08/06/2021

Peer Lending Project
Stage 5 - Modeling & Evaluation
Group C

In this stage we continued the modeling process, we chose our final model, did some fine-tuning, and used feature selection to get the best results for the company.

Model Selection

At the end of the previous stage, we remained with two classification models - Random Forest and Gradient Boosting. We compared those two models based on the average realized return and the standard deviation of the realized return for different fractions of loans invested. We saw (appendix 1) that the models have similar results of average realized return - with a difference around ~0.05% in the lower fraction of loans and around 0% as the fraction increases. Looking at the standard deviation, we noticed that in the lower fraction, where the Random Forest obtained better average realized return, it had a higher standard deviation which abolishes its advantage. As the fraction increases, the standard deviation of the models becomes the same. Since the models' performance is nearly similar, we decided to choose the Gradient Boosting model, which has slightly better results.

Handling Outliers

Outliers may harm our model's performance (as for all machine learning models), therefore we wanted to try to handle the outliers we have in our data and see if it improves our model. We tried the more 'traditional' methods in step 3 - IQR[1] and gaussian[2] outliers. These methods have two problems, firstly, they determine if an observation is an outlier based only on one feature which can be misleading. Secondly, they have an underlying assumption that our features' distribution is normal - an assumption we couldn't confirm on our data and therefore did not feel comfortable using them to remove observations.

We tried another method - Isolation Forest. This algorithm recursively generates partitions on the dataset by randomly selecting a feature and a split value for the feature. The anomalies need fewer random partitions to be isolated compared to normal data points. Therefore, the anomalies will be the points that have a shorter path in the tree. The algorithm assigns an anomaly score for observations which is defined as:

[1] below Q1 − 1.5 IQR or above Q3 + 1.5 IQR

[2] 3 SD from the mean

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}}$$

- E(h(x)) - The average path length of observation x, the number of internal nodes the observation passes through before reaching a terminal node (appendix 2)
- c(n) - average path length of an unsuccessful search, meaning the sum of the depths of all the external nodes
- n - number of external nodes

A score close to 1 indicates anomalies and a score much smaller than 0.5 indicates normal observations.
We deployed the isolation forest on our training dataset without our target variables since we will not have them when deploying our model on new data. Using different thresholds for profiling an observation as an outlier and comparing (appendix 3) it to the model's performance with and without outliers removal, we saw that the average realized return hasn't increased. Therefore, we decided to not remove any outliers.

RFECV features selection

We used the RFE (recursive feature elimination) algorithm to check if we have any redundant features in our model and drop them, to decrease our models' training time and complexity while keeping (or improving) the model performance. RFE fits a model and removes the weakest feature based on the coefficients or feature importance rank our models produce. The process continues until removing a feature lowers our scoring function. The scoring function we chose for the feature selection process is **recall,** the ratio of correct positive predictions to the total positive examples. Defaulted loans classified as fully-paid generate more losses (average realized return of -12%) than fully paid loans classified as default prevents gain (average realized return of 6%), therefore we want to maximize recall and prevent false negative predictions. We used 5 cross-validations for every iteration and selected the best scoring collection of features.

To evaluate the impact of RFE on our model we compared the average realized return our models produce for every fraction of loan invested, with and without using RFE. Looking at the plot (appendix 4.1), we see that our model performance hasn't increased, however, we were able to obtain the same results with only 25 of the features, therefore we decided to proceed with these features and lower our model complexity while keeping the same results. (Appendix 4.2)

