System High-Level Design Document

South German Credit Risk Prediction

Version: 1.0.0

March 2023

**Table of Contents**

[1. Introduction 4](#_Toc125413681)

[1.1 Purpose 4](#_Toc125413682)

[1.2 Scope 4](#_Toc125413683)

[1.3 Definitions, Acronyms, and Abbreviations 4](#_Toc125413684)

[2. General description 5](#_Toc125413685)

[2.1 Product Perspective 5](#_Toc125413686)

[2.2 Problem Statement 5](#_Toc125413687)

[2.3 Proposed Solution 5](#_Toc125413688)

[2.4 Data description 6](#_Toc125413689)

[2.5 Data Set Information: 6](#_Toc125413690)

[3. Tools Used 6](#_Toc125413691)

[4. Design Details 7](#_Toc125413692)

[4.1 Data Engineering 7](#_Toc125413693)

[4.2 Model Engineering 8](#_Toc125413694)

[4.3 Model Deployment 8](#_Toc125413695)

[4.4 Process Flow 9](#_Toc125413696)

[4.5 Event Log 9](#_Toc125413697)

[4.6 Error Handling 9](#_Toc125413698)

[5. Performance 10](#_Toc125413699)

[5.1 Reusability 10](#_Toc125413700)

[5.2 Application Compatibility 10](#_Toc125413701)

[5.3 Resource Utilization 10](#_Toc125413702)

[6. Dashboards 10](#_Toc125413703)

[6.1 KPIs (Key Performing Index) 10](#_Toc125413704)

[7. Conclusion 10](#_Toc125413705)

# Introduction

## Purpose

This document provides a comprehensive architectural overview of the South German Bank Credit Prediction . The document depicts the technical aspects of the application and is intended to capture and convey the significant architectural design decisions which have been made in preparation for the development of this web application. This document is also intended to help detect contradictions prior to coding, and can be used as a reference manual for how the modules interact at a high level.

The HLD will:

* Present all the design aspects and define them in detail
* Describe the user interface being implemented
* Describe the hardware and software interfaces
* Describe the performance requirements
* Include design features and the architecture of the product
* List and describe the non-functional attributes like -
  + Security
  + Reliability
  + Maintainability
  + Portability
  + Reusability
  + Application
  + compatibility
  + Resource utilization
  + Serviceability

## Scope

This document seeks to define the high-level functionalities of the web application System Components in as much detail as possible. It presents the structure of the system, such as the database, architecture (layers), application flow (navigation), and technology architecture. The HLD uses non-technical to mildly-technical terms which should be understandable to the administrators of the system.

## Definitions, Acronyms, and Abbreviations

This subsection provides the definitions of all unclear terms, acronyms, and abbreviations required to properly interpret the System Design Document.

Table 1: Definitions, Acronyms and Abbreviations

|  |  |
| --- | --- |
| Term | Definition |
| CSV | Comma Separated Value: A file format for text files that is easily converted to and from Excel Worksheets via Microsoft Excel. |
| HTML | Hypertext Markup Language: is the predominant [markup language](http://en.wikipedia.org/wiki/Markup_language) for [web pages](http://en.wikipedia.org/wiki/Web_page). It provides a means to describe the structure of text-based information in a document – by denoting certain text as headings, paragraphs, lists, and so on. |
| XML | A type of file (or language used when generating files) which allows for data and documents to be easily exchange over the web. |

# General description

## Product Perspective

This is the web application, which classify whether an applicant will default or not based on the number of input features feed to the model. In other words, if a loan amount is given to an applicant, will he/she pay back the loan amount or not? Since a lot of the applications needs to be processed everyday, it will be helpful if there is a predictive model in place which can assist the executives to do their job by giving them a heads up about approval or rejection of a new loan application.

## Problem Statement

Nowadays there are many risks related to bank loans, especially for the banks so as to reduce their capital loss. The analysis of risks and assessment of default becomes crucial thereafter. Banks hold huge volumes of customer behavior related data from which they are unable to arrive at a judgement if an applicant can be defaulter or not. So, there is a need to build a system that should be able to predict whether an applicant will default the loan or not.

## Proposed Solution

Data Mining is a promising area of data analysis which aims to extract useful knowledge from tremendous amount of complex data sets. So, our aim is to design a model and prototype the same using a data set available in the UCI repository. The model is a decision tree-based classification model that uses the functions available in the python library. Prior to building the

model, the dataset is pre-processed, reduced and made ready to provide efficient predictions. The final model is used for prediction with the test dataset and the experimental results prove the efficiency of the built model.

Then it processed the data received, load the model and then sends the results back to the user. First, predictions are made using the final classifiers model. Second, we report different scoring metrics and the ROC curve to show the model performance. On one hand, different classical metrics are derived from the confusion matrix: accuracy, precision, recall. We add to this list the F beta score which balances the recall and the precision metrics by calculating a weighted harmonic mean. To avoid false negatives (FN), we specifically add the F2 score by giving twice as much importance to recall as to precision.

