GREAT YIELDS
a boutique investment house

Roy Madpis - 319091526
Rawaa Makhoul - 316114370
Alexandra Fatyeyeva - 336540950

Peer to peer lending project

Working Paper № 4 – Team A

Introduction

In the previous stages we filtered loans with irrelevant instances (such as – loans with a status different from *Fully Paid* or *Charged Off* and loans that were given for 60 months), we identified our target variable by creating a new categorical column "realized_return_2%", formulated our modeling approach and created a python script for building that model. In this stage we finalized our modeling approach, i.e. selected the best features, chose the type of model (XGBoost) and started tuning the model's hyperparameters. Moreover we developed and improved our python script. In the following paper we will present the work we have done, reasons for different decisions we made and present the way we fulfilled them.

Updates:

1. We replaced the labels of the target variable, so now 0 = "realized return **higher** than 2%" and 1 = "realized return **not** higher than 2%". Therefore, the main metrics to evaluate our model's performance (and optimize its parameters) will be: Accuracy, NPV (Negative Predictive Value), and realized return SP (see *Note 2* in the appendix of *Working Paper #3. Team A* for elaborate explanation of the last metric).
2. In the previous stage we accidentally kept a variable that we wanted to filter: *total_pymnt_inv,* thus got erroneous results. We fixed this problem and will present the new and up-to-date results of the model we built. (see in the appendix ***Final - up to date - results***).
3. We measured several metrics to evaluate our model's performance. However, only three of them are major metrics that we will use to finalize the model selection: Accuracy, Negative predictive value, and Weighted average of realized return gained from the suggested portfolio.

Finalize the model selection:

1. Trying additional modeling approaches

At the beginning of this stage, we tried additional modeling approaches (see in the Appendix *Additional modeling approaches we tried*). All those attempts didn't yield a much better model, meaning we did not succeed in increasing the accuracy, thus we decided to keep our original model approach: using a classification model to predict whether the value of the column "realized_return_2%" is 0 (realized return higher than 2%) or 1 (realized return lower than 2%).

2. Feature Engineering

We created a synthetic feature, see in the appendix – "feature engineering"

In the following stage, we will continue tryting to create new synthetic features and observe if they can increase our model's performance.

3. Feature Selection

We performed feature selection by using a built-in python function[1] and also by trying to insert / filter features manually. Due to high computation time, we couldn't complete a full feature selection process and needed to perform it in separate steps. Unfortunately, the feature selection didn't yield a much better-performing model (it didn't increase the accuracy). *Final list of features we removed* in the appendix, represents a list of the final features that we are going to filter out.

Note: The list may be altered in the following stage.

4. Final Selection of model's type + Tune Hyper Parameters

- **Final selection of model:** As presented in *Working Paper #3. Team A*, we tried building three main classification models: Random Forest, Logistic Regression, and XGBoost. The XGBoost model outperformed the other models in multiple metrics (see in the appendix *Final up to date results - Comparing the 3 models*) and thus was selected as our "final model".

- **Tune Hyperparameters:** XGBoost has many **hyperparameters** that we can try to optimize to get better performance. The main obstacle we faced was computation time, so rather than using a grid search (trying all possible combinations of hyperparameters) we used Halving GridSearchCV. Therefore, we can't guarantee we got the best possible XGBoost model. We wish to continue to try optimizing the model in the next phase.

- **Threshold = 0.75:**
  Regarding the threshold to classify loans as "realized return higher than 2%": by default, the model uses a threshold of 0.5. In order to choose the "best" threshold we built the function "profit_curve"[2]. In short, The function creates both a profit curve (that enables visualizing the effect of each threshold on the weighted average realized return) and a **"threshold table"** that presents, for each threshold, the weighted average realized return of the suggested portfolio as well as the number of loans, the total investment in thousands of dollars, the number of FN and TN yielded (see in the appendix - Final up to date results - for the final threshold table and profit curve we got for the XGBoost model).

