Kaggle Loan Default Competition Strategy

Dolly Ye
Kaggle team name: darwin
SID: 915511231

Final score: 0.59524

Code Running

1. In terminal, type “jupyter notebook.”
2. Put Dolly_kaggle.ipynb in any folder.
3. On top, find “Cell.”
4. Click “ Run All.”

Dependency:

pip install xgboost
pip install sklearn

1. Overview

Due to the fact that 90% of the losses are zero, I use a two-step process: XGboost for binary classification first, and then a XGB regression model. The classification model predicts whether the loan will default . Once I know that the loan has defaulted, my regression model predicts how much the loss will be. For the former, I binarize the loss.

The rationale for using a regression model is as follows:

All the loss values are between 0 and 100, so if I just binarize them and train them, the maximum loss I would predict would be 1, which is far from accurate. Since the second model predicts the amount of loss, it wouldn't have to be trained on those examples with 0 loss, thus I set aside a dataset from the training set where the losses are nonzero

2. Preprocessing

Due to missing columns in the training set, I relabel training set so that it aligns with the 769 features in the test set. I replace missing data with the median of its column.

First, for both training set and test set, I remove the duplicate features and features that have the same value in every row, so the remaining features come down to 675 columns.

For the regression model in step 2, I create a regression training set, which only contains rows in the training set where the loss is positive.

Thus,I can binarize the loss column in the training set for the binary classification model in step 1.

3. Feature Selection / Engineering

I hypothesize that pairs of highly correlated features (>0.995) contains time series information (eg, a loanee's behavior over time), which is indicative of eventual default status. Since most people did not default,features that have 0.995 correlation can be fixed monthly payment across different months. The 0.005 discrepancy could be explained by the fact that some people did not make the full payment and eventually default. Then, I use a correlation matrix to extract all pairs of features that have 0.995 correlation coefficient. There are 355 pairs of features with 0.995 correlation.

Since the difference f521 - f271 captures the same information as f521 and f271, I don't need to keep both f521 and f271. Instead, I create a new set of features, which are the differences of each highly correlated pair. To make my model simpler,I remove highly correlated features (>0.9), which become redundant because of the newly added features.

In the end, I am left with 587 features.

	f9	f11	...	f739-f702	f704-f703	f705-f703	f710-f709	f711-f710	f743-f738	f748-f739	f748-f740	f749-f740	f749-f741
	119.430000	8.0	...	-3.245799	2.235901	-1.847799	-4.005070	-2.952889	0.5160	-0.631101	2.131399	-0.745001	1.385200
	121.500000	8.0	...	-0.592600	1.435900	-0.606200	-0.636280	-0.291860	0.0000	-0.051400	0.255600	-0.045600	0.122400
	134.389999	8.0	...	-2.090499	1.360800	-1.212200	-2.867910	-2.304960	0.1541	-0.608900	1.437800	-0.771999	0.945400
	125.750000	5.0	...	-0.469700	0.099400	-0.093600	-0.460770	-0.371330	0.0758	-0.010800	0.370600	-0.012700	0.292600
	127.279999	6.0	...	-1.925300	1.176901	-0.976500	-1.726789	-1.281800	0.6159	-0.456301	1.348800	-0.577600	0.927401

4. Model Selection

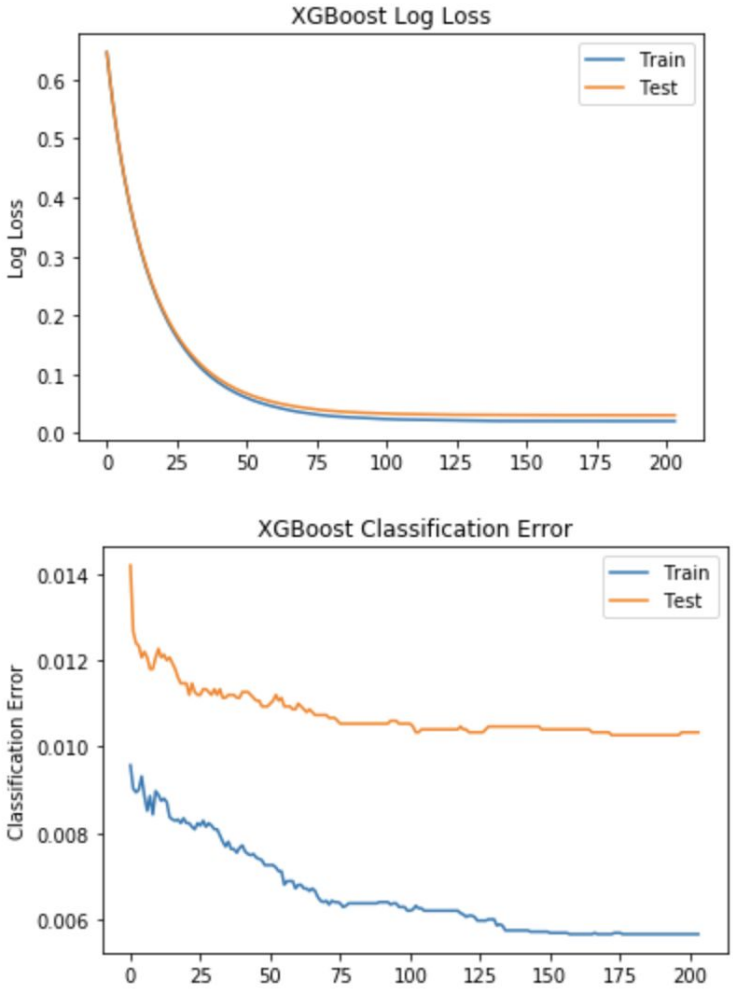
Classification

The classification step is done by xgboost. Boosting is an ensembling technique which uses sequential learning of classifiers. I tried both Bagging and Boosting. I found that if

the problem is that a single model gets a very low performance, Bagging will rarely get a better result. However, Boosting could generate a combined model with lower errors as it optimises the advantages and reduces pitfalls of the single model.

In Boosting, during each iteration, the learner will focus more on the wrongly predicted instances or give more weight to it. Thus it will try to predict the wrong instance correctly.

xgboost is a standalone python package.xgb.sklearn. Loss function is "binary:logistic."
I used grid search to find the following parameters: learning_rate=0.05, max_depth=20, gamma=10, n_estimators=500.
I tried AdaBoost(Adaptive Boosting),Stochastic Gradient Boosting, none of them performs as well as xgboost.



Regression

Similarly, I chose XGBRegressor after failing to get good results stacking Ridge, Lasso and Random Forest regressors. XGBRegressor is a bunch of weak learners in the form of regression trees. After each new weak learner is added into the mix, the regression forest re-weighs the data. So boosting can give a 'regression', but it is a *very* non-linear model,which works well for the data.