LOAN DEFAULT PREDICTION SYSTEM

Juan Antonio Morales Jiménez
4th July 2018

CONTENT

1 – Introduction

Lending Club is a peer to peer lending company based in the United States, in which investors provide funds for potential borrowers and investors earn a profit depending on the risk they take (the borrowers credit score). Lending Club is not a bank but they provides the "bridge" between investors and borrowers.
This system can make credit more affordable for the borrowers and investing more rewarding for investors.

Objectives
The main objective of the project is to develop a scoring system consisting of several machine learning models to predict whether a loan will be paid or not.

As secondary goals, we can mention the followings:

- Analyze which features are the most important to predict if borrowers will pay the loan.
- To carry out an exploratory data analysis in order to classify the type of users of the platform (most common debt purposes, employee length and types of jobs borrowers have).
- Learning a lot and have fun :)

2 – Data Description

Lending club provides data on their web page (www.lendingclub.com) that contain complete loan data for all loans issued from 2007 to last 2017 quarter, including the current loan status (Current, Late, Fully Paid, etc.) and latest payment information. Some additional features such as credit scores, number of finance inquiries, zip codes, states, or collections have been included among others.

The complete data set is made up of **150 variables** and **1646801 observations**, although there are some variables with just one value and other ones with too many NA's.

In this stage we have selected the variables that, reading the data dictionary, seemed more significant for a first approximation to the data, i.e., loan amount, date, borrower annual incomes, loan grade, interest rate, loan status, etc. The clean final dataset has the following **50 variables:**

LoanStat	Description
num_bc_sats	Number of satisfactory bankcard accounts
num_rev_tl_bal_gt_0	Number of revolving trades with balance >0
grade	LC assigned loan grade
avg_cur_bal	Average current balance of all accounts
pub_rec_bankruptcies	Number of public record bankruptcies
num_rev_accts	Number of revolving accounts
tax_liens	Number of tax liens
funded_amnt_inv	The total amount committed by investors for that loan at that point in time.
delinq_2yrs	The number of 30+ days past-due incidences of delinquency in the borrower's credit file for the past 2 years
total_bal_ex_mort	Total credit balance excluding mortgage
pct_tl_nvr_dlq	Percent of trades never delinquent
disbursement_method	The method by which the borrower receives their loan. Possible values are: CASH, DIRECT_PAY
fico_range_low	The lower boundary range the borrower's FICO at loan origination belongs to.
verification_status	Indicates if income was verified by LC, not verified, or if the income source was verified
delinq_amnt	The past-due amount owed for the accounts on which the borrower is now delinquent.

purpose	A category provided by the borrower for the loan request.
emp_title	The job title supplied by the Borrower when applying for the loan.
zip_code	The first 3 numbers of the zip code provided by the borrower in the loan application.
loan_amnt	The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.
installment	The monthly payment owed by the borrower if the loan originates.
fico_range_high	The upper boundary range the borrower's FICO at loan origination belongs to.
annual_inc	The self-reported annual income provided by the borrower during registration.
term	The number of payments on the loan. Values are in months and can be either 36 or 60.
int_rate	Interest Rate on the loan
emp_length	Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.
loan_status	Current status of the loan
revol_bal	Total credit revolving balance
application_type	Indicates whether the loan is an individual application or a joint application with two co-borrowers
num_bc_tl	Number of bankcard accounts
num_sats	Number of satisfactory accounts
tot_hi_cred_lim	Total high credit/credit limit
tot_coll_amt	Total collection amounts ever owed
initial_list_status	The initial listing status of the loan. Possible values are – W, F
bc_open_to_buy	Total open to buy on revolving bankcards.
total_bc_limit	Total bankcard high credit/credit limit
open_acc	The number of open credit lines in the borrower's credit file.
revol_util	Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.
pub_rec	Number of derogatory public records
funded_amnt	The total amount committed to that loan at that point in time.
num_il_tl	Number of installment accounts
addr_state	The state provided by the borrower in the loan application
num_accts_ever_120_pd	Number of accounts ever 120 or more days past due
total_il_high_credit_limit	Total installment high credit/credit limit
bc_util	Ratio of total current balance to high credit/credit limit for all bankcard accounts.
percent_bc_gt_75	Percentage of all bankcard accounts > 75% of limit.

sub_grade	LC assigned loan subgrade
mort_acc	Number of mortgage accounts.
num_op_rev_tl	Number of open revolving accounts
dti	A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan, divided by the borrower's self-reported monthly income.
home_ownership	The home ownership status provided by the borrower during registration or obtained from the credit report. Our values are: RENT, OWN, MORTGAGE, OTHER