  The **threshold table** is crucial since we can use it to decide the "**best**" threshold; The **best** threshold will be chosen according to the specific needs of our client. We can see that if we increase the threshold, we can get a higher weighted average realized return, however, we have fewer loans to invest in. As The company didn't specify a specific limit of loans/budget, and only asked to advise if it is worth investing in peer to peer lending (and get returns higher than 2%), we decided to use a threshold of 0.75 - thus get a weighted average that is higher than 2% and at the same time don't "lose" high quantity of loans.

5. Evaluate Our Model's Performance:

To finalize our model selection, we must evaluate its performance. To evaluate our model, we can relate to two main types of metrics:

1. Popular and well-known metrics such as: Accuracy, Precision, Recall, F1-score, NPV, AUC.
2. Unique Monetary Metric: Weighted average of the realized return of the Suggested Portfolio.

Regarding the well-known metrics, we have already presented different ways that we used to improve those (Feature engineering, feature selection, hyperparameter tuning) and have already presented the

[1] Function: SequentialFeatureSelector from mlxtend.feature_selection

[2] See in the appendix of *Working paper #3. Team A - Profit curve – illustration*

model_all function that enables measuring those metrics[3]. Nevertheless, we must make sure our results weren't influenced by a specific test set. Therefore, we created a cross-validation function that enables measuring the Minimum, Maximum, Mean, and Standard Deviation (of the K folds) of the model's performance (regarding the well-known metrics) and also the *baseline grades table* of the model (see in the next paragraph explanation of this *baseline grades table*). For an elaborate explanation of this function see in the Appendix - *Elaborate explanation of the Function*: ***cross_val_RAR.***

We haven't yet addressed ways to evaluate how well our model is performing regarding the monetary metric: Weighted average realized return of the Suggested Portfolio. In order to do so, we decided to use the given inner model - the grade column:

We took all the 265,316 loans that were left after the first cleaning in the 2nd stage (after filtering the loans with term higher than 3 years, filtering loans that were not fully paid or charged off, and a few more cleaning[4]), and grouped them by the grade column, then we calculated for each grade three things:

a. The weighted average realized return (the weights are the *funded_amnt_inv*).
b. The weighted Standard Deviation of the realized return (the weights are the *funded_amnt_inv*).
c. Number of loans.

See the table we got in the appendix: *Baseline grades table*.

We claim that if we are able to create a model that can generate a Suggested Portfolio that its "baseline grades table" is better than the original "baseline grades table"[5] and can generate a weighted average realized return that is higher than 2% - then we have a "good" model = a model that is better than a model which selects loans merely by the grades and fulfills our client request to generate annual returns higher than 2%.

Note: Creating a **better** "baseline grades table" means that for every grade we can get a higher weighted average realized return as well as a smaller weighted standard deviation - compared to the original "baseline grades table". In other words, creating a **better** "baseline grades table" = succeeded generating higher returns with lower risks. See in the appendix *Baseline grades table* the results we got with our XGBoost model.

As we succeeded in generating a model with a better "baseline grades table" we can claim that the modeling approach that we chose and presented in this working paper is valid.

Plans for the next step:

In the following step, we will continue optimizing our model, i.e. continue the process of feature selection, parameter optimization and feature engineering. Hopefully getting better performance (Higher Accuracy, Negative predictive value and weighted average of realized return gained from the Suggested portfolio). Moreover, we wish to submit a concise python script that builds the final optimized model ready for the last step - conclusion and evaluation.

[3] See in the attached python file the "model_all" function.
[4] The full information regarding those cleaning is presented in *Working Paper #2.Team A*.
[5] i.e. a model that can filter from the table of 265,316 loans the "unwanted loans" (loans with realized return lower than 2%).

