# Low Level Design Document

## Credit Card Default Prediction

LAST DATE OF REVISION: 25/12/2023

# Contents

# ABSTRACT

Financial threats are displaying a trend about the credit risk of commercial banks as the incredible improvement in the financial industry has arisen.

In this way, one of the biggest threats faces by commercial banks is the risk prediction of credit clients. The goal is to predict the probability of credit default based on credit card owner's characteristics and payment history.

# 1 Introduction

## 1.1 Why this Low-Level Design Document?

The purpose of this document is to present a detailed description of Credit Card Default Prediction. It will explain about the purpose and feature of the application and the interface of the system.

**Problem Statement:**

**Note: All the code will be written in python version 3.7**

# 2 Scope

This Web-Application will help in detecting the defaulter and risk prediction of credit clients.

.

# 3 Technical Specifications

## 3.1 Dataset Source

| |
|---|
| Google Open Dataset |
| Kaggle |
| |
| |

# 4 Technology Stack

| | |
|---|---|
| **Frontend** | HTML/CSS |
| **Backend** | Python |
| **Database** | SQLite |
| **Framework** | Flask |
| **Deployment** | Docker, AWS, Heroku |

# 5 Data Description

The client will send data in multiple sets of files in batches at a given location. The data has been extracted from the census bureau.

The data contains 32561 instances with the following attributes:

**Features:**

1. **LIMIT_BAL:** continuous Credit Limit of the person.
2. **SEX:** Categorical: 1 = male; 2 = female
3. **EDUCATION:** Categorical: 1 = graduate school; 2 = university; 3 = high school; 4 = others
4. **MARRIAGE:** 1 = married; 2 = single; 3 = others
5. **AGE**-num: continuous.
6. **PAY_0 to PAY_6:** History of past payment. We tracked the past monthly payment records (from April to September, 2005)
7. **BILL_AMT1 to BILL_AMT6:** Amount of bill statements.

8. **PAY_AMT1 to PAY_AMT6:** Amount of previous payments.

## Target Label:

Whether a person shall default in the credit card payment or not.

9. default payment next month: Yes = 1, No = 0.

Apart from training files, we also require a "schema" file from the client, which contains all the relevant information about the training files such as:
Name of the files, Length of Date value in Filename, Length of Time value in Filename, Number of Columns, Name of the Columns, and their datatype.

# 6 Project process structure:

## Data Validation

In this step, we perform different sets of validation on the given set of training files.

1. Name Validation- We validate the name of the files based on the given name in the schema file. We have created a regex pattern as per the name given in the schema file to use for validation. After validating the pattern in the name, we check for the length of date in the file name as well as the length of time in the file name. If all the values are as per requirement, we move such files to "Good_Data_Folder" else we move such files to "Bad_Data_Folder."

2. Number of Columns - We validate the number of columns present in the files, and if it doesn't match with the value given in the schema file, then the file is moved to "Bad_Data_Folder."

3. Name of Columns - The name of the columns is validated and should be the same as given in the schema file. If not, then the file is moved to "Bad_Data_Folder".

4. The datatype of columns - The datatype of columns is given in the schema file. It is validated when we insert the files into Database. If the datatype is wrong, then the file is moved to "Bad_Data_Folder".

5. Null values in columns - If any of the columns in a file have all the values as NULL or missing, we discard such a file and move it to "Bad_Data_Folder".

## Data Insertion in Database

1) Database Creation and connection - Create a database with the given name passed. If the database has already been created, open a connection to the database.
2) Table creation in the database - Table with name - "Good_Data", is created in the database for inserting the files in the "Good_Data_Folder" based on given column names and datatype in the schema file. If the table is already present, then the new table is not created, and new files are inserted in the already present table as we want training to be done on new as well as old training files.
3) Insertion of files in the table - All the files in the "Good_Data_Folder" are inserted in the above-created table. If any file has invalid data type in any of the columns, the file is not loaded in the

table and is moved to "Bad_Data_Folder".

