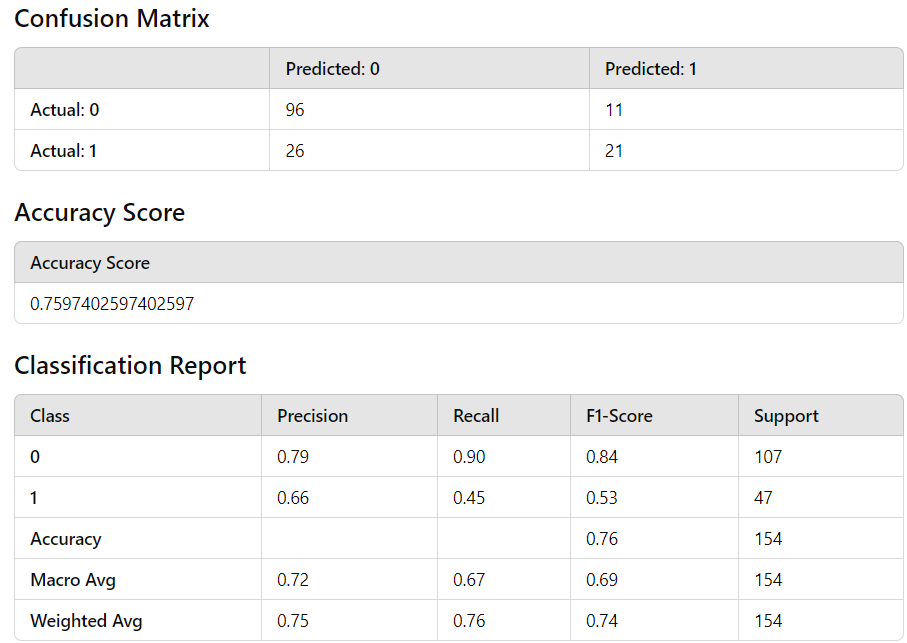
RANDOM SEARCH

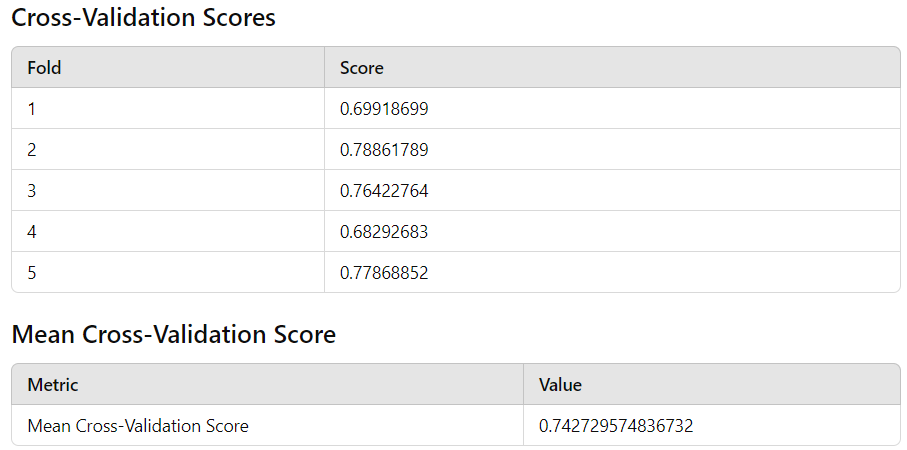


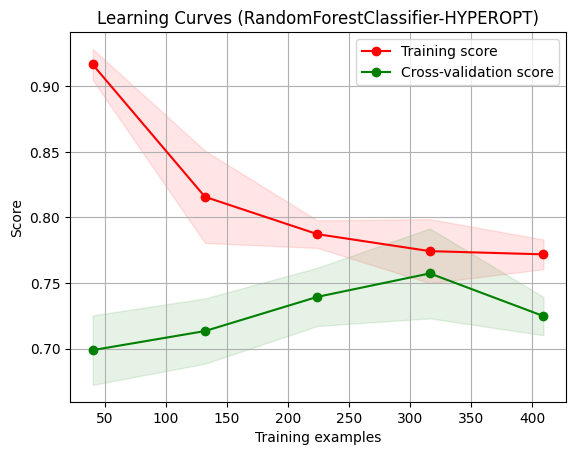


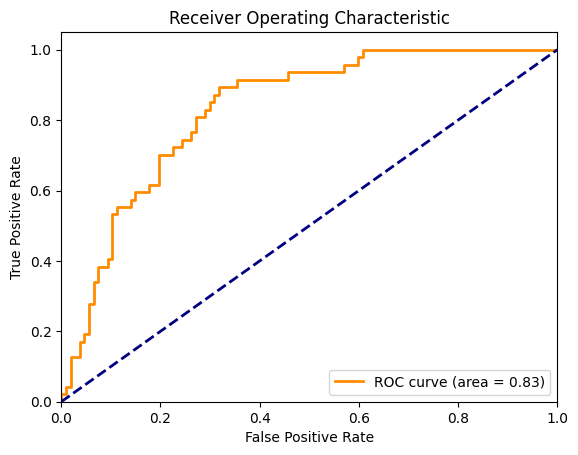


Hyperopt

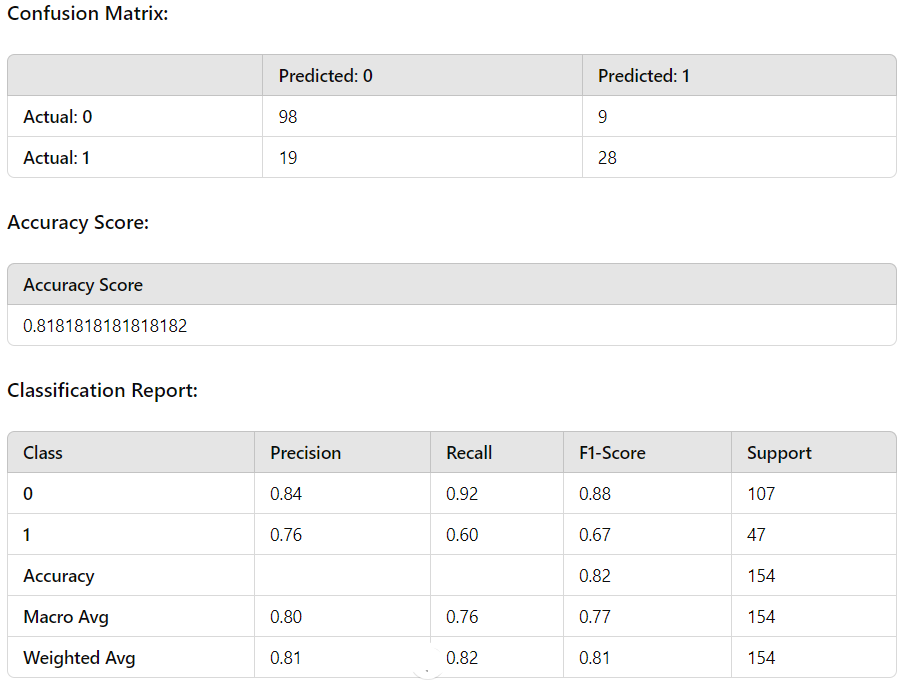


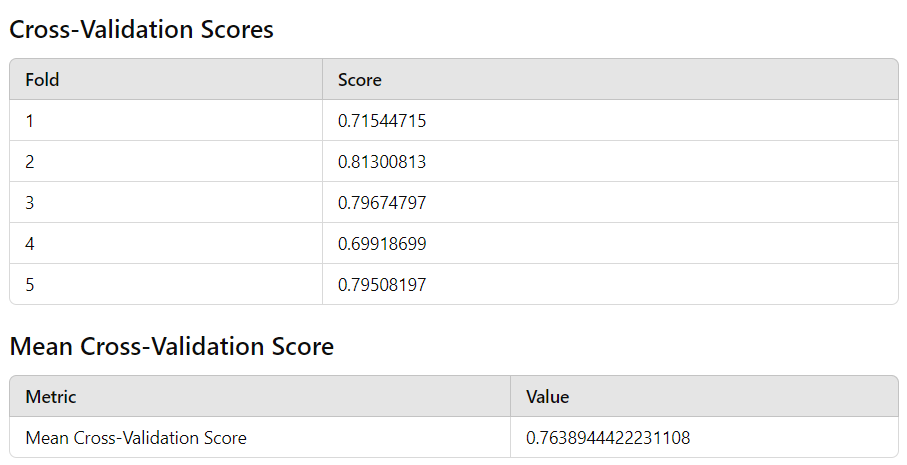




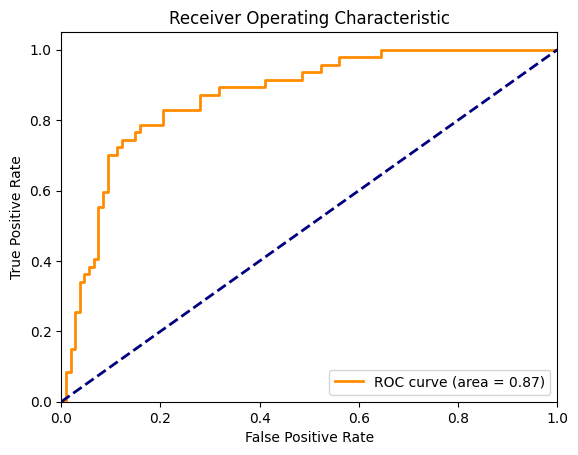


Bayesian









Conclusion

This implementation of Bayesian optimization provides a structured and efficient method for hyperparameter tuning. By leveraging the Gaussian Process model and the expected improvement acquisition function, it intelligently navigates the hyperparameter space to find the optimal settings for the specified machine learning model, significantly improving model performance.