Appendix

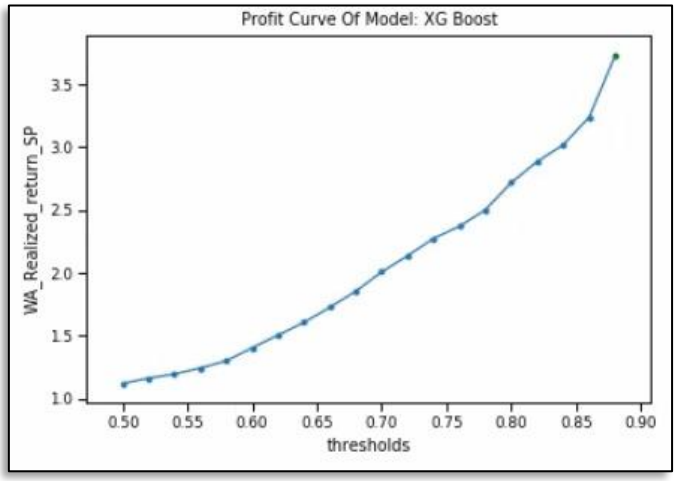
❖ Final up to date results

In this section, we present the final results of the **XGBoost model** that predicts the label of the categorical target variable: realized_return_2% (predicting whether a loan will have a realized return that is lower (True) or Higher (False) than 2%).

- **Comparing the 3 models**: (via function: **model_all**) Random Forest, Logistic Regression, and XGBoost so we can select the best performing model: (threshold = **0.5**)

	Random Forest	Logistic Regression	XG Boost
Accuracy	0.7134	0.7138	0.7153
Precision	0.5016	0.5101	0.5690
Recall	0.0357	0.0357	0.0273
F1 Score	0.0667	0.0667	0.0522
Negative_Predictive_Value	0.7178	0.7179	0.7173
AUC	0.6149	0.6194	0.6310
Num of FN	21086.0000	21087.0000	21269.0000
Num of TN	53642.0000	53669.0000	53965.0000
Realized Return SP	1.1324	1.1616	1.1189

- The **Profit Curve** of the XGBoost model got from the function: **profit_curve**



The matching **"threshold table"** of the profit curve: (presenting for each threshold the weighted average realized return of the suggested portfolio as well as the number of loans, the total investment in thousands of dollars, the number of FN and TN yield):

	thresholds	WA_Realized_return_SP	number of loans	Total investment in thousands of $	Number of FN	Number of TN
0	0.5000	1.1184	75234.0000	972215.3570	21270.0000	53964.0000
1	0.5200	1.1612	74614.0000	963287.1570	20946.0000	53668.0000
2	0.5400	1.1927	73712.0000	951033.9070	20529.0000	53183.0000
3	0.5600	1.2414	72481.0000	934561.1820	19969.0000	52512.0000
4	0.5800	1.2987	70795.0000	912825.7820	19263.0000	51532.0000
5	0.6000	1.4027	68326.0000	881089.8570	18207.0000	50119.0000
6	0.6200	1.5038	65174.0000	841899.1570	16954.0000	48220.0000
7	0.6400	1.6050	61129.0000	792916.6000	15437.0000	45692.0000
8	0.6600	1.7250	56112.0000	732939.0250	13647.0000	42465.0000
9	0.6800	1.8538	50319.0000	662557.6750	11663.0000	38656.0000
10	0.7000	2.0074	44122.0000	587217.8750	9713.0000	34409.0000
11	0.7200	2.1366	38105.0000	512540.3750	7966.0000	30139.0000
12	0.7400	2.2741	32169.0000	438182.4000	6339.0000	25830.0000
13	0.7600	2.3707	26273.0000	363322.0750	4925.0000	21348.0000
14	0.7800	2.5023	20026.0000	282817.3750	3503.0000	16523.0000
15	0.8000	2.7190	13616.0000	196735.6000	2122.0000	11494.0000
16	0.8200	2.8863	7669.0000	114667.1250	1102.0000	6567.0000
17	0.8400	3.0186	2701.0000	42918.7750	340.0000	2361.0000
18	0.8600	3.2382	379.0000	6845.0250	45.0000	334.0000
19	0.8800	3.7263	2.0000	46.0000	0.0000	2.0000