## Data description

The business meaning of each column in the data is as below

* **GoodCredit**: Whether the issued loan was a good decision or bad
* **checkingstatus**: Status of existing checking account.
* **duration**: Duration of loan in months
* **history**: Credit history of the applicant
* **purpose**: Purpose for the loan
* **amount**: Credit amount
* **savings**: Savings account/bonds
* **employ**: Present employment since
* **installment**: Installment rate in percentage of disposable income
* **status**: Personal status and sex
* **others**: Other debtors / guarantors for the applicant
* **residence**: Present residence since
* **property**: Property type of applicant
* **age**: Age in years
* **otherplans**: Other installment plans
* **housing**: Housing
* **cards**: Number of existing credits at this bank
* **job**: Job
* **liable**: Number of people being liable to provide maintenance for
* **tele**: Is the Telephone registered or not
* **foreign**: Is the applicant a foreign worker

## Data Set Information:

Two datasets are provided. the original dataset, in the form provided by Prof. Hofmann, contains categorical/symbolic attributes and is in the file "german.data". For algorithms that need numerical attributes, Strathclyde University produced the file "german.data-numeric". This file has been edited and several indicator variables added to make it suitable for algorithms which cannot cope with categorical variables. Several attributes that are ordered categorical (such as attribute 17) have been coded as integer. This was the form used by StatLog.   
  
The rows represent the actual classification and the columns the predicted classification.   
It is worse to class a customer as good when they are bad, than it is to class a customer as bad when they are good.

# Tools Used

Python programming language and frameworks such as NumPy, Pandas, Scikit-learn are used to build the whole model.

* PyCharm is used as IDE.
* For visualization of the plots, Matplotlib, Seaborn and Plotly are used.
* AWS is used for deployment of the model.
* MongoDB is used to retrieve, insert, delete, and update the database.
* GitHub is used as version control system.

# Design Details

Generally, the goal of a machine learning project is to build a statistical model by using collected data and applying machine learning algorithms to them. Therefore, every ML-based software includes three main artifacts: **Data**, **ML Model**, and **Code**. Corresponding to these artifacts, the typical machine learning workflow consists of three main phases:

* **Data Engineering**: data acquisition & data preparation,
* **ML Model Engineering**: ML model training & serving, and
* **Code Engineering**: integrating ML model into the final product.

## Data Engineering

The initial step in any data science workflow is to acquire and prepare the data to be analysed. Typically, data is being integrated from various resources and has different formats. The data preparation follows the data acquisition step, which is according to Gartner “an iterative and agile process for exploring, combining, cleaning and transforming raw data into curated datasets for data integration, data science, data discovery and analytics/business intelligence (BI) use cases”. Notably, even though the preparation phase is an intermediate phase aimed to prepare data for analysis, this phase is reported to be the most expensive with respect to resources and time. Data preparation is a critical activity in the data science workflow because it is important to avoid the propagation of data errors to the next phase, data analysis, as this would result in the derivation of wrong insights from the data.

The Data Engineering pipeline includes a sequence of operations on the available data that leads to supplying training and testing datasets for the machine learning algorithms:

1. **Data Ingestion** - Collecting data by using various frameworks and formats, such as xlsx, csv, etc. This step might also include synthetic data generation or data enrichment.
2. **Exploration and Validation** - Includes data profiling to obtain information about the content and structure of the data. The output of this step is a set of metadata, such as max, min, average of values. Data validation operations are user-defined error detection functions, which scan the dataset in order to spot some errors. Some checks need to be performed for the smooth continuous training of the model like name of columns, number of columns, data drifts etc.
3. **Data Wrangling (Cleaning***)* - The process of re-formatting particular attributes and correcting errors in data, such as missing values imputation, removing the noise.
4. **Data Splitting** - Splitting the data into training, validation, and test datasets to be used during the core machine learning stages to produce the ML model.

## Model Engineering

The core of the ML workflow is the phase of writing and executing machine learning algorithms to obtain an ML model. The Model Engineering pipeline includes a number of operations that lead to a final model:

