Loan Delinquency Prediction

Project Report

MF810: FinTech Programming

Professor Jun Fan

May 5th, 2022

Boston University, Questrom School of Business

Daniel Babitskiy

Haozhi Chen

Table of Contents

Abstract

Predicting loan delinquency is a critical way in which to be able to have insights on the likelihood of someone defaulting on their payments owed, in this case, loan payments. Loans are considered as delinquent when the borrower makes late payments, even if just by a day, or misses a regular installment payment or payments. The prediction that is made in regard to loan delinquency becomes critical when considering that once there are consumers who are repeat offenders, the probability of the borrower actually defaulting on their debt is much greater. Defaulting on a loan is a concluding consequence of extended payment delinquency, in which the borrower fails to keep up with their ongoing loan obligations or conversely, doesn't repay the loan according to the terms laid out in the promissory note agreement.

The purpose of this project is to use consumer financial data to predict the prospect that a borrower will default on their loans, based on their historical borrowing background. Within the project, we examined the plausibility of processing large datasets utilizing tools such as pyspark, spark SQL, and Google Cloud. Additionally we used various classification models such as Logistic Regression, Naive Bayes, Linear Support Vector Machine, Naïve Bayes, Decision Tree, Random Forest, Gradient-Boosted Tree, MLP Classifier, One-vs-Rest and Factorization Machines in order to classify our findings and aid our our overall objective of predicting consumer loan delinquency.

I. Data

We chose to use Kaggle, a reputable online collection of datasets and models in a web-based data-science environment, in order to search for a dataset for loan delinquency prediction. Given that Kaggle has received many contributions from data scientists and machine learning practitioners, the datasets found there are helpful in identifying potential financial problems. The specific dataset that we obtained is called AlfaBattle 2.0, roughly 5 gigabytes in size and holds 102 Apache Parquet files which are designed for efficient data storage and retrieval, where each file includes 20 variables, including:

- app_id: loan application identifier
- amnt: normalized value of the logarithm of the transaction amount
- currency: Transaction currency
- operation_kind:
- card_type: Unique identifier of the card type (Visa Classic, Visa Gold, Visa Platinum, etc.)
- operation_type: Identifier of the type of operation on a plastic card
- operation_type_group: Card transaction group identifier (Debit Card (DR), Credit Card (CR), NOOP)
- ecommerce_flap: E-commerce flag
- payment_system: Payment system type identifier
- income_flap: Sign of debiting / depositing funds to the card card
- mcc: Unique identifier for the outlet type
- country: ID of the country of the transaction
- city: Transaction city ID

- mcc_category: Transaction store category ID
- day_of_week: Day of the week when the transaction was committed
- hour: The hour the transaction was committed
- days_before: The number of days before taking the loan issue date
- weekofyear: Number of the week of the year when the transaction was committed
- hour_diff: Number of hours since the last transaction for this client
- transaction_number: This is the #th transaction done by this client

as well as 3 CSV files, which include:

- Train_target which contains the unique identifier for each user as 'app_id', the one product each user chosen as 'product', and the binary indicator for delinquency as 'flag'.
- Test_target which contains the unique identifier for each user 'app_id', and the one product each user chosen 'product'
- sample which contains the unique identifier for each user 'app_id', and the dummy indicator for delinquency 'flag' (currently 0.5)

Through data visualization, the 'train' and 'test' are already in tidy data format. Three principles from Wickham (2014) stated that each variable forms a column, each observation forms a row, and each type of observational unit forms a table. Also, all variables are already pre-transformed into numerical form, so there's no further data transformation that needs to be done at the first glance.

However, when specifically picking one user (uniquely identified with app_id)out, say app_id = 5, and looking over his/her data, it was found that each user in 'train' and 'test 'has one or more transactions. Since users have different numbers of transactions, performing exploratory data analysis based on the current 'train' does not tell the story for users. Because users have different amounts of transactions, they weigh differently on graphs. With the fact that each transaction are shown as a row, putting raw variables into classification models will not generate accurate prediction. Thus, each user's information should be contained in one row only. We will talk about the implementation in the methodology section.

I. Methods

We first imported the required packages, including pyspark.sql, pyspark.mllib, and pandas. Given that our code is able to run in Google Cloud, it is not necessary to mount to Google Drive as the files could be uploaded onto the designated bucket that is attached to the project. One other benefit is that pyspark could also be selected from Google Cloud instead of being initialized through sparksession.

