Predictive mass Project

Master 2 Informatique (SID)
2023-2024

Auteurs:
Idriss FELLOUSSI
Aghilas SMAIL
Lamara MOUZNI

Encadrant:
Feda Almuhisen

January 16, 2024

Contents

1 Introduction

The core of this project is the development of an MLOps pipeline, leveraging the data from 104 simulations encompassing 15 different sensor readings. Our goal is to predict the mass of element X within various containers, a task that necessitates a nuanced approach to machine learning and data processing. The report will chronicle the journey from raw data acquisition to the deployment of a predictive model, focusing on data preprocessing, feature engineering, and the intricacies of model selection and optimization.

We emphasize not just model accuracy but also scalability and robustness, ensuring the pipeline's compatibility with diverse simulation scenarios. By integrating advanced MLOps methodologies, we aim to automate and streamline the machine learning workflow, thereby enhancing efficiency and predictive reliability in real-world applications.

The subsequent sections of this report will detail the methodology, challenges encountered, solutions implemented, and the results achieved, providing a comprehensive guide to the implementation of MLOps in simulation-based predictive analytics.

2 Project Overview

The project structure includes many directories :

- **data** : Contains the raw data file "data-Mass.xlsx", and the .dvc file of our data.
- **models** : Contains serialized versions of trained models, such as 'gbr.joblib' and 'lr.joblib'
- **notebooks** : Jupyter notebooks like test-scripts.ipynb for interactive development, testing scripts, and Predictive Mass of an element.ipynb for perhaps exploratory data analysis or initial modeling.
- **scripts** : Python scripts for different tasks in the machine learning pipeline:
    1. train.py: For training machine learning models.
    2. evaluate.py: To assess the performance of the models.
    3. mass-prediction.py: Likely contains the code to make predictions using the trained models.
    4. preprocess-data.py: For data cleaning and preparation.
    5. visualise.py: For data visualization purposes.
- **mlflow** content :
    1. 'mlruns' how is used by MLflow to store metadata and artifacts for each run.
    2. 'mlflow.db' how is a database for mlflow.
    3. 'models.ipynb' to train , test , evaluat and deploy the models on mlflow.

3 Data Exploration

3.1 Loading and Exploring the Dataset

Exploring this dataset provides promising insights into the relationship between various features. The linear regression models, both basic and Lasso with normalization, showcase strong predictive capabilities, as indicated by low mean absolute errors and high R-squared values.

The correlation matrix provides a clear picture of the interdependencies among features, with notable positive correlations observed, particularly between 'mass_X' and several sensors ('sen1', 'sen2', 'sen4', 'sen5', etc.). This suggests that changes in the mass are associated with consistent variations in sensor readings. Furthermore, the summary statistics offer a detailed snapshot of the dataset's central tendencies and dispersion, while the identification of missing values ensures transparency in assessing data completeness.

3.2 Feature Analysis

A thorough analysis was conducted to examine the impact of various features on the mass of element X. This study includes container numbers, serving as categorical identifiers, and sensor signals, which are numerical predictors. We also performed a correlation between the 15 sensors to assess their distribution within the dataset.

The correlation matrix provides insight into the linear relationships among the features in your dataset. Values close to 1 indicate a strong positive correlation, suggesting a tendency to increase or decrease simultaneously. For instance, variables "sen4" and "sen5" show a correlation very close to 1, indicating a strong positive linear relationship. Conversely, values close to -1 signal a strong negative correlation, suggesting a tendency to vary in opposite directions.

The correlation between "sen3" and "sen10" is close to -0.75, highlighting a strong negative relationship. Values close to 0 indicate a weak linear correlation. For example, the correlation between "sen3" and "sen4" is close to 0, indicating a weak linear relationship between these two variables. In summary, this matrix helps identify linear associations between features, which can be crucial for understanding potential dependencies in the data.

4 Model Development

4.1 Choice of Machine Learning Models

For the task of predicting the mass of element X based on sensor signals, we have selected two contrasting yet complementary machine **learning models**: Linear Regression and **Gradient Boosting Regressor**. These models were chosen for their distinct learning mechanisms and their

4.2 Implementation and Training

Implement and train the chosen models using the dataset. he provided script encapsulates functions for training different regression models, namely Linear Regression, Gradient Boosting Regressor, and XGBoost. In the 'train_lr' function, a Linear Regression model is initialized and trained using the input features (X_train) and target variable (y_train). The trained model can be saved to a file if specified. The 'train_gradient_boosting_regressor' function employs a GradientBoostingRegressor, utilizing GridSearchCV for hyperparameter tuning.

