LOW LEVEL DESIGN

# Credit card default prediction

**Introduction**

**What is a Low-Level Design Document?**  
The Low-Level Design (LLD) Document provides the internal logical design of the program code for the Credit Card Default Prediction System. It describes the system's architecture, modules, and components in detail, enabling developers to write the actual code based on this document. LLD focuses on class diagrams, method descriptions, relationships between classes, data structures, and algorithms, offering a clear roadmap for implementation.

**Scope**

Low-Level Design (LLD) is a component-level design process that refines the high-level architecture into detailed specifications. This includes defining data structures, algorithms, software architecture, and source code specifics. The goal is to break down the system into manageable modules that can be developed, tested, and integrated to form the complete application.

**Architecture**

**Architecture Description**

The architecture for the Credit Card Default Prediction System is divided into several layers:

1. **Data Ingestion Layer**: This layer handles the extraction and loading of data from the Kaggle dataset. It ensures that the data is correctly formatted and ready for preprocessing.
2. **Data Preprocessing Layer**: This layer includes all steps required to clean and prepare the data for model training. This involves handling missing values, encoding categorical variables, and scaling features.
3. **Model Training Layer**: In this layer, machine learning models are trained using the preprocessed data. Various algorithms are tested, and the best-performing models are selected.
4. **Prediction Layer**: Once trained, models are used to predict whether a customer will default on their credit card payment. This layer interfaces with the front-end to provide real-time predictions.
5. **Deployment Layer**: This layer involves deploying the trained models to a production environment (AWS), making them accessible via a REST API.

**Data Description**

**Dataset**

* **Source**: Kaggle (Credit Card Default Dataset)
* **Format**: CSV
* **Features**:
  + LIMIT\_BAL: Credit limit.
  + SEX: Gender.
  + EDUCATION: Education level.
  + MARRIAGE: Marital status.
  + AGE: Age.
  + PAY\_0 to PAY\_6: Payment history (last 6 months).
  + BILL\_AMT1 to BILL\_AMT6: Bill statement amounts (last 6 months).
  + PAY\_AMT1 to PAY\_AMT6: Previous payments (last 6 months).
* **Target**: default.payment.next.month: Binary outcome indicating whether a customer will default.

**Data Ingestion**

* **Database Creation**: Create a database to store raw data, preprocessed data, and predictions.
* **Table Creation**: Tables are created in the database to store different stages of the data pipeline (raw, processed, and predictions).
* **Data Insertion**: Insert CSV data into the database tables for further processing.

**Data Preprocessing**

* **Null Value Handling**: Replace missing values with appropriate imputed values or drop rows/columns as necessary.
* **Encoding**: Convert categorical variables (SEX, EDUCATION, MARRIAGE) into numerical format using one-hot encoding or label encoding.
* **Scaling**: Normalize numerical features (LIMIT\_BAL, AGE, etc.) to ensure all features contribute equally to the model.
* **Feature Engineering**: Create new features if needed to enhance model performance.

**Module Descriptions**

**Data Ingestion**

Data Ingestion Stage: The data ingestion stage involves acquiring raw data from a source and storing it locally for further processing. The approach is broken down into several key components:

1. **Configuration Management:**
   * Entity Definition: The DataIngestionConfig class, defined using the dataclass decorator, encapsulates the configuration parameters required for data ingestion. These parameters include the root directory, the source URL for data download, the local file path, and the directory for unzipping data.
   * Configuration Manager: The ConfigurationManager class reads the configuration details from a YAML file and initializes the DataIngestionConfig instance. It ensures the required directories are created and returns the configuration object.
2. **Pipeline Execution:**
   * The data ingestion process is encapsulated in the DataIngestionTrainingPipeline class. The main() method in this class orchestrates the data ingestion workflow, handling tasks like downloading the data from the source URL, storing it locally, and unzipping it if necessary.
   * This stage is triggered in the main.py file, where the pipeline is executed within a try-except block for error handling, ensuring the robustness of the workflow.
3. **DVC Integration:**
   * The DVC pipeline is configured to execute the data ingestion stage as a command, ensuring that the output is tracked and managed effectively. The resulting data is stored in a specified location (artifacts/data\_ingestion/UCI\_Credit\_Card.csv).

**Data Validation Stage:**

* Purpose: Ensures that the ingested data meets the required quality and schema standards.
* Components: Uses DataValidationConfig to encapsulate validation parameters. The validation checks against a schema defined in a YAML file and saves the status of the validation process.
* Pipeline Execution: The DataValidationTrainingPipeline class handles the validation process, and its execution is managed similarly to the data ingestion stage.

**Data Transformation Stage:**

* Purpose: Transforms the validated data into a format suitable for model training.
* Components: DataTransformationConfig holds paths and directories for transformed data storage.
* Pipeline Execution: Managed by the DataTransformationPipeline class, which processes the data and saves the transformed output.

**Model Training Stage:**

* Purpose: Trains the machine learning model using the transformed data.
* Components: ModelTrainerConfig stores configuration parameters like training data paths, model hyperparameters, and the target column.
* Pipeline Execution: The ModelTrainingPipeline class executes the training process, saving the trained model and the scaler used.

**Model Evaluation Stage:**

* Purpose: Evaluates the trained model's performance on the test dataset.
* Components: ModelEvaluationConfig contains paths for the test data, model, scaler, and evaluation metrics.
* Pipeline Execution: The ModelEvaluationPipeline class handles model evaluation, outputting performance metrics.

**Utility Functions**

* Common Utility Functions: Functions such as read\_yaml, create\_directories, and save\_bin/load\_bin are implemented in the utils/common.py file to streamline tasks like reading configuration files, managing directories, and handling data serialization.

**Overall Workflow**

* The entire machine learning workflow is managed through a sequence of pipeline stages, each responsible for a specific aspect of the project. The stages are executed in sequence as defined in the main.py file, ensuring a systematic approach to model development and deployment. The use of DVC for managing these stages ensures reproducibility and version control for the data and models.

**Unit Test Cases**

| **Test Case Description** | **Pre-Requisite** | **Expected Result** |
| --- | --- | --- |
| Verify data ingestion from CSV | CSV file path is provided | Data is correctly loaded into the database |
| Verify handling of missing values | Data contains missing values | Missing values are correctly imputed or handled |
| Verify encoding of categorical variables | Data contains categorical variables | Variables are correctly encoded |
| Verify scaling of numerical features | Data contains numerical features | Features are correctly scaled |
| Verify model training using Logistic Regression | Preprocessed data is available | Model is trained and saved successfully |
| Verify prediction accuracy using test data | Trained model and test data are available | Model returns predictions with acceptable accuracy |
| Verify model deployment to AWS | Trained model is available | Model is deployed and accessible via API |
| Verify real-time prediction API | User input is provided | API returns correct prediction |
| Verify data insertion into prediction table | Prediction is made | Prediction results are stored in the database |

This LLD provides a detailed breakdown of each component in the Credit Card Default Prediction System. With this document, developers should be able to code the system with clarity and precision, ensuring that all aspects of the project are covered.