Considering there are 50 files for the training set, as well as 50 additional for the testing set, all of the files are looped through and appended into the dataframes 'train' and 'test'. Since the dataset is large, we used the spark dataframe and Google Cloud platform in order to run our models. The variable that is needed to be predicted was saved in separate files. At this point, those files were imported as 'train_target', 'test_target', and 'sample'.

After we imported the dataset as spark dataframe, we first visualized each data frame by using .show(), then we checked for nulls and nans for each column using isnan() and isnull(). We further checked for distinct variables within each column by using select(‘column_name’).distinct().count().

When specifically picking one user out (uniquely identified with the app_id variable), for example app_id = 5, and looking over his or her data, it was seen that users in the 'train' and 'test’ datasets 'had one or more transactions. Each user's information should be contained in one row only. In order to do that, all of the transaction records for each user were summarized and subsequently, variables were formed to describe all of the transactions, in order to fit all information into 1 row for each user.

Specifically, transaction amounts per user were described by using: count, min, max, average, and variance. To accomplish this, spark sql was imported to write a sql select query given that it was a spark dataframe. A select query was used to select and calculate the new variables for each app_id.

Additionally, given that there are 5 products and each user only has one product associated with them, the 'train' was separated into 5 sub-trains with each sub-train targeting one product. Therefore, 5 models will be performed for each classification model included. At this juncture, we only pick group 0 and transaction counts larger than 1000 in order to demonstrate our model.

As all data was sorted out, some exploratory data analysis could be performed. The correlation matrix was calculated for all of the variables with printed-out numbers. In order to better visualize the relationship, we converted the spark dataframe into a pandas dataframe and used seaborn to plot the correlation matrix. We have also examined the skewness and kurtosis for most variables.

Firstly, the Logistic Regression classification model was used. Logistic Regression is a classification algorithm which is used to identify the probability of success or failure of occurrences. It is used when the dependent variable is essentially a binary variable (0/1, true/false, yes/no). It supports the classification of data into discrete classes by examining the relationships of a given labeled data set. Although logistic regression is mainly used for dichotomous dependent variables, the technique can be extended to cases involving outcome variables with 3 or more classes known as multicategorical or multinomial dependent variables (Wright, 1995). Based on the training results, a prediction model was obtained with an accuracy level of 96.9241% which can effectively help us in loan delinquency prediction.

Secondly, a One vs Rest classification model was used. One vs rest, known as OvR for short, also known as One vs All or OvA, is a heuristic for multi-class classification using a binary classification algorithm. It involves splitting a multi-class data set into multiple binary classification problems. A binary classifier is then trained for each binary classification problem and the most reliable model is used for the purpose of making predictions. The goal of a One vs Rest (OVR) classification model is to distinguish a single category of interest from other categories. In OVR, the notion of novelty detection and robustness to dataset movement becomes critical when the range of rest classes is extended from classes observed during training to unseen and potentially irrelevant classes. (Lübbering et al., 2021). In light of this, the model is very well suited to our dataset in order to avoid overfitting and good data splitting. Also due to this reason, we obtained a 97% level of accuracy.

Factorization Machines (FM) are models capable of estimating the interactions between features. Factorization Machines are a general-purpose supervised learning model that can map arbitrary real-valued features to a low-dimensional latent factor space and can be naturally applied to a variety of prediction tasks, which include regression, classification, and ranking. Similarly to support vector machines (SVMs), FMs are general-purpose predictors that can handle any real-valued feature vector. Unlike SVMs, FMs use decomposition parameters to model all interactions between variables. As a result, they are able to estimate interactions even in huge sparse problems where the support vector machines fail, for example, in recommendation systems. We demonstrated that the model equations of FMs can be computed in linear time, thus allowing for direct optimization of FMs. Thus, unlike nonlinear support vector machines, no transformation in binary form is required and the model parameters can be estimated directly without using any support vectors in the solution. Therefore, in terms of efficiency of model training, we are able to obtain results more quickly because this is not a linear and single-threaded training approach. We obtained FM models with an accuracy of

Conclusion

Based on a thorough and extensive usage of spark and various classification models, we were able to train and test the models and get a decent accuracy in testing the training set.

Some further improvements that we could make are using unsupervised learning as well as training the entirety of the train dataframe in order to improve the accuracy of our results. We could also use the area under the curve to prove the accuracy of our results. We could also use the for loop in order to read the train and test datasets in a more efficient manner.