

Github:

https://github.com/sudhanshu2198

**Credit Card Defaulter Prediction App**

Low Level Design

Domain: Machine Learning

Creator: Sudhanshu Rastogi

Date: 04.09.2023

|  |  |  |  |
| --- | --- | --- | --- |
| Date issued | Version | Description | Author |
| August 28, 2023 | 1.1 | First Draft | Sudhanshu Rastogi |
| September 3, 2023 | 1.2 | Added documents files | Sudhanshu Rastogi |

**Document Version Control**

Contents

[Introduction 3](#_Toc110433758)

[What is Low-Level Design Document? 3](#_Toc110433759)

[Scope 3](#_Toc110433760)

[Architecture 4](#_Toc110433761)

[Architecture Description 5](#_Toc110433762)

[Data Preparation 5](#_Toc110433763)

[Data Description 5](#_Toc110433764)

[Data Preprocessing 5](#_Toc110433765)

[Exploratory Data Analysis 5](#_Toc110433766)

[Feature Engineering 5](#_Toc110433767)

[Model Development 6](#_Toc110433768)

[Model implementation 6](#_Toc110433769)

[Hyper-parameter Tuning 6](#_Toc110433770)

[Model Evaluation 6](#_Toc110433771)

[Deployment 6](#_Toc110433772)

[Designing UI with Anvil 6](#_Toc110433773)

[Designing a server 6](#_Toc110433774)

[Code deployment on cloud 7](#_Toc110433775)

[Deployment Process 7](#_Toc110433776)

[Unit cases 8](#_Toc110433777)

# Introduction

## 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 Metro Interstate Traffic Volume 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.

The main objective of the project is to develop a credit card defaulter predicton model given previous 6 months of data of customer account history.

## 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.

# Architecture

**Data Preparation**

**Model**

**Development**

**Deployment**

**Deployment**

**Architecture Description**

Data Preparation

The features variables include education, sex , age, credit limit, bill amount statement,

payment amount statement, payment status for 6 months for 30,000 instances. The data

is taken from Kaggle Platform. It can be used for developing a predictive model that

predicts probability of credit card customer defaulting on the due payments in the given

month given previous 6 month account history.

Data Preprocessing

In data preprocessing step, we check if there missing data, duplicate values, and

mixed datatypes presence in feature variables. In our dataset, there was no duplicate

values, missing values but datatype of multiple columns was “object” type, therefore they

was converted to proper datatype.

Data Cleaning

This step involves outlier removal such extremely high credit limit, negative values for

age, payment account statement. Datatype correction performed on features

variables containing integer variable but having object type as object.

Exploratory Data Analysis

This step includes data visualization of features and target variables. Visualization

utilizes pie charts, scatterplot, histograms, countplot, boxplots..

Feature Engineering

New features were created utilizing provided information such as Boolean payment

defaulter feature if the customer has defaulted ever in past 6 months, categorizing age

of the customer in uniform interval brackets. Including others option in sex, education

and marital status variable.

**Model Development**

Algorithm Spot Checking

Algorithm spot checking was performed on cleaned dataset by training the different models and evaluating their performance using cross validation having 5 folds. The different model are Random Forest Regressor, Xtreme Gradient Boosting Regressor, Light Gradient Boosting Regressor , Logistic Regression. Their area under curve were calculated. The highest score is achieved by Light Gradient Boosting Regressor.

Hyper-parameter Tuning

The best model is Light Gradient Boosting Regressor from algorithm spot checking, random search on no of estimators, learning rate, max depth and subsample is performed using optuna library to get the best hyper parameters for the model. Those parameters is then used on the model to get better result.

Model Evaluation

Test dataset is used to evaluate the model. 20% of dataset was separated for testing. Predicted results of the model are compared with the actual data to check the amount of error.

**Deployment**

Streamlit Web App

### For this project, the trained is deployed for inference using an interactive web app. Interactive web app is created streamlit. Streamlit is a popular open-source framework Code deployment on cloud

The codes for this machine learning model should be deployed to the cloud, so that when data is entered into the application, our code runs, and a user gets the result online.

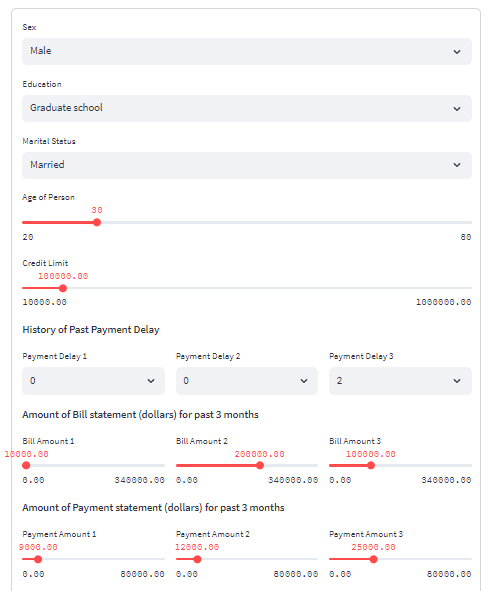
used for model deployment by machine learning and data science teams. The web application contains three pages introduction, prediction and visualization.

Designing Web App

The web application contains three pages introduction, prediction, experiments results and visualization. Prediction pages provides inference services for user defined features variables. Dynamic visualization offers valuable insights about various factors affecting the restaurant establishment at different locations.

**Deployment Process**

After creating the streamlit app and testing it on local machine, code is pushed to github and linked to streamlit cloud. Specifying the branch to be utilized for creating web app, we will in creating an app hosted on streamlit cloud.



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 |