CASE STUDY: Prediction of loan application payback

1. Abstract

Loan application repayment issues have been steadily increasing in recent years along with the increase in loan applications. Hence, the primary goal of loan application payback detection is to precisely score every loan application on a scale of 0 to 100, with 100 representing the loan application with the highest chance of default at onboarding. There are several current models that are used to detect loan default, including XGboot and lightGBM. The objectives of this study are to predict loan application payback on 48,000 unclassified loans before onboarding, and to compare both machine learning on the scorecard results.

2. Loan Application dataset

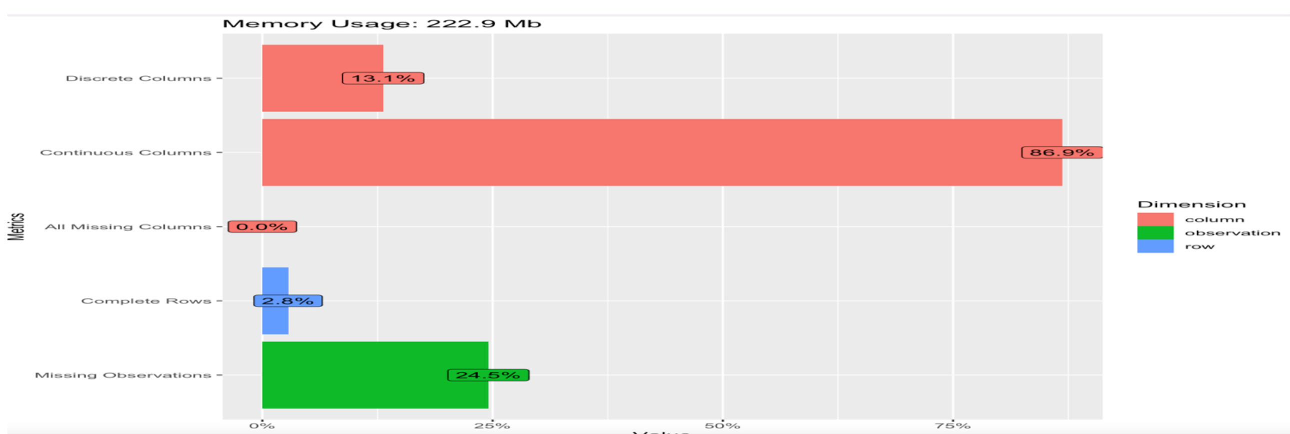
The data set contains more than 300,000 (276,000 didn't default and 24,000 default) classified loans received and 48,000 unclassified loan applications. The objective is to determine whether each applicant for an unclassified loan would experience payback issues.

Since only 8% of all classified loan received have payment difficulties in this dataset, particular care must be taken when using machine learning on such unbalanced datasets.

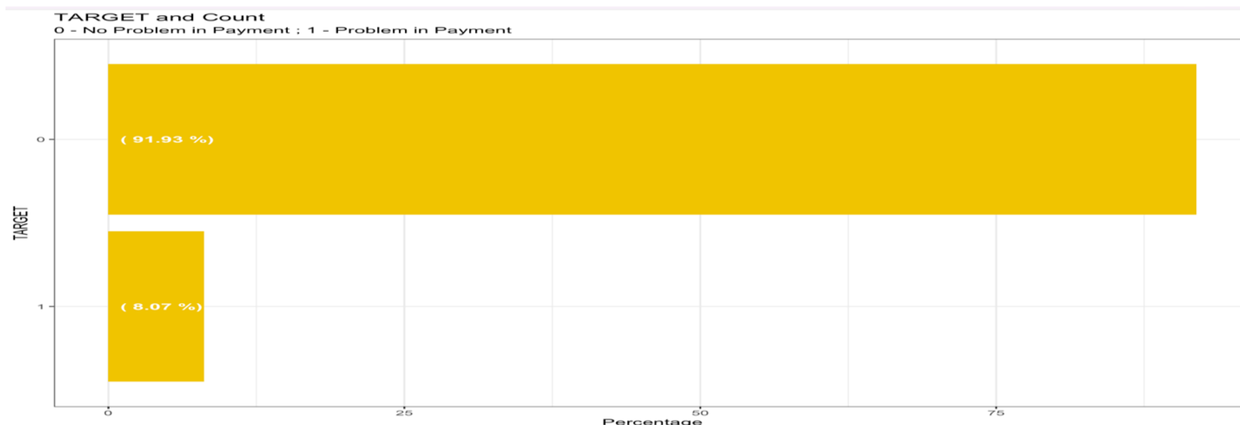
For this study, I will be using the weight column method instead of the over- sampling method (which poses the risk of overfitting our model due to the risk of having the identical samples in the training and test data), or the under-sampling method (where potentially important dataset from the samples can be lost). In the following sections, I will demonstrate how to assign weight, how to pre-train a classification model using weight column, and how to evaluate the performance of a model on unbalanced data.