The best model is identified based on the specified parameters, and the model can be saved to a file. Lastly, the 'train_xgboost' function trains an XGBoost regressor with adjusted parameters, providing flexibility in the number of estimators and maximum depth. Similar to the other functions, the trained XGBoost model can be saved to a file. These functions offer a modular and efficient way to train regression models with the flexibility to save the resulting models under the 'models' folder for future use, enhancing reproducibility and scalability in machine learning workflows

4.3 Model Evaluation

Through the evalutaion script designed to assess the performance of a given model by making predictions on a test set and calculating a specified error metric. The function takes the trained model (model), the test features (X_test), the corresponding target variable (y_test), and an error metric function (f) as inputs. The error metric is calculated by comparing the actual target values with the predictions made by the model.

An example of its application is demonstrated using the 'evaluate' function to compute both the Mean Absolute Error and R-squared Score for a Gradient Boosting Regressor (model_gbr) on a validation set (X_val, y_val). The results are then printed to the console. This modular approach allows flexibility in choosing different error metrics for model evaluation based on specific requirements or preferences.

5 MLOps Approach

5.1 Version Control with Git

Git, a distributed version control system, is employed to manage the evolution of project code. It facilitates collaborative development by allowing multiple team members to work on different features simultaneously without conflict. Branching and merging strategies are employed to manage feature development, bug fixes, and releases. Moreover, Git's robust logging of changes enables us to track the history of modifications, aiding in debugging and understanding the project's evolution.

5.2 Data Versioning with DVC

Data Version Control (DVC) is an open-source tool designed to handle large files, data sets, machine learning models, and metrics as well as code. In our project, DVC is utilized to maintain versioning of the datasets and machine learning models, which

are typically too large for Git to handle efficiently. DVC allows us to track changes to the data, reproduce experiments accurately, and share our data and model states across different stages of the pipeline. It integrates seamlessly with existing Git infrastructure, so we can use standard Git commands to manage data alongside code.

5.3 Experiment Tracking with MLflow

The integration of MLflow, a machine learning lifecycle management tool, to log and track the performance of a Gradient Boosting Regressor model. Within an MLflow run, the model's hyperparameters are logged, providing a record of the configuration used during training. Subsequently, the model's performance is evaluated on both the test and validation sets using the R-squared (r2_score) and Mean Squared Error (mean_squared_error) metrics

