Low Level Design (LLD)

Credit Card Default Detection Application

by

Abhijit Paul

Gouthami K

Revision Number: 1.0
Last date of revision: 13/12/2023

Document Version Control

Date Issued	Version	Description	Author
13/12/2023	1	Initial LLD - V1.0	Abhijit Paul

Contents

Abstract

The Low-Level Design (LLD) document for the Credit Card Default Detection machine learning project serves as an in-depth guide to the intricacies of the system's architecture and implementation.

The document begins with a clear introduction, outlining its purpose and scope, emphasizing its role in detailing the intricate design decisions and specifications for the Credit Card Default Detection system.

Within the LLD, the system architecture is depicted, illustrating the relationships and interactions between components. The chosen classification algorithm, its architecture, and the requisite input features are meticulously explained, offering insights into model training methodologies and validation techniques.

API design considerations are documented, encompassing endpoints, input validation, error handling, and security measures such as token authentication. Deployment strategies, including environment specifications and containerization details, are outlined for seamless implementation.

Testing methodologies, both unit and integration, are elucidated to ensure the reliability of the machine learning model and API functionalities. Logging mechanisms and monitoring metrics are defined for effective debugging and system health assessment.

Security measures address data privacy concerns, outlining strategies to safeguard sensitive financial data. Model explainability techniques are incorporated to enhance transparency in predictions.

This abstract encapsulates the salient features of the Low-Level Design document, serving as a succinct reference for stakeholders involved in the development and evolution of the Credit Card Default Detection system.

1 Introduction

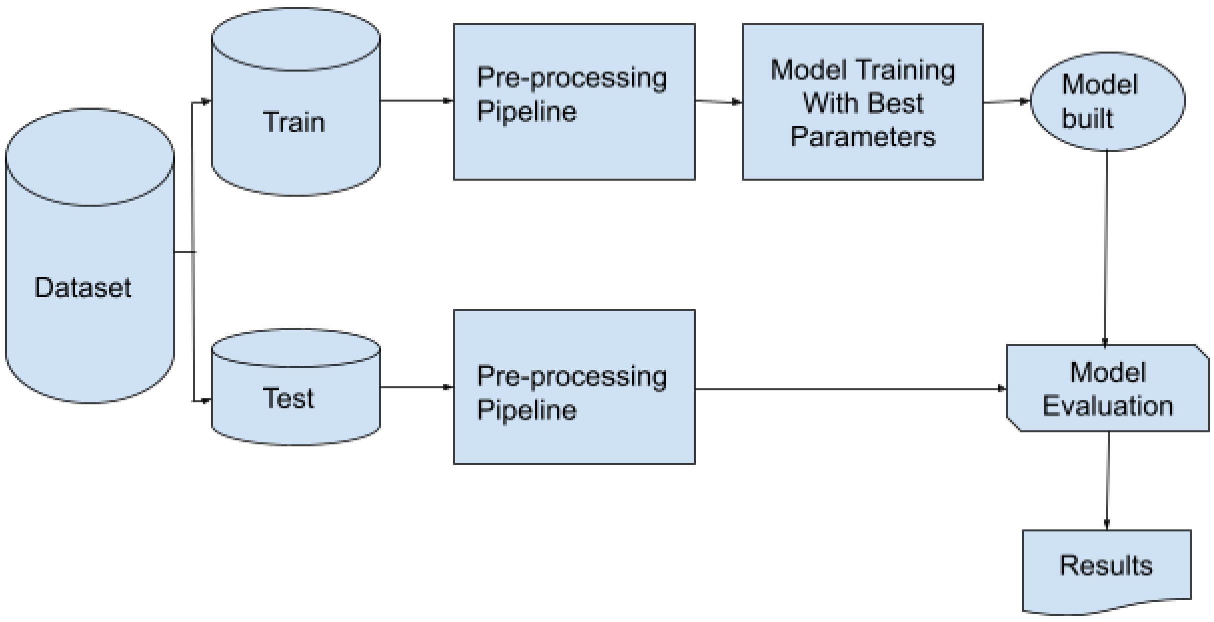
1.1 Why this Low-Level Design Document?

The purpose of this Low-Level Design (LLD) Document is to add the detailed description of the Credit Card Default Detection model. 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. This document is intended for both the stakeholders and the developers of the system and will be proposed to the higher management for its approval. This document provides a detailed description of the application's internal design, focusing on various components and their interactions.

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 and source code. Overall, the data organization may be defined during requirement analysis and then refined during data design work.

