Task Report

# Introduction

This report is a summary and explanation of code of the Rangeme Data scientist Task. There are 4 Sections in this report. Section one explains the stage SQL tasks. Section two explains the basic data clean process included in the Stage tasks. Section three shows how I design the whole prediction system. While section four gives finding and conclusion.

# Section One - SQL

In order to run SQL on the task dataset without a database, I load raw data into pandas DataFrame and use pandasql package to execute sql queries. In Stage1.py, for each sub task, query is provided for review, and you will see the executed result in both console and Data/Stage1Output.log file.

# Section Two - Data Clean

The purpose of data clean is to identify and fix invalid values in order to improve data quality and model accuracy.

1. Identification

From the view of data scientist, there are two types of data for modelling purpose, numeric and category data.

For numeric data, in general, the invalid value will be a null or empty. If the data is not accurate, such as a right value is 0.9 but we have 0.3 in the field, it is hard to identify them as a user of the data. The upstream data engineer should guarantee the quality by having data validation or providing data quality report to data scientist in order to make comprehensive decision. This is not the discussion in this project.

For category data, the invalid value can be empty or a special value. This this task, I notice that the proportion of value 999999 in column *company\_state* and value INVALID in *column company\_years\_trading* are strongly different from the others. So I consider these two are the invalid values.

1. Fixing

There are two major steps used for data fixing.

1. Formatting. This step fixes the type of each field in dataset. For numeric columns, I remove the symbols from the number string and cast all the types to float. For category columns, the invalid values will be set to null for further process. The reason to do so is to keep invalid values in both numeric and category columns the same, and later can use an unify function to deal with all the invalid values.
2. Imputer. The simplest way to clean the empty value from the dataset is to remove the row which contains the empty value. However, from experience, this method will reduce the accuracy of prediction model, especially for the low-quality data. In the imputer module, an existed value is assigned to the empty fields. Meanwhile, For each column, a new ‘\_missing’ column is populated, which contains value of True or False to mark the invalid value in corresponding columns. In stage 3, in detail, category columns will take the most frequency value in train dataset as replace and number columns use median value.

An extra step of data clean, which is low variance column removing, will be introduced in stage 3. Because this step is after encoding, which transferred the meaning of columns of original dataset.

# Section Three - Modelling

A screenshot of a social media post

Description automatically generated

The figure above shows the system I build for predicting the upgrade. Pre-processing pipeline is for preparing features for the following xgboost model. In the pipeline, The formaters and imputers are introduced in the previous section.

Input data will be split into X\_train, X\_test, Y\_train, Y\_test for modelling training and validation purpose respectively. Since the input data is fixed, the volume of test data is set to 30% of whole dataset. In practice, upper and lower thresholds of test data size should be considered.

The feature selector splits input data into category and numeric parts.

Category encoder uses one-hot encoder to represent the category values into 0s and 1s, so it can be consumed by the model.

In the filters, the zero-variance columns will be removed. In this step, 8 columns are removed.

Feature Concat joins the two set of features together.

Since the target upgrade has a value in 0 and 1, this is a classification problem. Moreover, all the data are structured, XGBClassifier is the best option in all the state-of-the-art models. In order to overcome overfitting problem, early stop is used.

The system can consume dynamic inputs by using a configurable file, params.json.

The file defines which are the category and numeric columns, and also the hyperparameters for the model. In real production environment, It can keep delivery the best model to fit the changes in data.

Feature weighting shows how important of the input features. Ideally, this module’s output should feedback to update params in order to improve model automatically. Regarding of project complexity, this is not included.

Similarly, Automatic tuning model which is to find the best hyperparameters is not covered in this project.

Section 4 Results and finding

The model accuracy on the test dataset is **95.97%.**

The most import feature is **supplier\_retailer\_submissions\_count**, which takes a portion of **96.68%**. It means that if we only use this feature to predict the potential upgrades, we still get the similar accuracy. From the data, we also observe that the supplier who has higher submissions count is more liked to be upgraded.

If there is only one relevant feature contributing to accuracy, the system losses stability and accuracy will go down when more data comes.

If we want to improve the accuracy and stability of the system, it is wise to,

1. Keep the features which has low or none coefficient to **supplier\_retailer\_submissions\_count** when computation resource is enough.
2. Remove the high coefficient but low weight features.
3. Involve new features of different dimension which can explain the submission count. For example, retailer feedback (in average, max, min, std scores) of the submissions.

Loyalty score of retailers to the supplier. Content featuring is a good option. By mapping the relation between retailer feedback and submission content, it helps to understand why the submission is accept or reject. It includes not only numbers in the content, but also features like image quality, product description accuracy and so on.

Another finding from this task is a cold start problem. The dataset is good for providing insight but not good enough for prediction purpose, even the historical counts represents the good matching to the target. It takes time to collect the count information. For a new joined supplier, we are not able to tell from the model when and how we can upgrade the supplier. Instead of featuring supplier by the count values, using frequency in a proper time window and abstract user behaviour with clickstream chains to predict will explain better why the supply is upgraded and will also guidance supplier operations on the platform.