LOW LEVEL DESIGN (LLD)

CREDIT CARD DEFAULT PREDICTION

DOCUMENT VERSION CONTROL

| Date Issue | Version | Description | Author |
| --- | --- | --- | --- |
| 07/03/2023 | 1 | Initial LLD – V 1.0 | Abdul Jaweed  Lovely Patra |
| 17/03/2023 | 2 | Final LLD - V 2.0 | Abdul Jaweed  Lovely Patra |

Content

1. Introduction
   1. What is Low Level Design Document
   2. Scope
2. Architecture
3. Architecture Description
   1. Data Description
   2. Data Transformation
   3. Feature engineering
   4. Feature Selection
   5. Testing for Classification algorithm
   6. Selecting model with best accuracy
   7. Model training
   8. Model deployment

Introduction

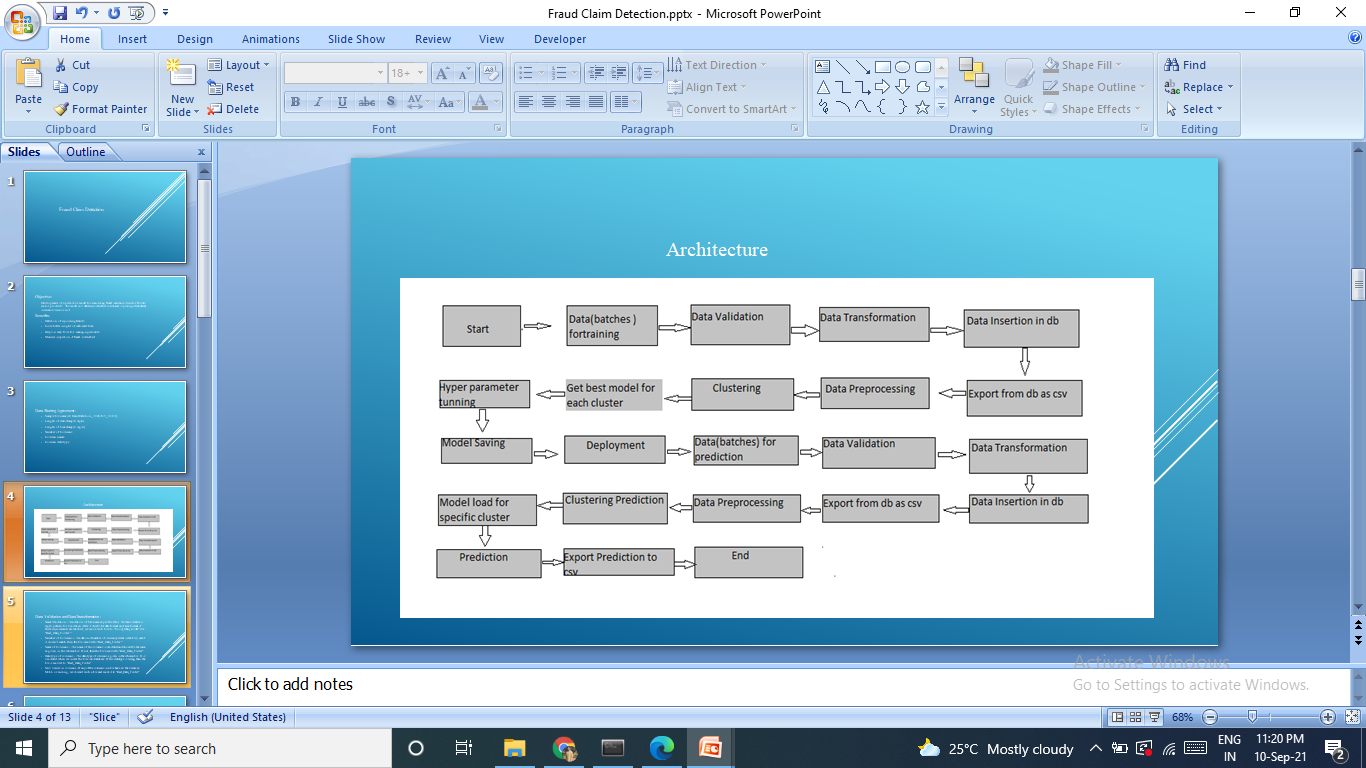
What is a Low Level Design Document?

The goal of LLD or a low-level design document (LLDD) is to give the internal logical design of the actual program code for Credit Card Default Prediction. LLD describes the class diagrams with the methods and relations between classes and program specs. It describes the modules so that the programmer can directly code the program from the document. Low-level design is a detailed description of every module of software. It describes every module in detail by incorporating the logic behind every component in the system. It delves deep into every specification of every system, providing a micro-level design.

Scope

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

Architecture



Architecture

Data Description –

This dataset contains information on default payments, demographic factors, credit data, history of payment, and bill statements of credit card clients in Taiwan from April 2005 to September 2005.

The credit card default prediction dataset which is available in UCI Machine Learning Repository consists of 30000 rows and 25 columns.

Data Validation-

Data validation is done on the data extracted from the sources, in which data types of the column, encoding of the categorical columns, and feature engineering are performed.

Data Transformation-

Here the platforms transform data into numerical to categorical, and in standardized form using a python package for further processing.

Database Operation –

For the database operations, we have chosen MongoDB. In the database operations, we are inserting the good data as a collection in MongoDB. We are choosing the M20 cluster in MongoDB Atlas with AWS as cloud provider. After the good data is inserted, we are exporting the good data as a collection in the form of csv and then uploading it to the csv file back to S3 buckets.

Data Preprocessing

In the preprocessing pipeline, we are replacing the invalid values with nan values, encoding the targets column, imputing the missing values, removing unnecessary columns, applying standard scalar and PCA transformation for dimensionality reduction Since we are following a customized machine learning approach, with KMeans model, we are creating the clusters of data, performing train test split and pass that cluster data for training.

Model Training-

Platform provides the ability for the end user to train models using our library and pipeline on the transformed and validated data based on request. The model training will add the best trained model to the system.

Here the best trained model is returned after completion of the complete machine learning pipeline.

Loading of the production model-

Now that all the trained models are kept on track using MLFlow, we need to put our best models in production or staging depending on the condition. We are putting the best model of the particular cluster in production and others in staging for every cluster.

Containerization-

The entire system is containerized using Dockers with AWS and MLFlow credentials.

Deployment –

The entire solution is now containerized and we have implemented GitHub Actions as a CI-CD pipeline, which will deploy our system to AWS ECR and ECS and an endpoint is given to us for using the application.

Further improvements and Conclusion

For the improvements we can use Kubernetes for container orchestration and management, with that we can also use tools like Prometheus, Grafana for Kubernetes management. We can adapt to micro service architecture for the entire pipelines mentioned can be individually containerized and monitored using Kubernetes We also include more DevOps tools like Jenkins and more automation. Most importantly, the retraining approach is important considering model performance from time to time.