

Prediction of LC50 value using

Quantitative structure activity relationship models (QSAR models)

Low Level Design

Domain: Machine Learning

Author : Debasish Mohanty

Date: 1.06.2023

**Document Version Control**

|  |  |  |  |
| --- | --- | --- | --- |
| Date issued | Version | Description | Author |
| June 1st , 2023 | 1.1 | First Draft | Debasish Mohanty |

CONTENTS

1. [Introduction](#_Toc110595994) 03

1.1[What is Low-Level Design Document?](#_Toc110595995) 03

1.2[Scope](#_Toc110595996) 03

2.Architecture 04

2.1Model Development Architecture 04

2.2Model Deployment Architecture 05

2.3Architecture Description 05

2.4Model Development 07

3.Deployment 08

3.1Frontend with ReactJS 08

3.2Backend with FastAPI 08

3.3Deployment on Cloud 09

4. Unit Cases 10

# Introduction:

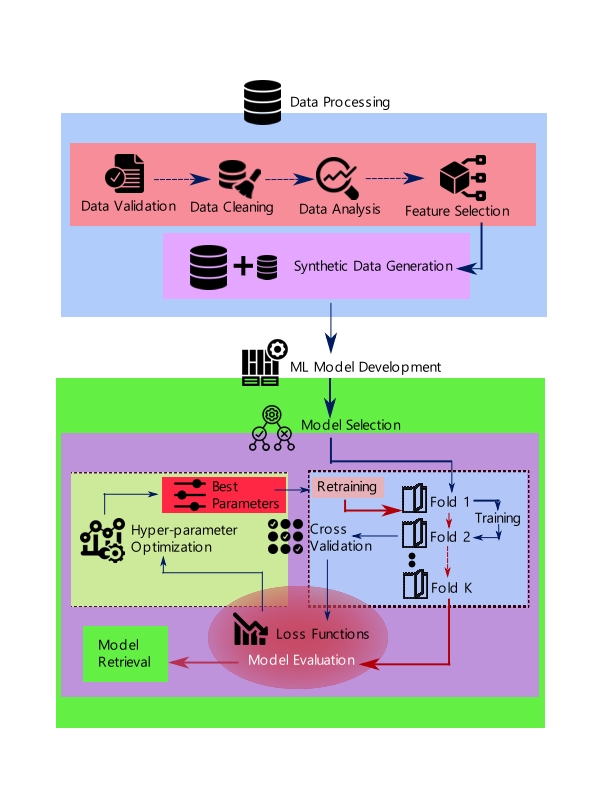
## 1.1 What is Low-Level Design Document?

The goal of LLD or a low-level design document is to give the internal logical of the actual program code for LC50 toxicity value prediction. It will explain the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli.

## 1.2 Scope:

Low-level design (LLD) is a component-level design process that follows a step-by-step refinement process. This process can be used for designing data structures, required software architecture, source code and ultimately, performance algorithms. Overall, the data organization may be defined during requirement analysis and then refined during data design work.

1. **Architecture:**
   1. **Model Development Architecture**

****

* 1. **Model Deployment Architecture**

**FRONT-END**

**(React JS)**

**CORS and Ports**

**Configuration**

**TRAINED MODEL**

**BACKEND-END**

**(FastAPI)**

**Cloud Deployment**

**(AWS EC2)**

* 1. **Architecture Description**
     + **Data Preparation**

The acquisition of data is conducted from a designated source facilitated by ineuron. The imported dataset comprises a solitary column where values are delimited by semicolons. Subsequently, the data is meticulously formatted through Python scripting, resulting in its transformation into seven distinct columns featuring appropriate numeric representations.

* + - **Data Description**

The dataset primarily encompasses seven distinct molecular descriptors, which serve as numerical indicators of compositional attributes within the given environment. The corresponding column labels are denoted as CIC0, SM1\_Dz(Z), GATS1i, NdsCH, NdssC, MLOGP, and LC50 [-LOG(mol/L)]. Notably, for the LC50 column, a logarithmic transformation is employed due to the relatively low value per unit of volume, thereby standardizing its representation. Each column, with the exception of MLOGP, manifests as a positive floating-point value. The minimum recorded MLOGP value within the dataset is -2.884.

* + - **Synthetic Data Generation**

Given the voluminous nature of the available data, it was deemed prudent to augment the dataset with a modest infusion of synthetic data in order to facilitate the training of diverse models. To accomplish this, we employed the SMOGN algorithm, which generated a new ensemble of data points imbued with subtle Gaussian perturbations. In light of the potential presence of outliers, the augmentation process was carefully constrained to ensure that the increase in data size remained below a conservative threshold of 120%. Consequently, the resultant dataset boasted a substantial number of rows, surpassing the 1000 mark.