2 System Architecture



3 Machine Learning Components

3.1 Dataset

The Dataset Component handles the acquisition and management of data for the Credit Card Default Detection model.

Data Source: Specify the source of the dataset (e.g., CSV file, database).
Data Loading: Describe the process of loading data into memory.
Features and Target: Identify the features and the target variable (credit card default status).

- Data retrieval, cleaning, and initial exploration.
- Providing data for the Preprocessing Pipeline.

3.2 Preprocessing Pipeline

The Preprocessing Pipeline prepares the dataset for model training by handling missing values, encoding categorical variables, and scaling features.

Imputation: Define the strategy for handling missing values.
Encoding: Specify the encoding method for categorical variables.
Scaling: Describe the method used for feature scaling.

- Cleaning and transforming raw data into a format suitable for model training.
- Ensuring data consistency and compatibility with the chosen machine learning algorithm

3.3 Model Training & Hyper-parameter Tuning

The Model Training & Hyper-parameter Tuning Component focuses on training a machine learning model to predict credit card defaults.

Algorithm Selection: Specify the chosen classification algorithm (e.g., Random Forest, Logistic Regression).
Hyper-parameter Tuning: Define the hyper-parameters to be tuned.
Cross-validation: Describe the cross-validation strategy.

- Training the machine learning model on the preprocessed dataset.
- Fine-tuning hyper-parameters to enhance model performance.

3.4 Model Evaluation

The Model Evaluation Component assesses the performance of the trained model.

Metrics: Define evaluation metrics (e.g., accuracy, precision, recall, F1-score).
Cross-validation Evaluation: Specify the strategy for cross-validated model evaluation.

- Evaluating model performance on both training and validation sets.
- Calculating and reporting relevant metrics to assess prediction quality.

4 API Design

4.1 Front End Application

Develop a front end application using Flask for API and HTML as user-interface to accept input data to the model

4.2 API Endpoints

Expose API endpoints for the model to consume requests from users and revert the prediction as an output.

5 Model Deployment

5.1 AWS

- Prepare Your Model: Ensure your machine learning model is trained and ready for deployment. Save or export your model in a format compatible with your chosen framework
- AWS S3 Bucket: If your model artifacts or datasets are large, consider storing them in an S3 bucket. Upload your model files to an S3 bucket.
- AWS App Runner: It is a fully managed service that makes it easy for developers to deploy from source code or container image directly to a scalable and secure web application.
- Security Measures: Implement security measures, including VPC configurations, encryption, and IAM roles.
- Monitoring and Logging: Utilize AWS CloudWatch for monitoring your deployed services. Configure logging to capture relevant information for debugging and analysis.

5.2 Hugging Face

- Configure Hugging Face Spaces: Set up a Hugging Face account and create a space for your project.
- Prepare Your Model: Save or export your trained model in a format compatible with Hugging Face

- Upload Model to Spaces: Upload your model files to the Hugging Face Spaces.
- Create Inference API: Define an inference API using a Flask app or a FastAPI app
- Deploy on Hugging Face Inference API: Deploy your API using the Hugging Face Inference API service.
- Monitoring and Logging

6 Documentation

6.1 Code Documentation

Use Sphinx to create a project documentation of the machine learning codebase thoroughly.

6.2 User Guide

Create a user guide for end-users or developers interacting with the system.

7 Testing

7.1 Unit Testing

Unit testing is an essential component of the development process to verify that individual components of the system perform as expected. In the context of the Credit Card Default Detection machine learning project, unit testing is focused on ensuring the correctness of key functions and modules.

7.1.1 Unit Test Cases

Define unit test cases for critical functions and modules related to dataset handling, preprocessing, model training, and evaluation.

Example Unit Test Cases:

1. **Dataset Component:**
   - Test data loading functions to ensure they correctly fetch data from the specified source.
   - Verify that the loaded dataset has the expected structure and features.

```python
def test_data_loading():
    # Your test code here...
    assert ...

def test_dataset_structure():
    # Your test code here...
    assert ...
```

2. **Preprocessing Pipeline Component:**
   - Test preprocessing functions to confirm that missing value handling and feature scaling are performed accurately.

```python
def test_missing_value_handling():
    # Your test code here...
    assert ...

def test_feature_scaling():
    # Your test code here...
    assert ...
```

3. **Model Training Component:**
   - Test the model training function to ensure that it converges and produces a trained model.

```python
def test_model_training():
    # Your test code here...
    assert ...
```

4. **Model Evaluation Component:**
   - Test the model evaluation functions to verify that metrics are calculated correctly.

```python
def test_evaluation_metrics():
    # Your test code here...
    assert ...
```

7.1.2 Testing Framework

Specify the testing framework used (e.g., `pytest`, `unittest`) and how tests can be executed.

