Stat 542 Third Project-Lending Club Loan Status

Team: RSS

1. General Idea

(1) Data cleaning steps

First, we defined a new variable called “pred” into the raw data frame and gave the observations “1” if the corresponding “loan_status” showed any of the following: 'Default’, ‘Charged Off’, ‘Late (31-120 days)','Late (16-30 days)','Does not meet the credit policy. Status: Charged Off', so that a “1” in column “pred” means Default loan, otherwise we filled the Good loans as “0”. By this way we convert loan status into a factor with 2 levels, lightening the burden of RAM capacity. After these steps, variable "loan_status" variable was removed from the original dataset.

Next, 8 obvious non-sense variables were removed, shown in the following list: "id","member_id","emp_title","url","desc","title","zip_code","policy_code". Note that “id” was not deleted for test data.

N.A. value proportions were then calculated for each variable. Among the name list, we found two interesting patterns:

1. For variable “collection_recovery_fee”, all the rows that have inputs in this column will match a “1” in “pred”.

2. Variable group ("open_acc_6m","open_il_6m","open_il_12m","open_il_24m","mths_since_rcnt_il","total_bal_il","il_util","open_rv_12m","open_rv_24m","max_bal_bc","all_util","inq_fi","total_cu_tl","inq_last_12m") had the similar pattern of value inputting, meaning that N.A. values tended to occur across all the same rows. Also, whenever there were non-N.A. values, the corresponding “pred” would be “0”.

We can use these two patterns to modify our result after modeling. As a result, we kept variable “collection_recovery_fee” and only one random variable ("inq_fi") from the variable group. Other variables with too much N.A. values were removed, which is the following list:

"collections_12_mths_ex_med","mths_since_last_major_derog","annual_inc_joint","dti_joint","acc_now_delinq","tot_coll_amt",.

Finally, we also see there are several characters appeared in the numeric variable such as “annual_inc” and “collection_recovery_fee”. This would change the variable from “numeric” into “factor” and thus may cause problem in the following process. To deal with such miswriting, we picked the variables that had more than 10000 levels, which indicated that it must be a numeric variable, and then transformed them into “numeric”.

(2) Model selection steps

We first realized the giant size of the dataset. It had 887379 rows inside the dataset. As a result, algorithm with complexity like O (n^2) or O (n*log(n)) might not be suitable within this case. Consequently we gave up svm and neural network at first cause they will take days to run even with an AWS instance.

Binary logistic classification model was on the list of our next trial. It is a simple model with complexity O (n). However, the Log loss will increases with the number of samples, thus leading to a large error in the overall prediction.

Finally, we went to the algorithms that can be both efficient and accurate. So we started to look at some fast implementation of traditional classification method. Package “ranger” is specifically designed for high-dimension large sample random forest. It’s efficient enough and can perform as well as the traditional random forest, thus became our final choice.

2. Model Performance

(1). Run time

Time(second)	Run in AWS m4.x4large	Run in 2.4Ghz RAM:8GB
Data Cleaning	220.673	531.251
Modeling	283.199	5798.364
	279.112	
	279.287	
	278.662	
	280.501	

Modeling	277.486	6124.385
	279.062	
	278.559	
	277.964	
	279.334	
Estimate time	499.993	6492.625

(2).Accuracy estimate

Number	Log Loss
1	0.02851615
2	0.02895631
3	0.02869477
4	0.02967941
5	0.02942500
6	0.02939668
7	0.02795250
8	0.02789681
9	0.02878660
10	0.02973433
Average	0.02890386

3. Conclusion

1. Since the number of rows is pretty large, algorithm choosing is extremely important in this project. In order to reduce running time(and money spent on AWS instances), time-consuming method like neural network and svm can’t be applied even though they may have a better performance.

(In our project process, we actually tried svm and neural network with small sample. Neural network with size>10 and iteration>200 may have slightly better accuracy than ranger function when n=100000. However, according to our estimate, running neural network on the whole dataset 5 times may take more than 30 hours, which means hundreds of dollar on AWS budget)

2. Logistic binary classification is an efficient way to classify without much need of RAM or CPU. If facility is restricted (if AWS is not allowed), logistic model might be a pretty good choice.

3. Parallel computing is a good way to save running time and make full use of CPU cores. Some functions like “Parallelsvm”,”multicore”,”detectcore()” might be useful.