Risk Prediction for Home Equity Line of Credit (HELOC)

Wanyu Cheng wanyu.cheng@ur.rochester.edu
Qingyu Liu qingyu.liu@ur.rochester.edu
Tong Niu tong.niu@ur.rochester.edu
Yunqing Yu yunqing.yu@ur.rochester.edu
Yujia Zhang yujia.zhang@ur.rochester.edu

December 19th, 2019

1. Define the problem & background

Recent years, more and more financial institutions are taking advantage of different credit scoring methodologies to evaluate the risk in billions of dollars in loans.The FICO score is used in more than 90% of lending decisions in the US. Credit score is an important factor that financial institutions consider when deciding whether or not to approve a loan. The scores are designed to predict the likelihood of loan repayment. Recently, the rise of artificial intelligence began to be trending and advanced machine learning methods are widely applied throughout the financial services industry. Faced with large and complex datasets, the challenge is always constructing interpretable global models and tuning effective predictive models.

This project aims at building a machine learning model that can use the information about the applicants in their credit report to predict whether they will repay their Home Equity Line of Credit (HELOC) account. This prediction result is then used to decide whether the homeowner qualifies for a line of credit and, if so, how much credit should be extended.

We are assigned with a real-world financial dataset from an anonymized credit bureau. The 23 predictor variables are all quantitative or categorical. The target variable that we are going to predict is a binary variable called *RiskPerformance*. The value “Bad” means that a consumer was 90 days past due or worse at least once over a period of 24 months from when the credit account was opened. The value “Good” indicates that they have made their payments without ever being more than 90 days overdue.

2. Data cleaning

1) Checking and converting the data types

The target to predict the risk is the RiskPerformance variable. The original data consists of two values: good and bad. RiskPeformance is factorized to 0 and 1. 0 represents bad and 1 represents good.

For variables MaxDelq2PublicRecLast12M and MaxDelqEver, they are categorical variables that represent related records of max delinquency. These variables are converted into categorical variables to facilitate model building.

All other variables remain as a numeric type.

2) Dealing with special values

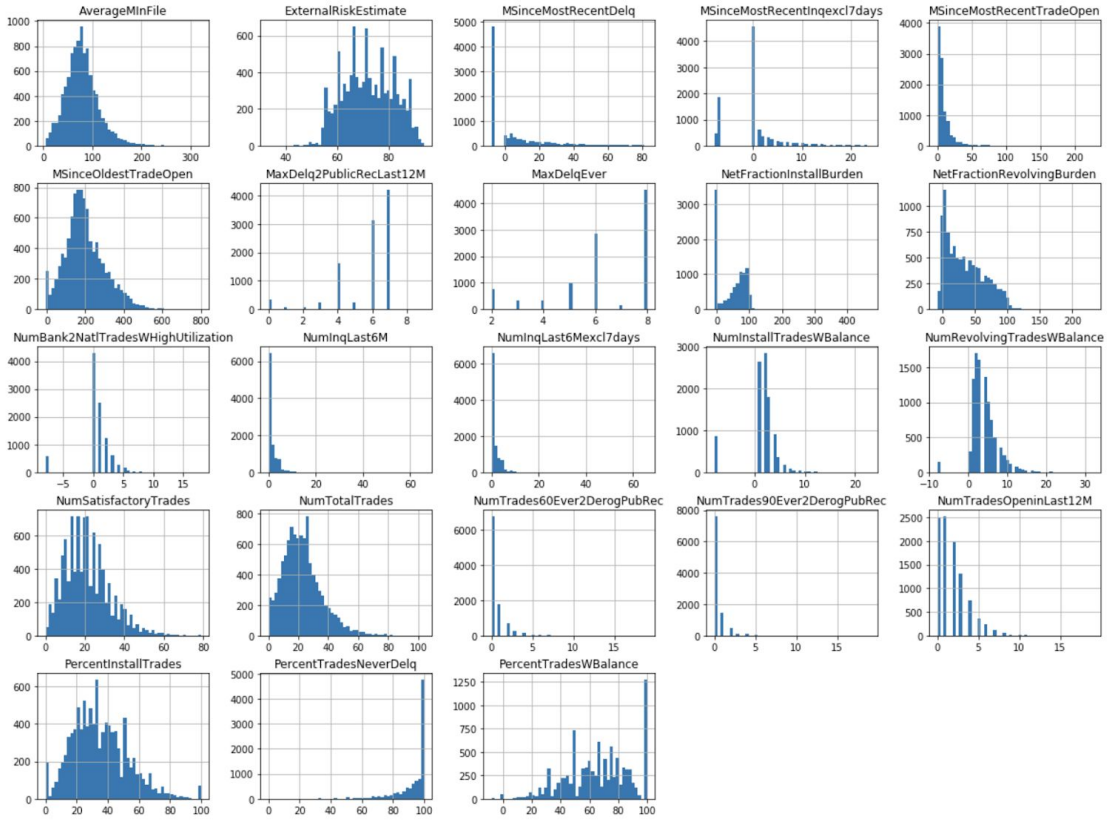
Special values, i.e. -7, -8, -9, are changed according to their meaning. For -9, the original meaning is no bureau record or no investigation. 588 rows are all with -9 and there are only 10 rows with -9 in some of the features. Thus, records that contain -9 are changed as the missing value NaN and dropped.