## Model Training

1) **Data Export from Db** - The data in a stored database is exported as a CSV file to be used for model training.

2) **Data Preprocessing**

a) Check for null values in the columns. If present, impute the null values using the categorical imputer.

b) Scale the numeric values using the standard scaler.

c) Check for correlation.

3) **Clustering -** KMeans algorithm is used to create clusters in the preprocessed data. The optimum number of clusters is selected by plotting the elbow plot, and for the dynamic selection of the number of clusters, we are using "KneeLocator" function. The idea behind clustering is to implement different algorithms
The Kmeans model is trained over preprocessed data, and the model is saved for further use in prediction.

4) **Model Selection –** After the clusters have been created, we find the best model for each cluster. We are using two algorithms, “Naïve Bayes” and "XGBoost". For each cluster, both the algorithms are passed with the best parameters derived from GridSearch. We calculate the AUC scores for both models and select the model with the best score. Similarly, the model is selected for each cluster. All the models for every cluster are saved for use in prediction.

## Prediction Data Description

The Client will send the data in multiple sets of files in batches at a given location. Data will contain the credit and payment record of customers.
Apart from prediction files, we also require a "schema" file from the client, which contains all the relevant information about the training files such as:
Name of the files, Length of Date value in FileName, Length of Time value in FileName, Number of Columns, Name of the Columns and their datatype.

## Data Validation

In this step, we perform different sets of validation on the given set of training files.

1) Name Validation- We validate the name of the files based on given Name in the schema file. We have created a regex pattern as per the name given in the schema file, to use for validation. After validating the pattern in the name, we check for the length of date in the file name as well as the length of the timestamp in the file name. If all the values are as per requirement, we move such files to "Good_Data_Folder" else we move such files to "Bad_Data_Folder".

2) Number of Columns - We validate the number of columns present in the files, and if it doesn't match with the value given in the schema file, then the file is moved to "Bad_Data_Folder".

3) Name of Columns - The name of the columns is validated and should be same as given in the schema file. If not, then the file is moved to "Bad_Data_Folder".

4) Datatype of columns - The datatype of columns is given in the schema file. This is validated when we insert the files into Database. If the datatype is incorrect, then the file is moved to "Bad_Data_Folder".

5) Null values in columns - If any of the columns in a file has all the values as NULL or missing, we discard such file and move it to "Bad_Data_Folder".

## Data Insertion in Database

1) Database Creation and connection - Create a database with the given name passed. If the database is already created, open the connection to the database.

2) Table creation in the database - Table with name - "Good_Data", is created in the database for inserting the files in the "Good_Data_Folder" based on given column names and datatype in the schema file. If the table is already present, then a new table is not created, and new files are inserted into the already present table as we want training to be done on new as well old training files.

3) Insertion of files in the table - All the files in the "Good_Data_Folder" are inserted in the above-created table. If any file has invalid data type in any of the columns, the file is not loaded in the table and is moved to "Bad_Data_Folder".

## Prediction

1) Data Export from Db - The data in the stored database is exported as a CSV file to be used for prediction.

2) Data Preprocessing :

a) Check for null values in the columns. If present, impute the null values using the categorical imputer.
b) Scale the numeric values using the standard scaler.
c) Check for correlation.

3) Clustering - KMeans model created during training is loaded, and clusters for the preprocessed prediction data is predicted.

4) Prediction - Based on the cluster number, the respective model is loaded and is used to predict the data for that cluster.

5) Once the prediction is made for all the clusters, the predictions along with the Wafer names are saved in a CSV file at a given location, and the location is returned to the client.

## Deployment

We will be deploying the model to the Heroku cloud platform.