3. Exploring the data

The table below shows that most of the dataset's observations have continuous values.
13 percent of all data are undesired, which may be the outcome of data transformation (encoding).
Only 2.8% of the rows in our observation include all the necessary data.
I have 24.5% of our total observations missing, which is manageable and rather decent.



The graph below shows that there is a problem with class imbalance. Out of the 300,000 classified loans received, 276,000 (91.93%) didn't default and 24,000 (8.07%) did default

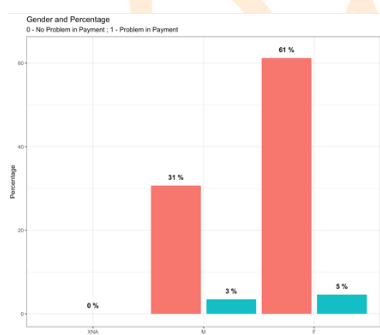


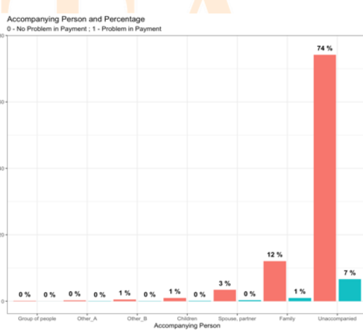
Loans that were paid back on time outnumber those that were not by a large margin. Assigning weights to the class during model training will therefore help to address the issue of data imbalance.

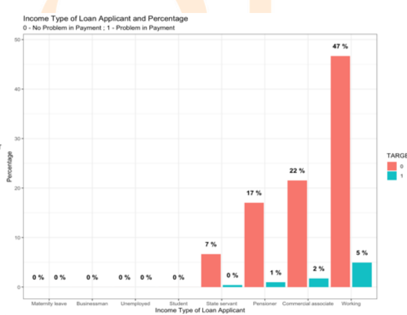
Weight Column, if applicable, is a column that shows the observation weight and that column must be a number higher than or equal to 0. Higher weighted rows are more significant. The weight has an impact on both model scoring and model training using weighted metrics. When producing test set predictions, the weight column is not utilized (but scoring of the test set predictions can use the weight).

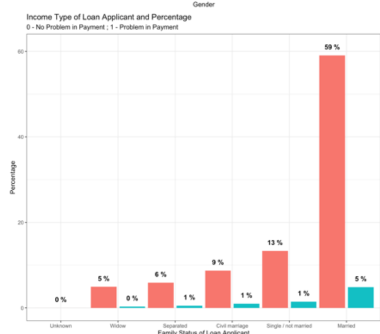
Without increasing the amount of data, the weight column gives the minority class more weight. It allocates higher loss for a certain row if the forecast were incorrect for positive, and it is preferable to over and under sampling. To obtain the best weight, we must take the square root of the positive number (for instance, if the ratio is 30 (0 class) to 1 (1 class) (a/b), then 5 to 1), that is 5 weight for each 1 class and 1 weight for each 0 class).

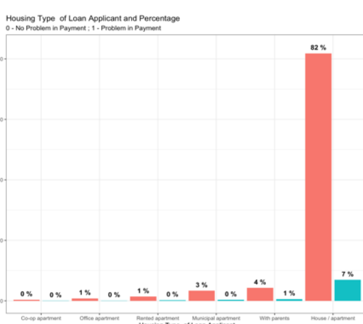
Below are some graphs comparing the target variable for the classified loans that were received with other variables