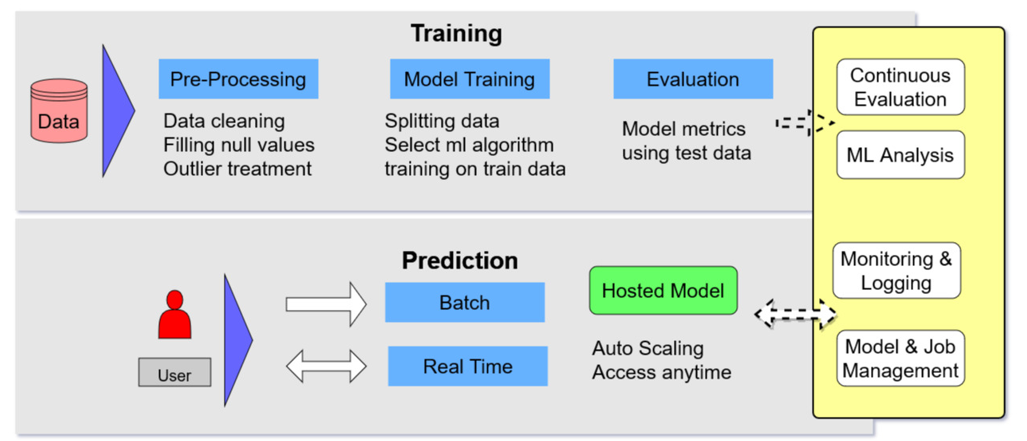
1. **Model Training**- The process of applying the machine learning algorithm on training data to train an ML model. It also includes feature engineering and the hyperparameter tuning for the model training.
2. **Model Evaluation** - Validating the trained model using validation datasets to ensure it meets desired objectives before serving the ML model in production to the end-user.
3. **Model Testing** - Performing the final “Model Acceptance Test” by using the hold test dataset.
4. **Model Packaging** - The process of exporting the final ML model into a specific format which describes the model, in order to be consumed by the business application.

## Model Deployment

Once machine learning model is trained, it needs to be deployed in some cloud platform in like AWS, GCP. Also, it can be integrated with a business application such as a mobile or desktop application. The ML models require various data points (feature vector) also called predictors to produce predictions. The final stage of the ML workflow is the integration of the previously engineered ML model into existing software. This stage includes the following operations:

1. **Model Serving**- The process of addressing the ML model artifact in a production environment.
2. **Model Performance Monitoring** - The process of observing the ML model performance based on live and previously unseen data, such as prediction or recommendation. In particular, we are interested in ML-specific signals, such as prediction deviation from previous model performance. These signals might be used as triggers for model re-training.
3. **Model Performance Logging** - Every inference request results in the log-record.

## Process Flow



## Event Log

The system should log every event so that the user will know what process is running internally. Python’s logging library provides a more complete solution for debugging and auditing your applications. Python’s logging library allows you to easily add metadata to your logs, such as timestamp, module location and severity level (DEBUG, INFO, ERROR etc.). This metadata is automatically added without having to hard code it into your statement. The logging library allows you to save logs in different formats including to a file. Useful for recording the logs for future analyses. You can also send the logs to multiple locations at the same time.

**Initial Step-By-Step Description:**

* The logging has been done of the process wherever required
* The system will generate a separate file for logging where we can be observed every step
* Logging enables us to easily debug issues

## Error Handling

Since writing a code without error in one go is almost impossible so we need to write a program such that catching an error is easy. For that only we have used custom exception handling along with inbuilt exception class of python. Exception can be helpful in case you know a code which can produce error in the program. You can put that code under exception and run an error free syntax to execute your program. This helps in preventing the error from any block which is capable of producing errors. Since exception can be fatal error, one should use exception syntax to avoid the hurdles.

Python itself contains a number of different exceptions. The reasons can vary as per the exception. Here, we are listing some of the most common exceptions.

**ImportError:** When any object to be imported into the program fails.

**Indexerror:** The list being used in the code contains an out-of-range number.

**NameError:** An unknown variable is used in the program. If the program does not contain any defined by the user and the used variable is also not pre-defined in the Python, hence this error comes into play.

**SyntaxError:** The code cannot be parsed properly. Hence it is necessary to take various precautionary measures while writing the code.

**TypeError:** An inappropriate type of function has been used for the value.

**ValueError:** A function has been used with correct type however the value for the function is irrelevant.

These errors are some of the most common ones and are required to be used, in case any sort of exceptions is required to be used. Python also has assertion feature which can be used for raising exceptions. This feature tests an expression, python tests it. If the result comes up false then it raises an exception. The assert statement is used for assertion. Assert feature improves the productivity of the exceptions in python. However, for now we will just focus upon raising exceptions with the use of raise statement.

# Performance

## Reusability

when we talk about the code reusability the first picture which comes into the mind is functions irrespective of the language. Take some lines of code, give them a name, and you’ve got a function which can be reused by calling the function while passing the appropriate arguments requirements. And take a collection of functions and package them as a file, and you’ve got a **module** which can also be reused by using import statements in python. Since in real life scenarios we need to manage a hundred or thousands of lines of code so we have to thing of strategies to reduce the complexity of code has written.

Python supports **modularity,** in that you can break large chunks of code into smaller, more manageable pieces. The code of this project has been written in modular fashion with proper documentation wherever required. The source code divided into various components namely data ingestion, data validation, data transformation, model trainer, model evaluation and model pusher. Also, separate files for project configuration and deployment.

## Application Compatibility

The different components for this project will be using python as an interface between

them. Each component will have its own task to perform, and it is the job of the python

to ensure proper transfer of information.

## Resource Utilization

When any task is performed, it will likely use all the processing power available until that

function is finished.