# 8. Exception Scenarios Overall

| Step | Exception | Mitigation |
|---|---|---|
| User gives Wrong Data Source | Give proper error message | Ask the user to re-enter the details |
| User gives corrupted video files | Give proper error message | Ask the user to upload correct format such as mp4 |
| Not able to access DB | Give proper error message | Ask user to try again later |
| User uploads corrupted video | Give proper error message | Request user to try again |

# 9. Database Design

For all the uses cases , one database will be created using mongoDB . There will be separate tables for each use-case storing the data such as timestamp and number plate details. The number of columns may further be added on the basis of use-case's enhanced requirements.

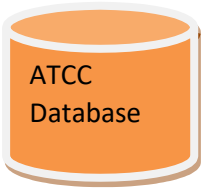
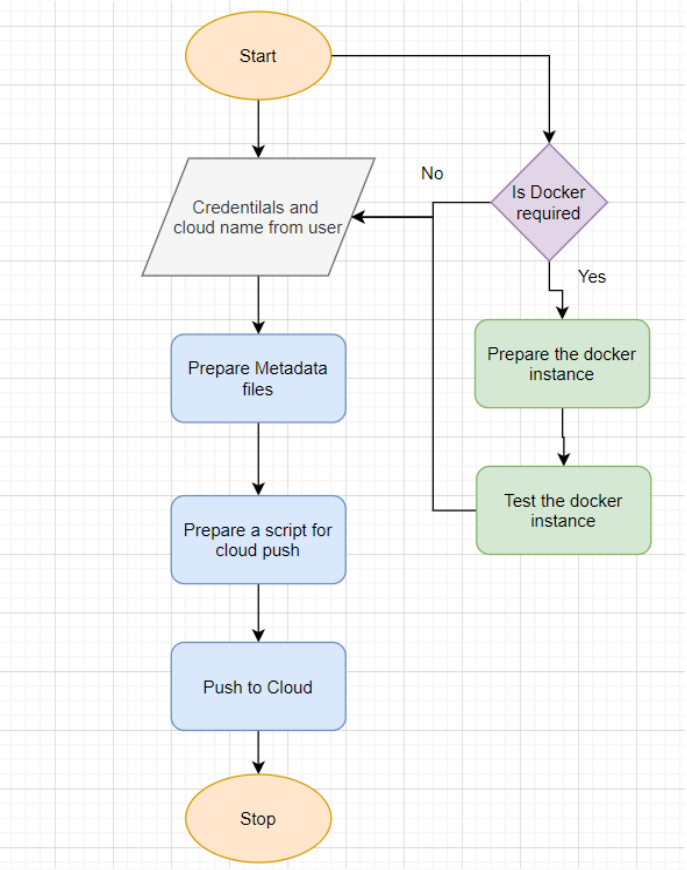



Table1: prohibited_traffic_table

| timestamp | numberplate |
|---|---|
| | |

# 10. Deployment Strategy



## 10.1. Exceptions Scenarios Module Wise

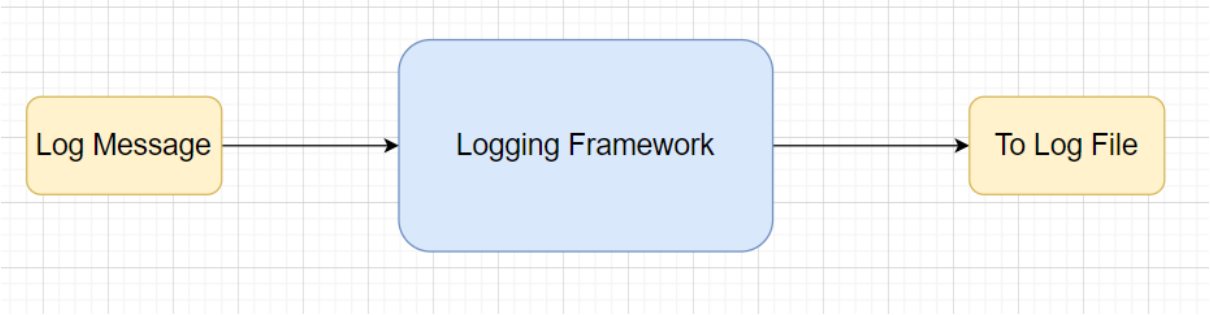
| Step | Exception | Mitigation |
|---|---|---|
| Wrong Cloud credentials | Show error message | The user enters the correct data |
| Docker instance not working | Show error message | Fix the error |
| Cloud push failed | Show the error | Make corrections to the metadata files |
| Cloud app not starting | | Ask the user for cloud logs for debugging |