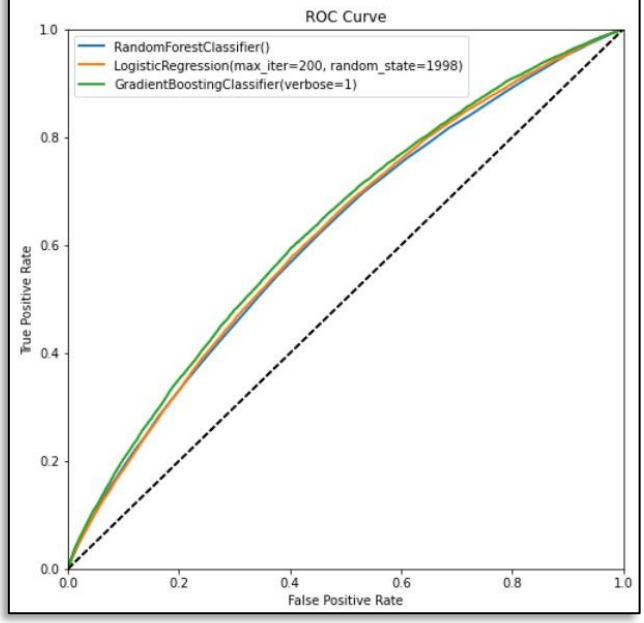
- The **Metrics table** got from the function: **model_all** for the best performing model – XGBoost (threshold = **0.75**)

	XG Boost
Accuracy	0.7153
Precision	0.5690
Recall	0.0273
F1 Score	0.0522
Negative_Predictive_Value	0.7173
AUC	0.6310
Num of FN	21269.0000
Num of TN	53965.0000
Realized Return SP	2.3182

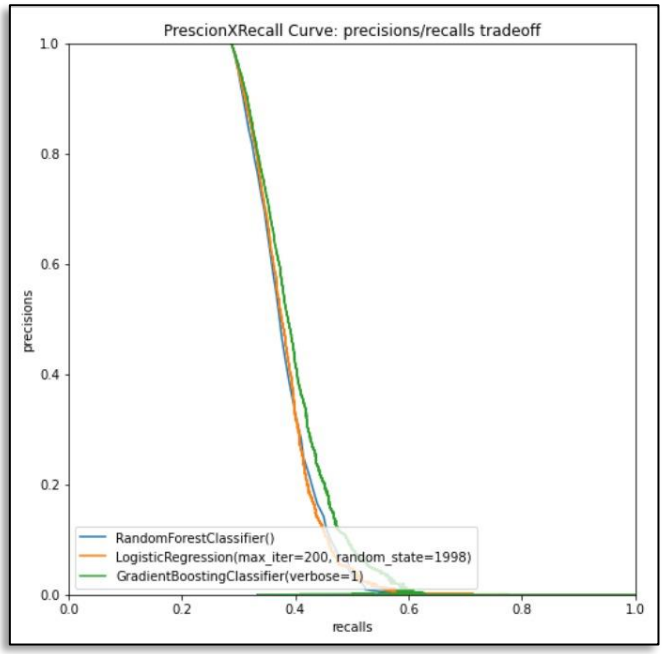
Note that we will mainly use the Accuracy, Negative Predictive Value and Realized Return SP metrices to evaluate and optimize our model. (see in "updates" on the first page point 3)

- The **ROC Curve** got from the function: **roc_curves_RAR**

	AUC Value
RandomForestClassifier()	0.6149
LogisticRegression(max_iter=200, random_state=1998)	0.6194
GradientBoostingClassifier(verbose=1)	0.6310



- The **Precision VS Recall Curve** got from the function: **Precision_VS_Recall_RAR**



❖ Feature engineering

In this stage we created a new feature:

- "Percent of revolving accounts": We decided to create a synthetic feature to get the percentage of the revolving accounts out of all open accounts:

$$\text{Percent of revolving accounts} = \frac{\text{column 100 (num\_op\_rev\_tl – number of open revolving accounts)}}{\text{column 33 (open\_acc – total open accounts)}} \times 100\%$$

❖ Baseline grades table:

The baseline grade table with **all the loans**

Base line Grades - all the loans

	weighted_average	weighted SD	Number of Loans
A	1.861573	4.514966	56753
B	1.532950	7.149496	92005
C	0.798874	9.438672	74610
D	-0.194200	11.545009	30543
E	-1.022335	13.340783	8798
F	-2.109506	14.559178	2114
G	-3.628924	15.255247	493
Total All loans Together	1.076391	8.414820	265316

