iNeuron

Credit Card Default Prediction

Architecture

K.Arunkumar

17/11/2023

Contents

1. Introduction

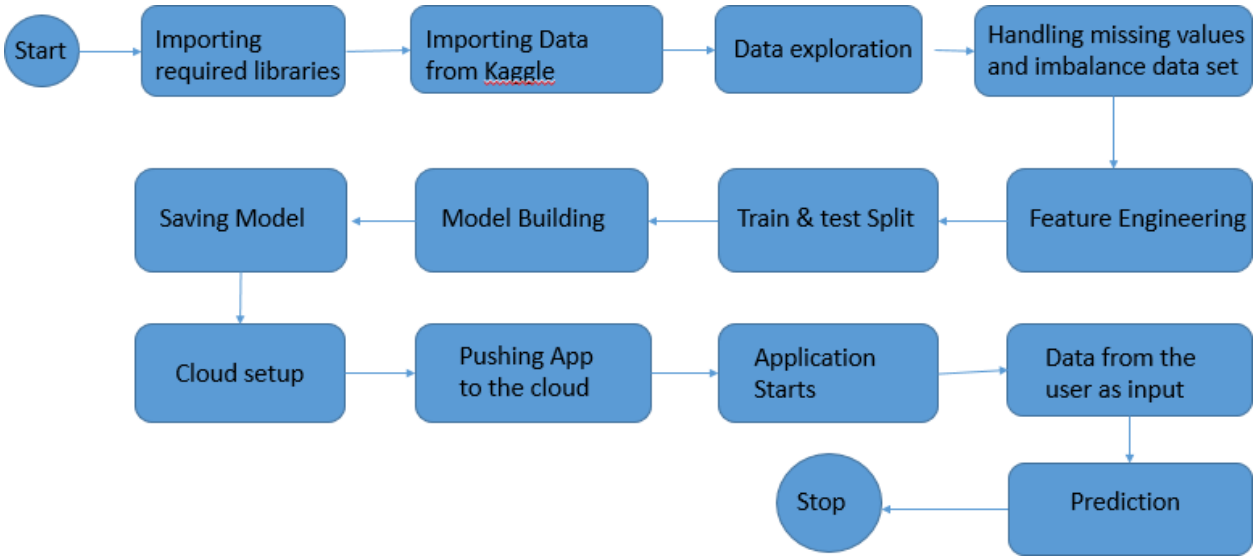
- What is a Low-Level design document?

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.

- 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.

2. Architecture



3. Architecture Description

❖ Data Description

This dataset is taken from Kaggle(URL: https://www.kaggle.com/uciml/defaultof-credit-card-clients dataset). It 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.

Content There are 25 variables:

- ID: ID of each client
- LIMIT_BAL: Amount of given credit in NT dollars (includes individual and family/supplementary credit
- SEX: Gender
  - 1=male,
  - 2=female
- EDUCATION:
  - 1=graduate school,
  - 2=university,
  - 3=high school,
  - 0, 4, 5, 6=others)
- MARRIAGE: Marital status
  - 1=married,
  - 2=single,
  - 3=divorce,
  - 0=others
- AGE: Age in years
- PAY_0: Repayment status in September 2005
  - -1: Paid in full;
  - 0: No consumption;
  - 1 = payment delay for one month;
  - 2 = payment delay for two months; . . .;
  - 8 = payment delay for eight months;
  - 9 = payment delay for nine months and above.

- PAY_2: Repayment status in August, 2005 (scale same as above)
- PAY_3: Repayment status in July, 2005 (scale same as above)
- PAY_4: Repayment status in June, 2005 (scale same as above)
- PAY_5: Repayment status in May, 2005 (scale same as above)
- PAY_6: Repayment status in April, 2005 (scale same as above)
- BILL_AMT1: Amount of bill statement in September, 2005 (NT dollar)
- BILL_AMT2: Amount of bill statement in August, 2005 (NT dollar)
- BILL_AMT3: Amount of bill statement in July, 2005 (NT dollar)
- BILL_AMT4: Amount of bill statement in June, 2005 (NT dollar)
- BILL_AMT5: Amount of bill statement in May, 2005 (NT dollar)
- BILL_AMT6: Amount of bill statement in April, 2005 (NT dollar)
- PAY_AMT1: Amount of previous payment in September, 2005 (NT dollar)
- PAY_AMT2: Amount of previous payment in August, 2005 (NT dollar)
- PAY_AMT3: Amount of previous payment in July, 2005 (NT dollar)
- PAY_AMT4: Amount of previous payment in June, 2005 (NT dollar)
- PAY_AMT5: Amount of previous payment in May, 2005 (NT dollar)
- PAY_AMT6: Amount of previous payment in April, 2005 (NT dollar)
- Default.payment.next.month: Default payment
    - 1=yes,
    - 0=n

❖ Feature Engineering
We created a new feature by taking the average of all 6 columns of the Bill Amount

❖ Train/Test Split
Split the data into 75% train set and 25% test set.

❖ Model Building
Built models and trained and tested the data on the models. Compared the performance of each model and selected the best one.

❖ Save the model
Saved the model by converting it into a pickle file.

❖ Cloud Setup & Pushing the App to the Cloud
Selected Streamlit Cloud for deployment. Loaded the application files from GitHub b to Streamlit Cloud.

❖ Application Start and Input Data by the User
Start the application and enter the inputs.

❖ Prediction
After the inputs are submitted the application runs the model and makes predictions. The out
is displayed as a message indicating whether the customer whose demographic and
behavioral data are entered as inputs, is likely to default in the following month or not.

4. Unit Test Cases

Test Case Description	Pre-Requisite	Expected Result
Verify whether the Application URL is accessible to the user	1. Application URL should be defined	The 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 the user is able to see input fields on logging in	1. Application URL is accessible 2. Application is deployed	The user should be able to see input fields on logging in
Verify whether the user is able to edit all input fields	1. Application URL is accessible 2. Application is deployed	The user should be able to edit all input fields
Verify whether a user gets Submit button to submit the inputs	1. Application URL is accessible 2. Application is deployed	The user should get Submit button to submit the inputs
Verify whether the user is presented with recommended results on clicking submit	1. Application URL is accessible 2. Application is deployed	The user should be presented with recommended results on clicking submit