Hyperparameter tuning

To finetune our model's hyperparameters and attempt to produce better results (i.e. better average return for every fraction of loans invested), we started by using the method **grid search** which tests the model's performance on each unique combination of hyperparameters. However, using this method is too time-consuming (~15 minutes for one set of hyperparameters using 5 Cross validations). Therefore, we looked for another solution and used the **Halving Grid Search** (HGS) which uses a different approach called *successive halving* (appendix 5.1.). HGS is like a competition among all candidates (hyperparameter combinations). In the first iteration, HGS trains all candidates on a small proportion of the training data. In the next iteration, only the candidates which performed best are chosen and they will be given more resources (training samples) to 'compete'. With each passing iteration, the 'surviving' candidates will be given more and more resources (training samples) until the best set of hyperparameters is left standing. Using this process, the models' performance increased slightly in lower fraction of investment (appendix 5.2)

Summary

We finished the technical steps with the best model we could've built. The model is the Gradient Boosting model, which works with 25 features and 194147 rows. In the next stage, we will show the combination between the technical side and the business side as well as answer Walter's questions.

Appendices

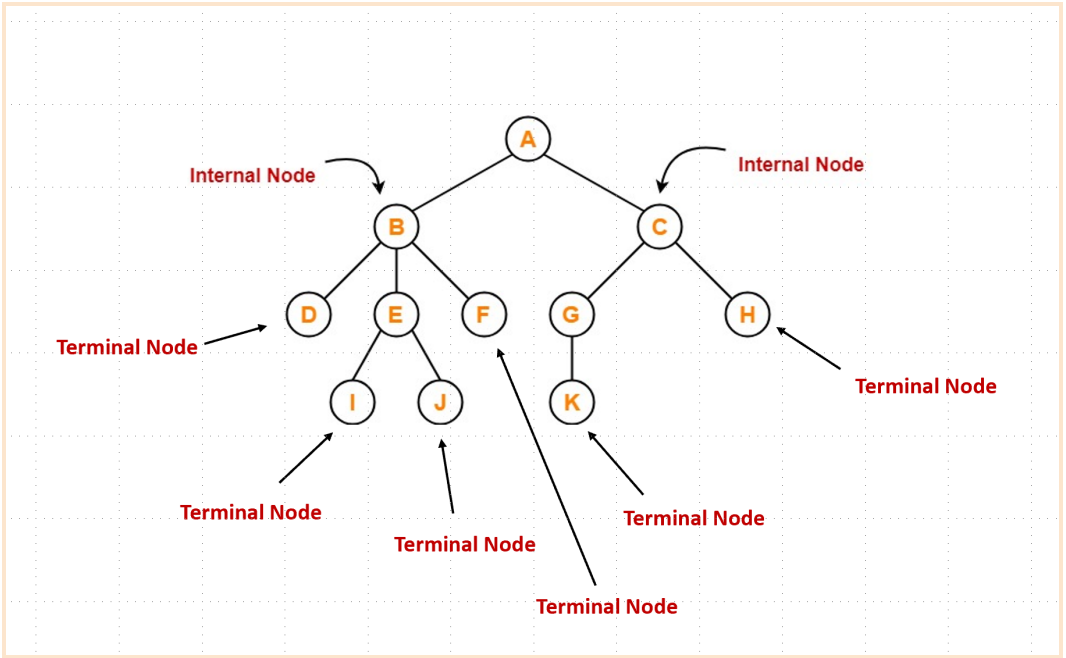
Appendix 1 - Differences between the models performances :