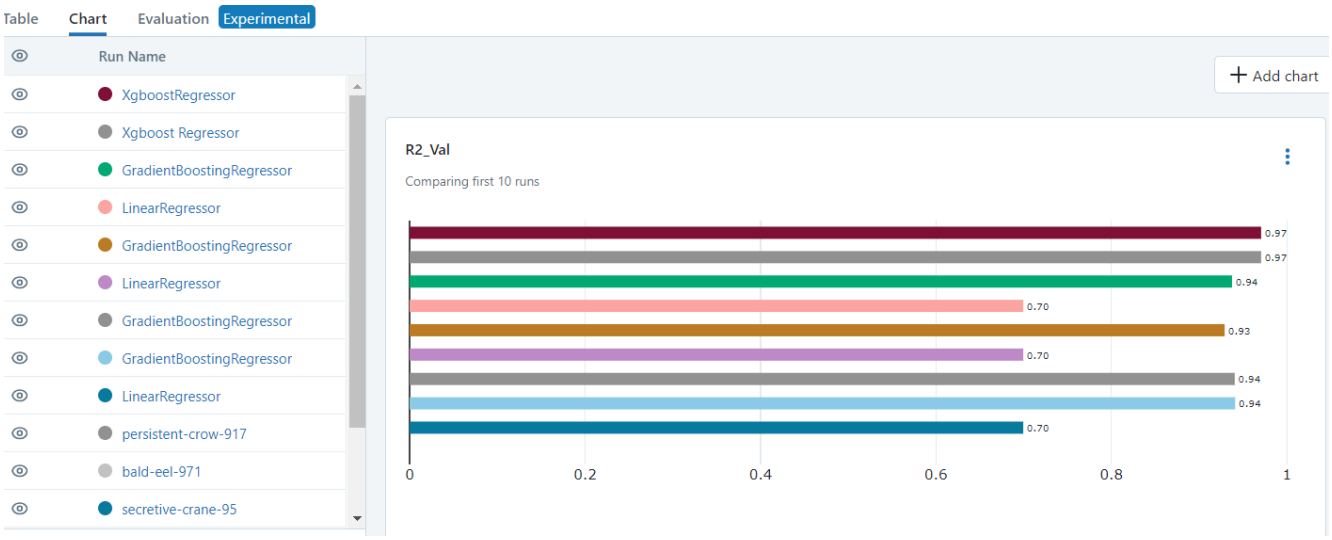

Figure 1: Comparison of 4 models r2 metric on Mlflow

Linear Regression: rose
GradientBoostingRegressor: jaune
XgboostRegressor: maron

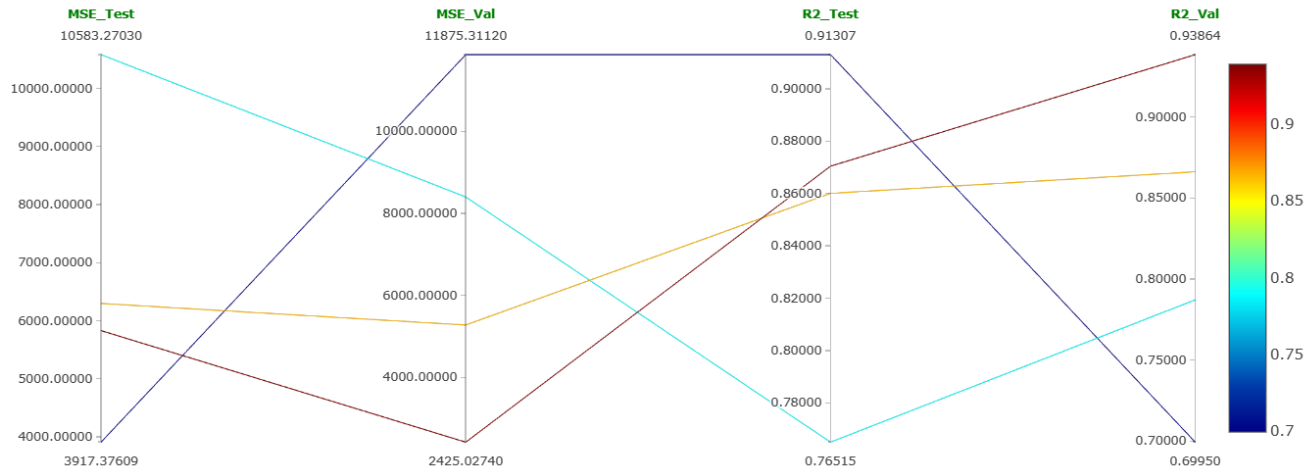

Figure 2: Comparison of 4 models on Mlflow

LinearRegressor performs well in terms of both MSE and R2 on the Test set, indicating good predictive power and fit to the data. GradientBoostingRegressor has the lowest MSE on the Validation set and the highest R2 on both sets, suggesting robust performance and generalization. XgboostRegressor performs consistently well across both sets, demonstrating good predictive accuracy. RandomForestRegressor shows decent performance but appears to be slightly outperformed by the other models in this specific scenario.

The obtained metrics are then logged using MLflow, associating them with the specific run. This practice ensures transparent and reproducible experimentation by capturing not only the model parameters but also the performance metrics in a centralized tracking system. Additionally, the trained model itself is logged as a 'sklearn' model within MLflow, allowing for easy retrieval and deployment. The use of MLflow in this context enhances model management, monitoring, and collaboration throughout the machine learning workflow

6 Model Deployment

The provided script establishes a Flask web application for deploying machine learning models predicting mass based on sensor readings. The application loads pre-trained linear regression (lr_model) , gradient boosting regression (gbr_model) and xgboost regrerssor (xgbr_model) models and sets up routes to handle user requests. The root route renders an 'index.html' template containing a form for inputting sensor readings.

The '/predict' route processes POST requests, extracting the selected model and sensor readings, performing predictions, and rendering a 'result.html' template displaying the predicted mass. This user-friendly interface allows individuals to interactively input sensor data and receive immediate mass predictions, showcasing the seamless integration of machine learning models into a web service for accessible deployment.

7 Conclusion

In conclusion, this project establishes a robust MLOps pipeline for predicting element X mass in diverse containers using 15 sensor readings. The journey from data acquisition to model deployment is meticulously detailed, emphasizing model accuracy, scalability, and robustness. The project structure, exploration insights, model choices, and MLOps tools showcase a systematic and flexible approach.

The integration of Git, DVC, and MLflow enhances version control, data management, and experiment tracking. The deployment of models in a user-friendly Flask web application demonstrates the practical application of predictive analytics, ensuring accessibility and real-world usability. Overall, this report offers a comprehensive guide to effective MLOps implementation in simulation-based predictive analytics.