To decide whether to keep a feature or not has been one of the most critical parts of the project since we have some information in the dataset related to time or previous payments that we will not have for new borrowers and they reveal much information about loan status. For example, if "*debt_settlement_flag*" is "Y", it implies that the borrower charged off, or if "*total_pymnt*" is greater than "*loan_amnt*", then the loan must be paid. For this reason, **only variables that don't update over time will be kept**. An Excel document called "*features_dict.xlsx*" with the initial variables and those that have been kept for modeling has been included in the Data folder of the github repository.

Files:

- *doc/01-getting_and_cleaning_data.ipynb*
- *src/data_acquisition.py*

3 – Methodology

The project in the following stages:

3.1 – Exploratory Data Analysis for a first approach to the problem.

3.2 – Data Preprocessing and feature engineering.

3.3 – Model training and evaluation for "*loan_status*" prediction.

3.4 – Main program implementation.

3.5 – Deploy prediction system as API.

Programming lenguages:

- **Linux shell**: Shell was used mainly to manage files and run scripts.
- **R**: It was used as a first quick approach to the project and to perform some hypothesis tests.
- **Python** for the Exploratory Data Analysis and the modeling phase. **Jupyter Notebooks** were used as first exploration to the data and to the model hyperparametrizations over the sample data set and Spyder IDE for source code implementation. Python was also used to deploy prediction system as API. **Anaconda** distribution was used for creating workflows and keeping the dependencies separated out.
- **Tableau** with the main results of the models.

3.1 – Exploratory Data Analysis

The aim of this phase was to get familiar with the data and study the relationship between those features we can all understand, paying special attention to the loan status since it is the variable that we want to predict through the models.

Lending Club user interests have been analyzed in this stage as well, studying the most common debt purposes, employee lengths and types of jobs borrowers have.

Files:

- *doc/02-exploratory_data_analysis.ipynb*

3.2 – Data Preprocessing

3.2.1 – Data Selection

In this stage we selected the variables we used for feeding machine learning models. Cleaning columns with just one value, selecting those features with less than 25% of NA's and removing the variables that don't update over time caused the final dataset to be left with 50 features.

loan_status is the variable we wanted to predict using machine learning methods. Initially, this variable had several categories, most of them related to loans in progress. For the purpose of this project we considered only finished loan categories:

- **Fully Paid** coded as 0. This is a paid loan.
- **Charged Off** coded as 1. This is considered as unpaid loan.

Among the variables that LendingClub provides on their webpage, there are two of them very correlated to the loan_status. They are the *last_fico_range_low* and *last_fico_range_high* which correspond with the lower and upper boundary range the borrower's last FICO pulled belongs to. Only using these two variables and the target, we achieved almost 100% accuracy and 0.99 of area under the curve ROC, regardless the type of model or the parameters of the model we trained.

Nevertheless, we don't know if we can use these two variables as training features since it is possible that one person FICO range changed once she/he had unpaid the loan ("*Correlation does not imply causation*").

Some hypothesis tests have been carried out to decide whether uses these variables and according to the results of the t-tests, we decided to remove them.

3.2.2 – Pre-processing numeric variables

Numerical variables were identified and just two actions were carried out:

- Fill **NA's** to the median of the variable.
- **Outliers:** Reject those observations which *z_score* (number of standard deviations from the mean a data point is) was greater than 3.5 in, at least, one variable.

3.2.3 – Pre-processing categorical variables

First, we filled NA's to the last valid observation in order to get a random filling of NA's. We tested two types of transformation for categorical variables:

- Transform categorical variables to **One Hot Encoding**, what it makes that the number of columns of the final data set were too big. Furthermore, it was impossible to compute some features such "*zip_code*" or "*emp_title*" because the number of categories were too many for these categories.
- Transform each categorical variable to numeric replacing each category with the mean of the target for each category (**ratio unpaid/paid loans per category**).

Finally, last option was chosen since better results were obtained.

Files:

Hypothesis tests:

- *doc/Appendix-ttestsLastFicoRange.ipynb*
- *src/R/ficoStudy.R*

Pre-processing:

- *doc/03-processing_features.ipynb*
- *src/cleaning_data.py*

3.3 – Model training and evaluation

We have been working with four algorithms:

- Logistic Regression.
- Random Forest.
- XG Boost.
- Neural Network autoencoder for feature extraction and Logistic Regression for predicting.