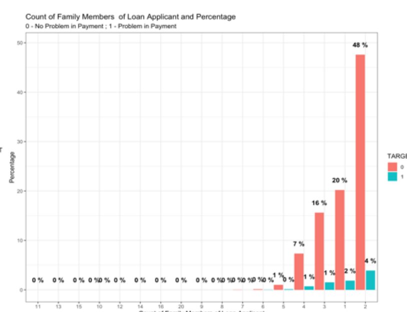










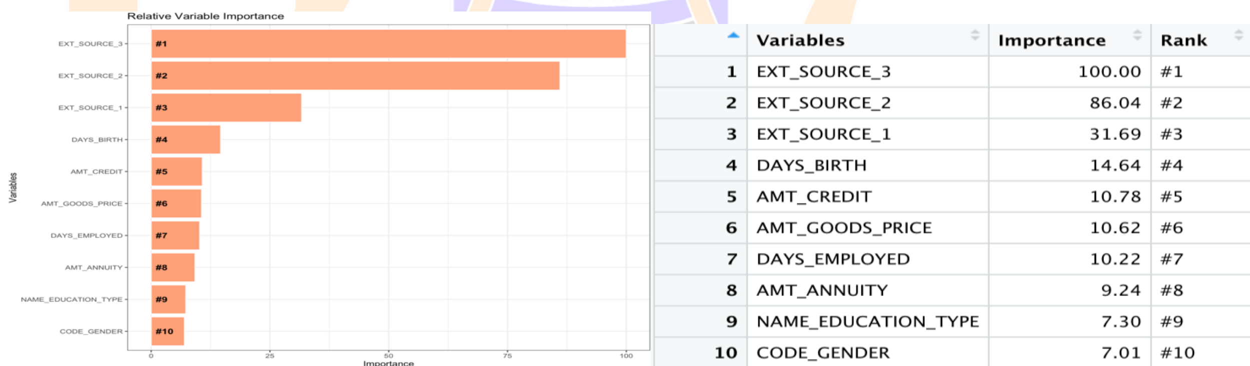


4. Modeling using XGBoost

Before modeling, I am converting all variables to numbers using the function below

```
Y2 = Y

features <- colnames(Y2)

for (f in features) {

 if ((class(Y2[[f]])=="factor") || (class(Y2[[f]])=="character")) {

  levels <- unique(Y2[[f]])

  Y2[[f]] <- as.numeric(factor(Y2[[f]], levels=levels))

 }

}
```

I can now test the XGBModel by fitting the model with the weight column and comparing the model's performance on extremely unbalanced data with the ROC (rock under the curve).
Since the majority class' classifications accounts for a high percentage of correct classifications, performance metrics like accuracy and area under the curve (AUC) cannot be used because they would produce results that are too optimistic.
The precision-recall curve or the sensitivity (recall)-specificity curve are two alternatives to AUC. I am using the ROCR (____) software to compute and visualize these metrics.



	Variables	Importance	Rank
1	EXT_SOURCE_3	100.00	#1
2	EXT_SOURCE_2	86.04	#2
3	EXT_SOURCE_1	31.69	#3
4	DAYS_BIRTH	14.64	#4
5	AMT_CREDIT	10.78	#5
6	AMT_GOODS_PRICE	10.62	#6
7	DAYS_EMPLOYED	10.22	#7
8	AMT_ANNUITY	9.24	#8
9	NAME_EDUCATION_TYPE	7.30	#9
10	CODE_GENDER	7.01	#10

The graph above ranks each variable according to importance.

Using the XBGModel, I can now forecast the scores for all 48,000 unclassified loan applications.

Bottom 10 Predicted		Top 10 Predicted	
SK_ID_CURR	TARGET	SK_ID_CURR	TARGET
313325	1.29%	164766	68.53%
146458	1.28%	365859	62.32%
443597	1.27%	165478	58.49%
228501	1.25%	327828	57.91%
172358	1.24%	419158	57.89%
102655	1.24%	195050	57.05%
118546	1.24%	429356	56.47%
293971	1.24%	382603	56.46%
356497	1.17%	119362	55.68%
335303	1.03%	266468	55.51%

5. Modeling using Lightgbm

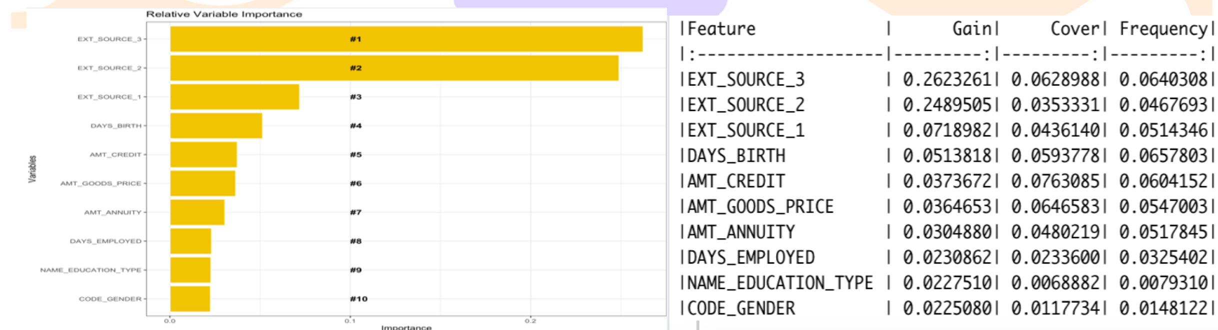
I perform a preprocessing on the full dataset by looking for mistakes, missing data, and N/A. After that, training and validation sets will be created from the train dataset. All new applications will be predicted using the 48,000 unclassified loan applications.