Gradient boosting - Random forest

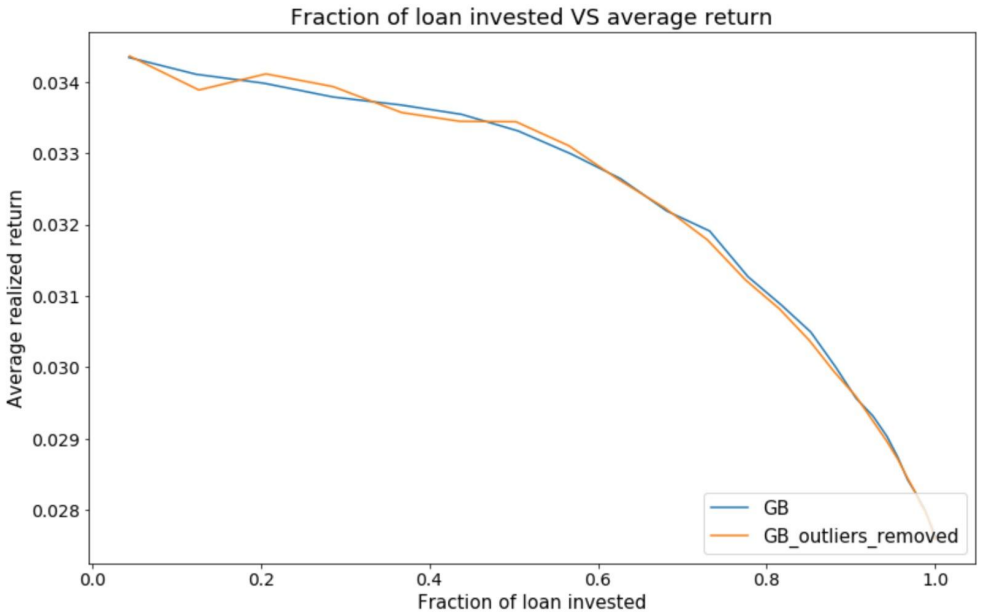
	Average realized return	Standard deviation of realized return	Investment fraction
0	-0.0492%	-0.4649%	4.4412%
1	-0.0802%	-0.1074%	12.3108%
2	-0.0536%	-0.0281%	20.2269%
3	-0.0494%	0.0794%	28.8794%
4	-0.0617%	0.1232%	36.9838%
5	-0.0287%	0.0886%	44.5116%
6	-0.0106%	0.0285%	51.2342%
7	-0.0067%	0.0361%	57.4092%
8	0.0025%	0.0203%	63.2980%
9	0.0092%	-0.0023%	68.6369%
10	-0.0188%	0.0343%	73.5704%
11	-0.0184%	0.0304%	77.8632%
12	0.0058%	-0.0008%	81.8198%
13	0.0148%	-0.0202%	85.2835%
14	-0.0028%	-0.0256%	88.2096%
15	-0.0008%	-0.0379%	90.5966%
16	0.0130%	-0.0657%	92.5542%
17	0.0181%	-0.0610%	94.3947%
18	0.0295%	-0.0655%	95.7536%
19	0.0226%	-0.0584%	96.8148%
20	0.0241%	-0.0478%	97.7464%
21	0.0221%	-0.0468%	98.4160%
22	0.0199%	-0.0359%	98.9373%
23	0.0169%	-0.0309%	99.2809%

24	0.0143%	-0.0228%	99.5207%
25	0.0083%	-0.0131%	99.7106%
26	0.0066%	-0.0105%	99.8014%
27	0.0023%	-0.0033%	99.9099%
28	0.0030%	-0.0042%	99.9500%
29	0.0013%	-0.0013%	99.9640%
30	0.0003%	-0.0006%	99.9922%
31	-0.0002%	-0.0006%	99.9940%
32	0.0001%	-0.0005%	99.9962%
33	0.0003%	-0.0003%	99.9975%
34	0.0000%	0.0000%	100.0000%

