**Physics Machine Learning Project Documentation**

**Table of Contents**

1. **Project Overview**
2. **Directory Structure and File Organization**
3. **File Descriptions**
4. **Model Selection and Training Process**
5. **Data Pre-processing**
6. **Hyperparameter Tuning**
7. **Model Evaluation Metrics and Formulas**
8. **Detailed Explanation of Optimization and Assessment**

**1. Project Overview**

The goal of this project is to build a machine learning regression model capable of making accurate predictions on the provided dataset. This project is designed to evaluate multiple regression models, optimize their hyperparameters, and assess their performance through various metrics. The models undergo hyperparameter tuning via grid search, and detailed metrics are recorded for performance evaluation, which includes error, accuracy, and intermediate data for further analysis.

**2. Directory Structure and File Organization**

Here is the structure of our project directory:

/physics\_ml\_project

├── amal\_mp\_project\_venv/ # Virtual environment for project dependencies

├── best\_hyperparameters.json # Best hyperparameters after tuning

├── config.yaml # Configuration file with initial settings

├── launch.py # Main script to execute training and evaluation

├── model\_metrics.json # JSON file containing evaluation metrics

├── physics\_ml\_<jobID>.err # Error log for each job

├── physics\_ml\_<jobID>.out # Output log for each job

├── requirements.txt # List of Python dependencies

├── slurm\_submit.sh # SLURM job submission script

├── src/ # Source code folder

│ ├── data\_processor.py # Data loading and preprocessing

│ ├── distributed\_trainer.py # Distributed training functions

│ ├── models.py # Model training and evaluation functions

│ ├── plot\_utils.py # Utility functions for plotting

└── updated\_data\_set\_ab2\_abc\_ab.xlsx # Input dataset

**3. File Descriptions**

**3.1 launch.py**

This is the main script that coordinates the loading of data, training of models, hyperparameter tuning, and evaluation. It pulls configurations from config.yaml, initializes the models and data, and triggers the grid search for hyperparameter tuning.

**3.2 slurm\_submit.sh**

This is a SLURM script used to submit the job to the HPC cluster. It specifies the resources needed (such as CPUs, GPUs, and memory) and activates the virtual environment before running launch.py.

**3.3 src/data\_processor.py**

This module is responsible for data loading and preprocessing. It loads data from the specified Excel file and performs data type checks, converts categorical variables as necessary, and splits the data into training, validation, and test sets.

**3.4 src/distributed\_trainer.py**

This file handles distributed training for scaling the model training across multiple devices. It ensures that processes are synchronized and distributed computation is efficiently managed.

**3.5 src/models.py**

This file contains the definitions of the regression models and includes functions for training and evaluating the models. It also sets up the grid search for hyperparameter tuning.

**3.6 src/plot\_utils.py**

This utility module provides functions to generate plots such as learning curves and residual plots, aiding in visual inspection of model performance. These plots are saved to the project directory for future reference.

**3.7 best\_hyperparameters.json**

This JSON file stores the best hyperparameters found after the grid search, allowing easy reference and reproducibility.

**3.8 model\_metrics.json**

This JSON file contains various evaluation metrics (like Mean Squared Error and R²) for the final selected model. It’s essential for assessing model accuracy and error.

**4. Model Selection and Training Process**

**Models Considered**

We focused on regression models for this project. These models are specifically designed to predict continuous target values based on input features. The models used in this project are:

1. **XGBoost Regressor**: An efficient and scalable implementation of gradient-boosting algorithms, which is particularly well-suited for structured/tabular data.
2. **Random Forest Regressor**: An ensemble method that builds multiple decision trees and merges them to get a more accurate and stable prediction.
3. **Support Vector Regressor (SVR)**: A regression variant of Support Vector Machine, suitable for linear and non-linear regressions.

**Training Procedure**

1. **Data Splitting**: The data is divided into training, validation, and test sets. The model is trained on the training set, hyperparameter tuning is done on the validation set, and final evaluation is performed on the test set.
2. **Grid Search for Hyperparameters**: Each model undergoes hyperparameter optimization through grid search. Different combinations of hyperparameters are tried, and the best configuration is selected based on performance on the validation set.
3. **Evaluation**: The selected model is evaluated on both validation and test sets using multiple metrics.

**5. Data Pre-processing**

The dataset is preprocessed in data\_processor.py:

1. **Data Loading**: Reads data from the Excel file.
2. **Type Handling**: Ensures all columns are either numeric or categorical. Non-numeric columns that cannot be converted are removed.
3. **Splitting**: The dataset is split into training (60%), validation (20%), and test (20%) sets.