Example:

```python
# Instructing developers to run tests using pytest
# Command: pytest test_module.py

# Test runner configuration details...
```

7.1.3 Continuous Integration

Discuss how unit tests are integrated into the continuous integration (CI) pipeline to ensure that tests are automatically executed whenever changes are made to the codebase.

Example:

```yaml
# CI configuration file (e.g., .github/workflows/ci.yml)
on:
  push:
    branches:
      - main

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout repository
      uses: actions/checkout@v2

    - name: Set up Python
      uses: actions/setup-python@v2
      with:
        python-version: 3.8

    - name: Install dependencies
      run: |
        pip install -r requirements.txt
        pip install pytest
    - name: Run tests
      run: pytest
```

7.1.4 Code Coverage

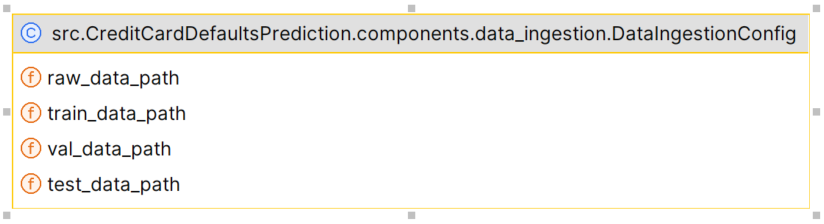
If applicable, discuss how code coverage metrics are monitored and maintained.

Example:

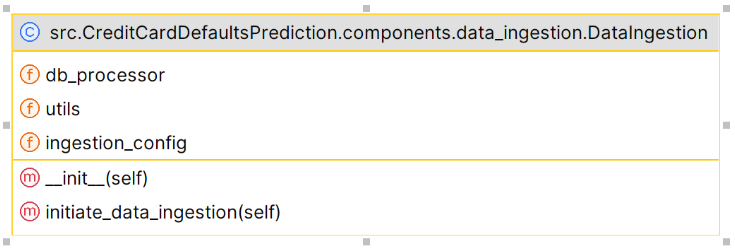
```yaml
# Code coverage configuration file (e.g., .coveragerc)
[run]
source = src/
omit = tests/*

# CI configuration for code coverage (continuation from previous example)
    - name: Install test coverage tool
      run: pip install coverage

    - name: Run tests with coverage
      run: coverage run -m pytest

    - name: Generate coverage report
      run: coverage report
```

6 UML Diagram

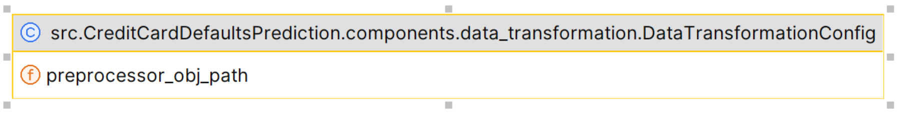
6.1.1 components.data_ingestion.DataIngestionConfig


src.CreditCardDefaultsPrediction.components.data_ingestion.DataIngestionConfig
raw_data_path
train_data_path
val_data_path
test_data_path

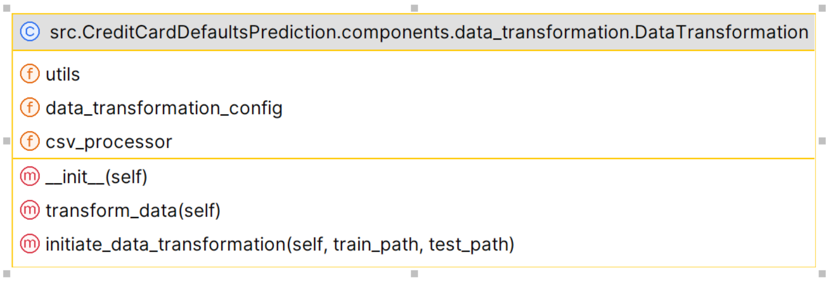
6.1.2 components.data_ingestion.DataIngestion


src.CreditCardDefaultsPrediction.components.data_ingestion.DataIngestion
db_processor
utils
ingestion_config
__init__(self)
initiate_data_ingestion(self)