Appendix 2 - Tree

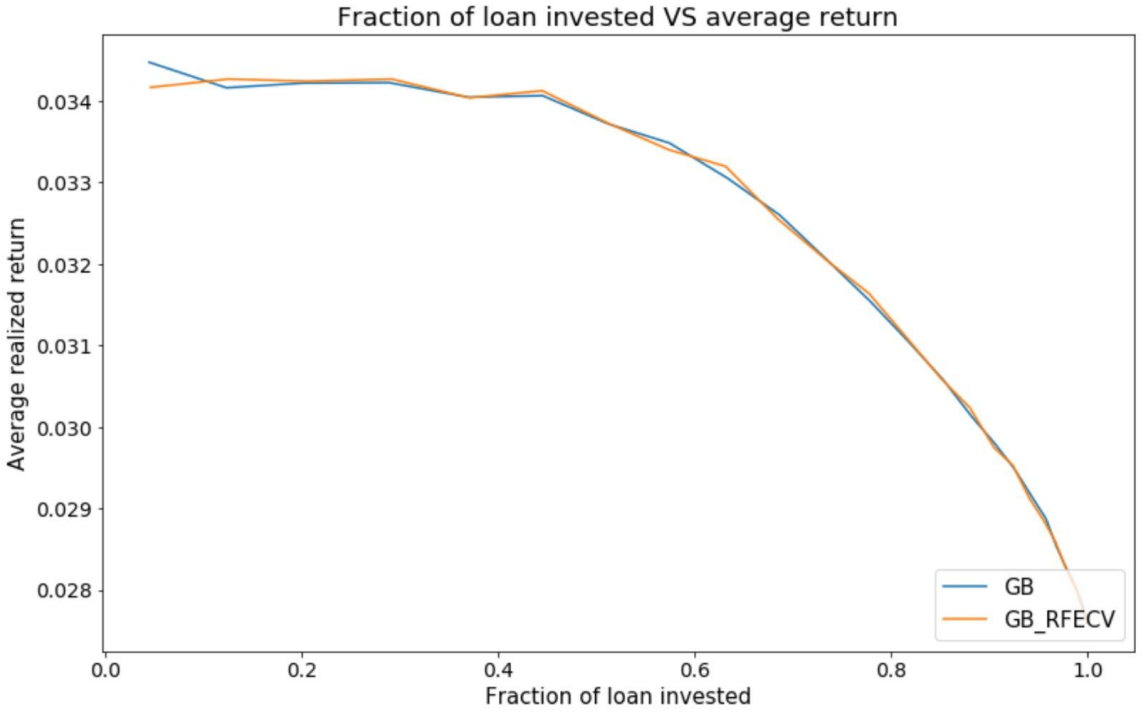


Appendix 3 - model performance with and without outliers



Appendix 4 - Feature Selection

4.1 - Model performance before and after the feature selection process
Blue - Before dropping features, Orange - after



4.2 - The names of features we kept after this process

1	loan_amnt
2	int_rate
3	installment
4	annual_inc
5	verification_status
6	dti
7	delinq_2yrs
8	fico_range_low
9	open_acc
10	revol_bal
11	total_bal_il
12	il_util
13	max_bal_bc
14	bc_open_to_buy
15	mo_sin_old_rev_tl_op
16	mort_acc
17	mths_since_recent_bc
18	num_act_bc_tl
19	num_il_tl
20	pct_tl_nvr_dlq
21	pub_rec_bankruptcies
22	cr_hist
23	Installment_feat
24	MORTGAGE
25	RENT

The names of features we dropped after this process

1	emp_length
2	purpose
3	pub_rec
4	revol_util
5	initial_list_status
6	tot_coll_amt
7	tot_cur_bal
8	all_util
9	num_bc_tl
10	OWN

Appendix 5 - Hyperparameter tuning

Appendix 5.1 - The hyperparameters we tuned

1. min_samples_split : Defines the minimum number of observations that are required in a node to be considered for splitting. Used to control overfitting and underfitting. Higher values prevent a model from learning relations that might be highly specific to the particular sample selected for a tree where Too high values can lead to not discovering important relations
2. min_samples_leaf : Defines the minimum or observations required in a terminal node. Used to control over-fitting similar to min_samples_split. Generally, lower values should be chosen for imbalanced class problems because the regions in which the minority class will be in majority will be very small.
3. max_depth : The maximum depth of a tree, Used to control over-fitting as higher depth will allow the model to learn relations very specific to a particular sample.

Appendix 5.2 - Model performance before and after the hyperparameters tuning process

Blue - Before tuning, Orange -