For each model, we have performed the following steps:

1. Normalization of the data if needed.
2. Parameter tuning with Cross Validation.
3. Model evaluation.

Let's discuss one by one:

3.3.1 – Normalization

Normalization only have been carried out in logistic regression model and neural network autoenconder. Three kind of normalization scalers have been tested: min-max, standard and robust. Best results have been obtained with standard scaler for logistic regression and min-max for neural network autoencoder.

Input normalization for gradient-based models such as neural nets is critical. For decision trees (Random Forest and XG Boost) normalization should have no impact on their performance. It is generally useful, when you are solving a system of equations, least squares, etc, where you can have serious issues due to rounding errors. In decision tree, you are just comparing stuff and branching down the tree, so normalization would not help.

3.3.2 – Hyperparameter tuning

Hyperparameter tuning refers to the shaping of the model architecture from the available space. For doing that, we have used **Cross Validation** with 3 k-folds in order to not have too many model fits. In the basic approach, the training set is split into k smaller sets . The following procedure is followed for each of the k "folds":

- A model is trained using k-1 of the folds as training data.
- The resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure such as accuracy).

The performance measure of the model is then the average of the values computed in the loop. The metric that have been used is the **area under the curve ROC** (AUC) which features true positive rate on the Y axis, and false positive rate on the X axis. This means that the top left corner of the plot is the "ideal" point - a false positive rate of zero, and a true positive rate of one. This is not very realistic, but it does mean that a larger area under the curve (AUC) is usually better.

To obtain the best parameters of each model the following hyperparameter tuning method have been used:

- **Grid Search** for logistic regresion. As we only tuned one hyperparameter (regularization strength), we have used this technique, where we simply build a model for every combination of various hyperparameters and evaluate each model. The model which gives the highest accuracy wins.
- **Randomized Search** for random forest and XG boost, where we had to tune several hyperparameters. As random values are selected at each instance, the

chances of finding the optimal parameter are comparatively higher in random search because of the random search pattern where the model might end up being trained on the optimized parameters without any aliasing.

- **Bayesian Optimization** for neural network autoencoder. We have chosen this method since it is adequate for situations where sampling the function to be optimized is a very expensive endeavor such an autoencoder neural network can be. Bayesian optimization works by constructing a posterior distribution of functions (gaussian process) that best describes the function you want to optimize. As you iterate over and over, the algorithm balances its needs of exploration and exploitation taking into account what it knows about the target function. At each step a Gaussian Process is fitted to the known samples (points previously explored), and the posterior distribution, combined with a exploration strategy (such as UCB (Upper Confidence Bound), or EI (Expected Improvement)), are used to determine the next point that should be explored.

3.3.3 – Model evaluation

Once the hyperparameters have been chosen for each model, this has been evaluated with cross validation score and 5 folds. In this case, we have used accuracy, auc, precision and recall as metrics (cross validation scores).

Files:

- doc/04a-logistic_regression.ipynb
- doc/04b-random_forest.ipynb
-doc/04c-XGBoost.ipynb
-doc/04d-neural_network_autoencoder.ipynb

3.4 – Main program implementation

For the training model phase, we have used Jupyter Notebooks with a sample of 20% of the data, since cross validation for hyperparameter tuning is computationally very expensive and an interaction with the programming language is necessary. Notebooks work very well for this function, however they are not enough, specially when the code gets a bit of complexity.

For the model finishing/refinement, I think it is better a program implementation of the complete process. This way allows us to evaluate the impact in the final result little

changes in the complete process and, finally, a final model scoring with the whole dataset (80% of the whole data for training and 20% for testing).
That is what I have done in the *src* folder of the repository, a main program implemented with all the insights I found out in the *documentation* folder with a sample of the data.

Files

- *src* folder

3.5 – Deploy prediction system as API

An API is a (hypothetical) contract between 2 softwares saying if the user software provides input in a pre-defined format, the later with extend its functionality and provide the outcome to the user software.
We have used this option to implement machine learning models due to they cater to the needs of developers / businesses that do not have expertise in ML, who want to implement ML in their processes or product suites. For doing this, we have used **Flask**, a web framework in Python.