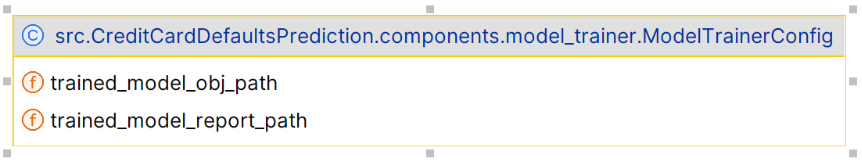
6.2.1 components.data_transformation.DataTransformationConfig


src.CreditCardDefaultsPrediction.components.data_transformation.DataTransformationConfig
preprocessor_obj_path

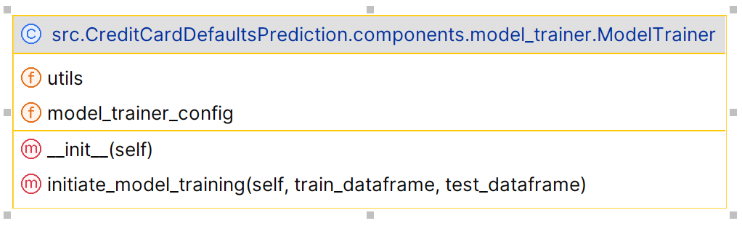
6.2.2 components.data_transformation.DataTransformation


src.CreditCardDefaultsPrediction.components.data_transformation.DataTransformation
utils
data_transformation_config
csv_processor
__init__(self)
transform_data(self)
initiate_data_transformation(self, train_path, test_path)

6.3.1 components.model_trainer.ModelTrainerConfig


src.CreditCardDefaultsPrediction.components.model_trainer.ModelTrainerConfig
trained_model_obj_path
trained_model_report_path

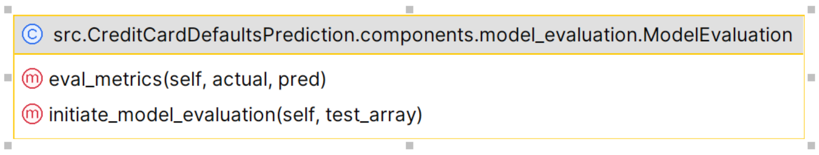
6.3.2 components.model_trainer.ModelTrainer



src.CreditCardDefaultsPrediction.components.model_trainer.ModelTrainer

- utils
- model_trainer_config
- __init__(self)
- initiate_model_training(self, train_dataframe, test_dataframe)

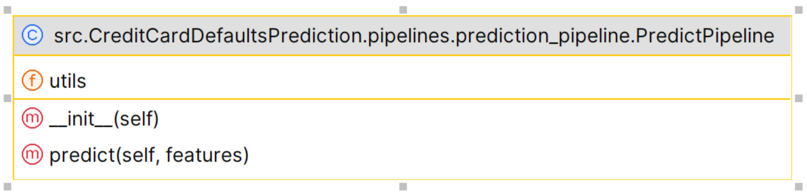
6.4.1 components.model_evaluation.ModelEvaluation



src.CreditCardDefaultsPrediction.components.model_evaluation.ModelEvaluation

- eval_metrics(self, actual, pred)
- initiate_model_evaluation(self, test_array)

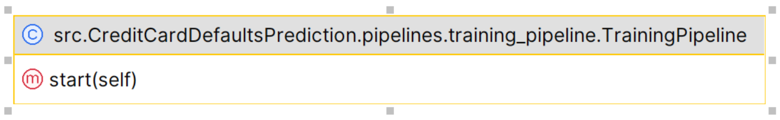
6.5.1 pipelines.prediction_pipeline.PredictPipeline



src.CreditCardDefaultsPrediction.pipelines.prediction_pipeline.PredictPipeline

- utils
- __init__(self)
- predict(self, features)