I can now test the LGBModel by fitting the model with the balance class, either by undersampling the majority classes or oversampling the minority classes. The projected probabilities of the first model will differ from a second model because the final model will additionally correct the final probabilities ("undo the sampling") using a monotonic transform. AUC won't be impacted, though, because it just worries about ordering.

```
lgb.model <- lgb.train(
  params = params.lgb
  , data = lgb.train
  , valids = list(val = lgb.valid)
  , learning_rate = 0.05
  , num_leaves = 7
  , num_threads = 2
  , nrounds = 3000
  , early_stopping_rounds = 200
  , eval_freq = 50
)
```

```
[LightGBM] [Info] Number of positive: 22294, number of negative: 254461
[LightGBM] [Warning] Auto-choosing row-wise multi-threading, the overhead of testing was 0.533095 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 11309
[LightGBM] [Info] Number of data points in the train set: 276755, number of used features: 120
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.080555 -> initscore=-2.434830
[LightGBM] [Info] Start training from score -2.434830
[1] "[1]:  val's auc:0.668183"
[1] "[51]:  val's auc:0.725628"
[1] "[101]:  val's auc:0.736699"
[1] "[151]:  val's auc:0.742364"
[1] "[201]:  val's auc:0.745336"
[1] "[251]:  val's auc:0.747091"
[1] "[301]:  val's auc:0.748381"
[1] "[351]:  val's auc:0.749462"
[1] "[401]:  val's auc:0.750327"
[1] "[451]:  val's auc:0.750793"
[1] "[501]:  val's auc:0.751209"
[1] "[551]:  val's auc:0.751395"
[1] "[601]:  val's auc:0.751659"
[1] "[651]:  val's auc:0.752005"
[1] "[701]:  val's auc:0.752302"
[1] "[751]:  val's auc:0.75246"
```

From the above table, I fit the model, evaluate the model on validation data, and calculate AUC



Feature	Gain	Cover	Frequency
EXT_SOURCE_3	0.2623261	0.0628988	0.0640308
EXT_SOURCE_2	0.2489505	0.0353331	0.0467693
EXT_SOURCE_1	0.0718982	0.0436140	0.0514346
DAYS_BIRTH	0.0513818	0.0593778	0.0657803
AMT_CREDIT	0.0373672	0.0763085	0.0604152
AMT_GOODS_PRICE	0.0364653	0.0646583	0.0547003
AMT_ANNUITY	0.0304880	0.0480219	0.0517845
DAYS_EMPLOYED	0.0230862	0.0233600	0.0325402
NAME_EDUCATION_TYPE	0.0227510	0.0068882	0.0079310
CODE_GENDER	0.0225080	0.0117734	0.0148122

The graph above ranks each variable according to importance.

Using the LGBModel, I can now forecast the scores for all 48,000 unclassified loan applications.

Top 10 Prediction		Lowest 10 Prediction	
SK_ID_CURR	TARGET LGBM	SK_ID_CURR	TARGET LGBM
121772	65.25%	249358	0.25%
149536	65.49%	443597	0.28%
313656	66.14%	279109	0.32%
265895	66.16%	208139	0.36%
119362	66.79%	430300	0.37%
396995	69.69%	174829	0.38%
215910	72.48%	298131	0.39%
250236	73.65%	329531	0.39%
195050	76.62%	170827	0.39%
164766	77.72%	224444	0.40%

6. Conclusion

In conclusion, both models perform well. XGBoost has better accuracy then LightGBM because we didn't lose any data in our train dataset.

Bottom 10 Predicted interval			Top 10 Predicted interval		
SK_ID_CURR	Min	Max	SK_ID_CURR	Min	Max
249358	0.25%	2.18%	431238	48.89%	56.60%
443597	0.28%	1.27%	379485	49.14%	57.85%
279109	0.32%	1.31%	353419	49.51%	58.38%
208139	0.36%	1.87%	141945	49.78%	51.39%
430300	0.37%	1.50%	417544	49.80%	57.75%
174829	0.38%	1.65%	429356	49.90%	56.47%
298131	0.39%	1.96%	265895	50.37%	66.16%
329531	0.39%	1.66%	288138	50.88%	55.43%
170827	0.39%	1.39%	294690	50.92%	52.36%
224444	0.40%	1.52%	250236	52.55%	73.65%
215958	0.40%	2.20%	438856	52.63%	58.62%
187516	0.41%	1.70%	423714	53.04%	55.74%
139208	0.41%	2.58%	141025	53.67%	62.75%
198997	0.41%	1.66%	266468	55.51%	56.12%
298142	0.42%	2.23%	119362	55.68%	66.79%
337310	0.43%	1.58%	382603	56.46%	62.06%
154529	0.43%	1.96%	195050	57.05%	76.62%
399953	0.43%	1.79%	327828	57.49%	57.91%
398910	0.43%	1.88%	365859	62.32%	62.87%
139755	0.43%	1.74%	164766	68.53%	77.72%