To query predictions in a new dataset, it is neccesary to run the API locally at: `gunicorn --bind 0.0.0.0:8000 src.server:app` and then calling the API with the following command in python:

```
requests.post("http://0.0.0.0:8000/predict", data = json.dumps(new_data),
                        headers = {'Content-Type': 'application/json',
                                   'Accept': 'application/json'}).json()
```

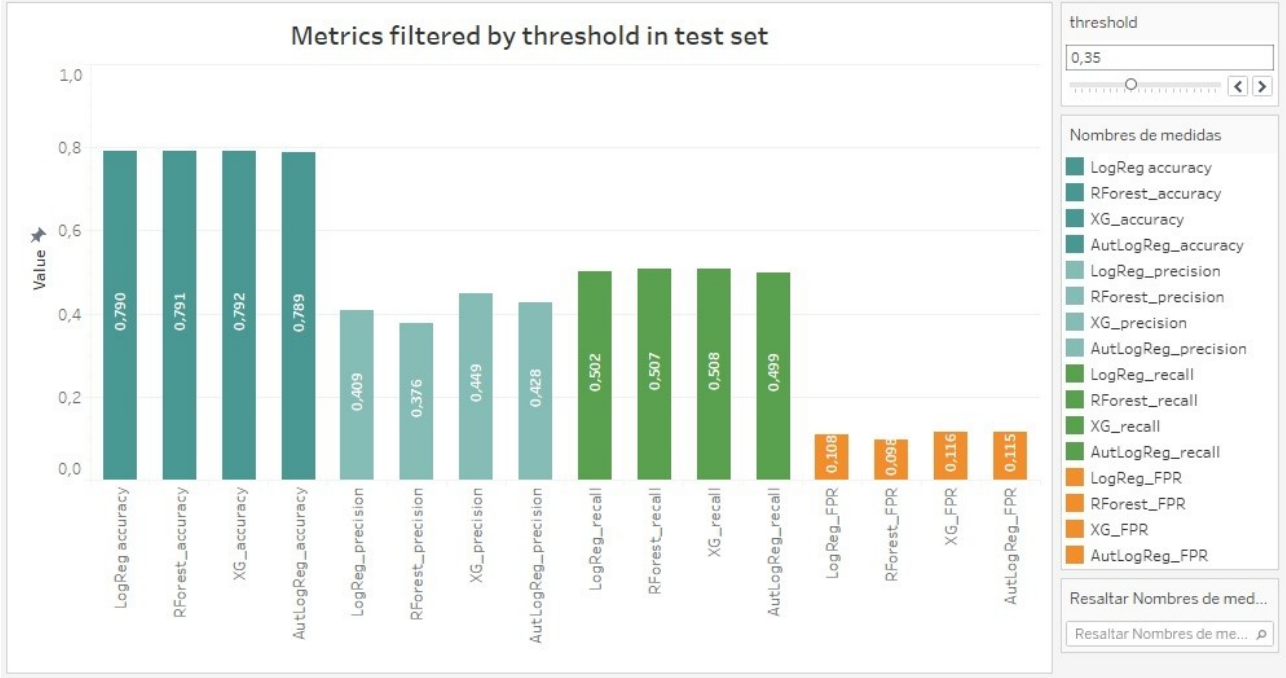
where "*new_data*" is the data for prediction in json format.

Files

- *doc/API_requests.ipynb*
- *src/server.py*

4 – Results

The visualization of the results has been done through a dashboard with tableau. The dashboard shows how the main metrics of the models vary (accuracy, recall, precision and False Positive Rate) when the cut threshold of the scores is modified (moving the slider in the top right corner).



On the other hand, in the dashboard you can also see the ROC curves for the test set and the AUC of the training and test sets in each of the developed models. According to this graphics, XG Boost had better results in the whole dataset (although Neural Network autoenconder + logistic regression model has been the best performance in the sample data):

	AUC train	AUC test	Accuracy train	Accuracy test	Recall train	Recall test	Precision train	Precision test
Logistic Regression	0,7700	0,7713	0,8066	0,8071	0,6187	0,6225	0,2135	0,2154
Random Forest	0,7788	0,7682	0,8046	0,8026	0,7194	0,6939	0,1187	0,1132
XG Boost	0,8004	0,7845	0,8144	0,8087	0,6693	0,6308	0,2352	0,2219
Autoencoder+LogReg	0,7738	0,7748	0,8073	0,8074	0,6222	0,6231	0,2165	0,2172

As a final conclusion it can be said that the **loan default prediction system improves the system without automation** since it is able to better distinguish **Charged Off cases with an average score of 0.36 and Fully Paid cases with an average score of 0.17** (initially the value of **prior was 0.21**).