For -7 and -8, the meaning is not the same as missing, but more as invalid values. Thus, -7 and -8 are replaced as the mode of a variable.

3) Building a data dictionary

A dictionary is built to store the original data, cleaned data, feature names, and factorized target value. Such a dictionary can facilitate future data manipulation, store original and cleaned data, and avoid confusion of all predictors and target value.

3. Building and Tuning Models



After data cleaning, Standardscaler is used to smooth the magnitude of different features. From the chart above, we could see some continuous features are between [0,800] while others [0,20] (excluding special values). By standardizing features, their distributions are close to normal distribution. The machine learning models can benefit from the same scale.

We chose five machine learning models to fit the data: Support Vector Machine (SVM), Logistic Regression (LR), K-Nearest Neighbors (KNN), Random Forest (RF), and Boosting.

In the tuning part, we chose 3-fold cross-validation to avoid overfitting. To use RandomizedSearchCV for tuning hyper-parameter, we first create a parameter grid to sample during fitting. For SVC and logistic regression, we set a penalty to [0.1,1] to realize a more robust model; and try linear, polynomial and RBF kernels to see the model performance.

Random forest and Boosting are ensemble methods for improving model predictions. We set n_estimators, the number of decision trees in each forest, around 100. Bootstrap is set True, which enables the replacement of data points.

After trying different combinations, we found the best model.

4. Model Comparison and Selection

	Classifier	Best Parameter	Dataset	Test Score	CV Score
0	Boosting	{'algorithm': 'SAMME.R', 'base_estimator': Non...	NaN	0.712895	0.731981
1	KNN	{'algorithm': 'auto', 'leaf_size': 30, 'metric...	NaN	0.678427	0.694523
2	LR	{'C': 0.1, 'class_weight': None, 'dual': False...	NaN	0.715734	0.736308
3	RF	{'bootstrap': True, 'class_weight': None, 'cri...	NaN	0.722628	0.732792
4	SVM	{'C': 1, 'cache_size': 200, 'class_weight': No...	NaN	0.711273	0.737120

According to the model comparison output, we choose Random Forest as the best model, which has the highest test score of 0.722628, meaning the highest classification accuracy (using Gini) in the test data set. Random Forest also has a high cross-validation score of 0.732792, which means the average accuracy score of 3 grid search cross-validations (cv = 3). The CV score is close to test score, which means that this model is robust.

The best parameters used in the Random Forest model are: using bootstrap method (bootstrap = True); number of decision trees is 100 (number of estimators = 100); the maximum number of features to consider when looking for the best split is 0.2 (max.feature = 0.2). The advantages of using Random Forest model are the following: it runs efficiently on large datasets; it can handle large amount of input variables without variable deletion.

5. Interface

We use the Streamlit package to design an interface to visualize the model. The aim of creating this interface is to allow any customer without much or any knowledge about ML to manipulate and to make predictions on the new dataset.

To create the interface, we follow a two-step roadmap. The first script: *model_training.py* is used to train models on the datasets that we already have and save the well-tuned parameters of the model into certain files for future use. The second script *streamlit_demo.py* is used to load the model that we already save and visualize the performance of the model.