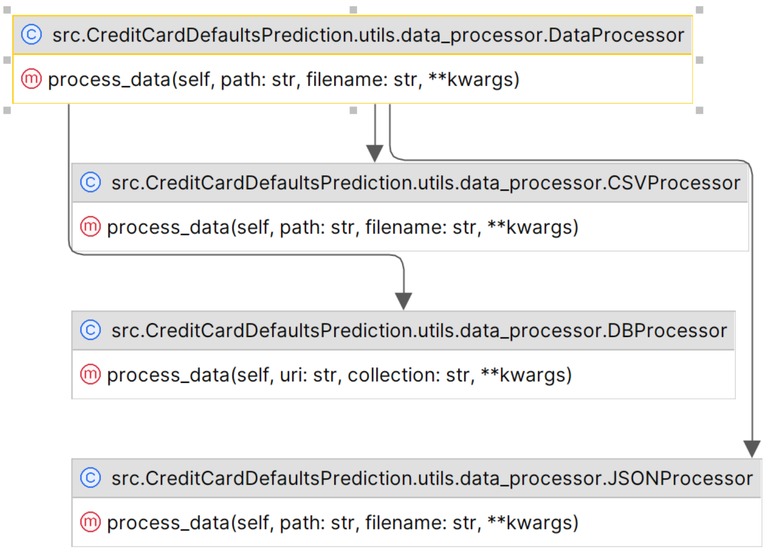
6.6.1 pipelines.training_pipeline.TrainingPipeline



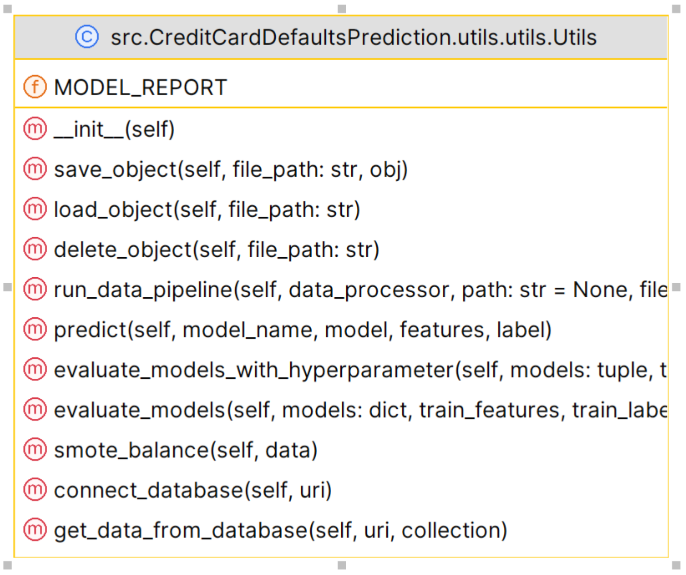
src.CreditCardDefaultsPrediction.pipelines.training_pipeline.TrainingPipeline

- start(self)

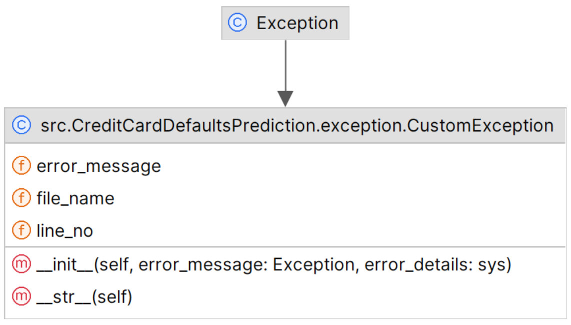
6.6.1 utils.data_processor.DataProcessor


src.CreditCardDefaultsPrediction.utils.data_processor.DataProcessor
process_data(self, path: str, filename: str, **kwargs)
src.CreditCardDefaultsPrediction.utils.data_processor.CSVProcessor
process_data(self, path: str, filename: str, **kwargs)
src.CreditCardDefaultsPrediction.utils.data_processor.DBProcessor
process_data(self, uri: str, collection: str, **kwargs)
src.CreditCardDefaultsPrediction.utils.data_processor.JSONProcessor
process_data(self, path: str, filename: str, **kwargs)

6.7.1 utils.utils.Utils


src.CreditCardDefaultsPrediction.utils.utils.Utils
MODEL_REPORT
__init__(self)
save_object(self, file_path: str, obj)
load_object(self, file_path: str)
delete_object(self, file_path: str)
run_data_pipeline(self, data_processor, path: str = None, file
predict(self, model_name, model, features, label)
evaluate_models_with_hyperparameter(self, models: tuple, t
evaluate_models(self, models: dict, train_features, train_labe
smote_balance(self, data)
connect_database(self, uri)
get_data_from_database(self, uri, collection)

6.8.1 utils.utils.Utils


Exception
src.CreditCardDefaultsPrediction.exception.CustomException
error_message
file_name
line_no
__init__(self, error_message: Exception, error_details: sys)
__str__(self)

8 Conclusion

This Low-Level Design document outlines the design details of essential components in the Credit Card Default Detection machine learning project. Each component is designed to fulfill specific responsibilities in the overall system, contributing to a robust and effective credit card default prediction solution.