* + - **Data Preprocessing**

During the data preprocessing stage, the range of values within the dataset was transformed into a format suitable for model training. Given the exclusively numeric nature of the data, a standard scaling technique was employed as the primary preprocessing method. Notably, the target column was exempt from any preprocessing steps to enable instantaneous assessment of loss functions. A serialized scaler object was preserved to facilitate scaling during the prediction phase, ensuring consistent transformations based on the training data.

* + - **Exploratory Data Analysis**

In this phase, a comprehensive examination of features is conducted. Boxplots are utilized to detect and address outliers, ensuring the integrity of the data. Additionally, the distribution of numerical values is meticulously plotted to assess the degree of skewness present within the dataset. This analytical process provides valuable insights into the nature and characteristics of the data under consideration. Finally the corelation matrix is plotted to investigate the relationship with other variables.

* + - **Feature Engineering**

Given that the dataset comprises only six features, no variables were omitted during the preparation of the machine learning model, regardless of their relatively lower significance with respect to the target feature.

* 1. **Model Development**
     + **Model Implementation**

Following the division of the dataset into training and testing subsets, a pipeline encompassing a Standard Scaler was employed to train multiple models, including XGB Regressor, Logistic Regressor, K Neighbour Regressor, and Random Forest Regressor. Various loss metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and coefficient of determination (r-2 score) were computed to evaluate the performance of these models.

* + - **Cross-Validation Setup**

To mitigate the risk of overfitting during the model training phase, we implemented a robust cross-validation setup. Specifically, we adopted a k-fold cross-validation approach, where the dataset was divided into 15 folds. This ensured that each fold was used as a testing set once, while the remaining 14 folds were utilized for training. By conducting the prediction process using 15 distinct models, we aimed to achieve result normalization and enhance the reliability of our predictions.

* + - **Hyper-Parameter Optimization**

In order to optimize the hyperparameters of our models, we employed the sophisticated Optuna framework. Our approach involved defining an objective function that was executed for a sample run with a random fold, allowing us to identify the best hyperparameters. These optimal parameters were subsequently applied across all folds to ensure superior results. The number of parameter samplings was adjusted based on the complexity of each model, enabling us to strike a balance between exploration and exploitation in the hyperparameter space.

* + - **Model Evaluation**

The test dataset, constituting 1/15th of the overall dataset, was employed to assess the performance of the model. The hyperparameter optimization process focused on minimizing the root mean square error (RMSE) as a measure of accuracy. In addition to RMSE, other evaluation metrics such as mean absolute error (MAE) and R-squared (R-2) score were utilized to comprehensively evaluate the model's predictive capabilities. The metrics MSE and RMSE were employed to gauge the presence and extent of outliers, while the R-2 score provided insights into the goodness of fit between the predictor features and the target feature.

1. **Deployment**
   1. **Frontend with ReactJS**

In this project, the user interface (UI) was developed using ReactJS, a JavaScript library. The front-end implementation involved the creation of a robust framework to facilitate the communication with the backend through API endpoints. To retrieve data from the server-side, a predominant utilization of the axios library was observed, enabling seamless and efficient API calls.

* 1. **Backend with FastAPI**

The backend API for this project was implemented using the FastAPI framework, renowned for its efficiency and performance. It served as the foundation for seamlessly integrating the machine learning model into the backend infrastructure. The FastAPI backend was responsible for receiving and processing user inputs, seamlessly feeding them into the underlying model for prediction, and subsequently delivering the corresponding results back to the frontend interface.

* 1. **Deployment on Cloud**

The fully operational code was obtained from the code repository using Git and deployed on an AWS EC2 instance. The necessary configuration was performed to enable Cross-Origin Resource Sharing (CORS) on the instance, allowing the service to be accessed via HTTP and HTTPS protocols. The ReactJS and FastAPI processes were seamlessly interconnected by appropriately configuring their respective ports, enabling smooth interaction between the frontend and backend via the API endpoints. Finally, the functionality of the application was tested by accessing it through the assigned IP address and the domain name provided by AWS, ensuring a seamless user experience.

# Unit cases

|  |  |  |
| --- | --- | --- |
| Test Case Description | Pre-Requisite | Expected Result |
| Verify whether the Application URL is  accessible to the user | Application URL should be defined | Application URL should be accessible to the user |
| Verify whether the Application loads completely for the user when the URL is accessed | 1. Application URL is accessible   2. Application is deployed | The Application should load completely for the user when the URL is accessed |
| Verify whether user is able to see input fields. | Application is accessible | User should be able to see input fields |
| Verify whether user is able to edit all input fields | Application is accessible | User should be able to edit all input fields |
| Verify whether user gets Submit button to submit the inputs | Application is accessible | User should get Submit button to submit the inputs |
| Verify whether user is presented with results on clicking submit | Application is accessible | User should be presented with results on clicking submit |
| Verify whether the results are in accordance to the selections user made | Application is accessible | The results should be in accordance to the selections user made |