**6. Hyperparameter Tuning**

**Hyperparameters Considered**

1. **XGBoost Regressor**:
   * n\_estimators: Number of boosting rounds.
   * learning\_rate: Step size at each iteration.
   * max\_depth: Maximum tree depth.
   * min\_child\_weight: Minimum sum of instance weight needed in a child.
2. **Random Forest Regressor**:
   * n\_estimators: Number of trees in the forest.
   * max\_depth: Maximum depth of the trees.
   * min\_samples\_split: Minimum number of samples required to split a node.
   * min\_samples\_leaf: Minimum number of samples required at a leaf node.
3. **Support Vector Regressor (SVR)**:
   * C: Regularization parameter.
   * epsilon: Width of the epsilon-tube within which no penalty is associated.
   * kernel: Specifies the kernel type to be used in the algorithm.

**Hyperparameter Optimization**

Grid search is applied to each model. This process involves:

* Exhaustively testing each combination of hyperparameters.
* Using cross-validation to assess performance for each combination.
* Selecting the combination that yields the lowest Mean Squared Error (MSE) on the validation set.

**7. Model Evaluation Metrics and Formulas**

**Metrics Used**

1. **Mean Squared Error (MSE)**: Measures the average squared difference between predictions and actual values.

MSE=1n∑i=1n(yi−y^i)2MSE = \frac{1}{n} \sum\_{i=1}^{n} (y\_i - \hat{y}\_i)^2MSE=n1​i=1∑n​(yi​−y^​i​)2

where yiy\_iyi​ is the actual value, y^i\hat{y}\_iy^​i​ is the predicted value, and nnn is the number of observations.

1. **Root Mean Squared Error (RMSE)**: Square root of MSE, providing an error metric in the same units as the target variable.

RMSE=MSERMSE = \sqrt{MSE}RMSE=MSE​

1. **Mean Absolute Error (MAE)**: Measures the average absolute error between predictions and actual values.

MAE=1n∑i=1n∣yi−y^i∣MAE = \frac{1}{n} \sum\_{i=1}^{n} |y\_i - \hat{y}\_i|MAE=n1​i=1∑n​∣yi​−y^​i​∣

1. **R² Score (Coefficient of Determination)**: Indicates the proportion of the variance in the dependent variable that is predictable from the independent variables.

R2=1−∑i=1n(yi−y^i)2∑i=1n(yi−yˉ)2R^2 = 1 - \frac{\sum\_{i=1}^{n} (y\_i - \hat{y}\_i)^2}{\sum\_{i=1}^{n} (y\_i - \bar{y})^2}R2=1−∑i=1n​(yi​−yˉ​)2∑i=1n​(yi​−y^​i​)2​

where yˉ\bar{y}yˉ​ is the mean of the actual values.

**Explanation of Metrics**

* **MSE and RMSE**: Useful for measuring average prediction error. RMSE is especially valuable when we want to understand the error in the same units as the target variable.
* **MAE**: Provides an average magnitude of errors, which can sometimes be more interpretable than MSE.
* **R² Score**: Helps in understanding the proportion of variance explained by the model, indicating the model’s fit.

**8. Detailed Explanation of Optimization and Assessment**

1. **Hyperparameter Optimization**: We optimize the model using grid search, iterating through a predefined set of hyperparameters to identify the configuration that minimizes validation MSE. The best hyperparameters are saved to best\_hyperparameters.json.
2. **Assessment**:
   * The model’s predictions on the test set are evaluated using MSE, RMSE, MAE, and R².
   * Plots such as learning curves and residuals are generated to visually assess model performance.
   * The metrics are stored in model\_metrics.json, providing a reference for comparing models and hyperparameter configurations.
3. **Plotting and Visualization**: The plot\_utils.py module generates diagnostic plots, such as:
   * **Learning Curves**: To show how error decreases with more training data.
   * **Residuals Plot**: To check for patterns in the residuals, which could indicate issues like heteroscedasticity or model bias.
4. **Model and Hyperparameter Logging**:
   * The best hyperparameters from the grid search are logged in best\_hyperparameters.json, ensuring reproducibility and documentation of the best configuration.
   * Additionally, model\_metrics.json captures various evaluation metrics for the final model. These metrics include the values of MSE, RMSE, MAE, and R² on both the validation and test sets, providing a quantitative assessment of model accuracy.
5. **Intermediate Data Storage**: Intermediate data such as validation predictions, test predictions, and any residuals calculated during training are saved. These can be used for further analysis, model diagnostics, or as data for generating publication-ready figures.
6. **Error Handling and Debugging**:
   * SLURM error and output logs provide critical information for debugging model training on the HPC. Each job generates an error log (physics\_ml\_<jobID>.err) and an output log (physics\_ml\_<jobID>.out), which helps trace issues such as dependency mismatches, computational errors, or resource allocation failures.
   * Through these logs, issues with GPU availability, Python packages, and SLURM configuration are identified and resolved, ensuring smoother model training and evaluation.

**Conclusion**

This documentation provides a comprehensive view of the project, including file organization, model selection, data processing, hyperparameter optimization, and model evaluation. Each component is designed to streamline the training, tuning, and assessment of multiple machine learning models, specifically for regression tasks. Through meticulous logging, visualization, and metric tracking, the project aims to create a high-performance predictive model with robust evaluation practices for publication.