# 11.Logging

To be followed by all use-cases

- Separate Folder for logs
- Logging of every step
- Entry to the methods
- Exit from the methods with success/ failure message
- Error message Logging
- Prediction start and end
- Achieve asynchronous logging

## 11.1. Technical solution design




## 11.2. Common Logging Framework Code

| | |
|---|---|
| Class Name | App Logger |
| Method Name | Log |
| Method Description | This method will be used for logging all the information to the file. |
| Input parameter names | self,file_object, log_message |
| Input Parameter Description | file_object: the file where the logs will be written<br>log_message: the message to be logged |
| Ouptput | A log file with messages |

# 12. Hardware Requirements

## 12.1. Requirements for model training

The minimum configuration should be:

12.1.2. 8 GB RAM
12.1.3. 2 GB of Hard Disk Space
12.1.4. Intel Core i5 Processor

12.1.5. Google colab/Colab pro/Jupyter Notebook

## 12.2.Requirements for model testing

The minimum configuration should be:

12.2.2. 4 GB RAM
12.2.3. 2 GB of Hard Disk Space
12.2.4. Intel Core i5 Processor

12.2.5. Google colab / Colab pro/jupyter notebook/Vscode

# 13. Coding standards

13.1. Imports should usually be on separate lines
2. Avoid trailing whitespace anywhere. Because it's usually invisible, it can be confusing.
3. Compound statements (multiple statements on the same line) are generally discouraged
4. Comments should be complete sentences. Always make a priority of keeping the comments up-to-date when the code changes. Ensure that your comments are clear and easily understandable to other speakers of the language you are writing in.
5. Never use the characters 'l' (lowercase letter el), 'O' (uppercase letter oh), or 'I' (uppercase letter eye) as single character variable names.
6. The name of the variables should start with small case capital letters and a multi word variable should be named as: word1_word2_word3.
7. The variable name should be appropriate based on the things that they do. DO NOT USE NAMES LIKE x, k, y etc. Always use a meaningful English word. For example, customer_name, nearest_neighbour etc.
8. Method names should start with small case characters. They should start with a verb and make a meaningful sense of what they are supposed to accomplish. For e.g.: load_data_from_sql()
9. Always use self for the first argument to instance methods.
10. Class names should normally use the CapWords convention. Class name should also represent the functionality of the class. For e.g. DataLoader()

11. Modules/Packages/Folders should have short, all-lowercase names. Underscores can be used in the module name if it improves readability. For e.g.: data_ingestion
12. Constants are usually defined on a module level and written in all capital letters with underscores separating words. Examples include MAX_OVERFLOW and TOTAL.
13. Comparisons to singletons like None should always be done with is or is not, never the equality operators
14. The code should be properly enclosed withing try and exception blocks and the exceptions should be handled with proper error messages.
15. Additionally, for all try/except clauses, limit the try clause to the absolute minimum amount of code necessary. Again, this avoids masking bugs
16. When a resource is local to a particular section of code, use a with statement to ensure it is cleaned up promptly and reliably after use.
17. Be consistent in return statements. Either all return statements in a function should return an expression, or none of them should. If any return statement returns an expression, any return statements where no value is returned should explicitly state this as return None, and an explicit return statement should be present at the end of the function (if reachable)
18. Object type comparisons should always use isinstance() instead of comparing types directly
19. Don't compare boolean values to True or False using ==