The baseline grade table - of our XGBoost model with a threshold of 0.75

Base line Grades - Suggested portfolio with threshold of: 0.75 XGBOOST

	weighted_average	weighted SD	Number of Loans
A	2.182472	4.279334	9355
B	2.288773	6.436471	13806
C	2.502109	8.261008	5705
D	4.214223	9.056423	397
E	1.171057	12.854635	26
F	9.791605	0.042519	2
Total All Suggested Portfolio	2.318193	6.273655	29291

Note how for each and every grade our XGBoost model, with a threshold of 0.75, is able to generate **higher** weighted average realized return and **lower** weighted standard deviation of realized return. Meaning it succeeds in generating higher returns with lower risk.

❖ Additional modeling approaches we tried

1. We tried to build a regression model that predicts the *total_pymnt_inv*, then insert the data with the prediction column into a classification model that predicts realized_return_2%;
2. We also tried building a classification model that predicts the *loan_status*, then filter all the loans that the model predicted as "Charged off" and inserting the filtered data into a classification model that predicts realized_return_2%.

As was mentioned at the beginning of the working paper, all those attempts didn't yield much better results.

❖ Elaborate explanation of the Function: cross_val_RAR

This function gets **8 arguments**:

2. **model** - A model **before** fitting**.** The function will fit the model to K-1 folds each time and test on the remaining fold.
3. **x_train** - the original x_train (with all the columns).

4. **x_train_for_models** - the original x_train without the columns: "id", "funded_amnt_inv", "realized_return", "grade", "sub_grade" (those columns are needed for calculating the actual realized return gained by the suggested portfolio).
5. **y_train** - y_train array.
6. **k** - Number of Folds for the cross-validation. By default **K=5**.
7. **threshold** - the threshold to use for filtering columns in the suggested portfolio - all the loans that their probability to be "0" = realized return higher than 2% - will be higher than the threshold - will be included in the suggested portfolio. by default the **threshold = 0.5**.
8. **shuffle** - whether to shuffle x_train_for_models before splitting into K folds. By default = **True.**
9. **random_state** - Int. If shuffle= True then you may want to generate replicable outcomes. By default = **None.**

<u>This function returns 5 values:</u>

1. **suggested_ports** = The first output of the function is a list of all the loans that the model suggests the company to invest in. The suggested portfolio will be influenced by:
   - The model that was used
   - The threshold

This list will have K values, each value is gained from a different test-fold. Meaning we will get a list with K's suggested portfolios.

2. **Metrics tables**: The function calculates the following metrics and returns a list with K elements, each contains the metric <u>table</u> of a different test-fold set:
   a. Accuracy
   b. Precision
   c. Recall
   d. F1 Score
   e. NPV (Negative Predictive Value)
   f. AUC = area under the ROC Curve
   g. Num of FN = The number of False Negative predictions
   h. Num of TN = The number of True Negative predictions
   i. Realized Return SP = The weighted average of the suggested portfolio. This is calculated by using the column *rezlied_return* to calculate the mean and using the *funded_amnt_inv* column as the weights. This is the return the company will get if it invests in **all the loans** in the **Suggested Portfolio**.
3. **Metrics_table_all_folds_statistics** = a table with **statistics** regarding the K metrics of the K folds: The statistics are: Minimum, Maximum, Mean and standard deviation of the: Accuracy, Precision, Recall, F1 Score, Negative Predictive Value, AUC, and Realized Return SP.
4. **baseline_grades_dfs** = This variable holds K <u>tables</u>. Each table presents, for the corresponding **fold** the:
   - **Weighted average of the realized return by grades** (weight = funded_amnt_inv)
   - **Standard deviation of the realized return by grades** (weight = funded_amnt_inv)
   - **Number of loans** in each grade

5. **base_line_all_folds** = This is a table summarizing the mean, Minimum, Maximum, and standard deviation value of the K folds regarding the "base_line_grades_dfs" (This variable shows statistics of the K tables of the 4th output of this function)

Example of using the function cross_val_RAR on the XGBoost model with threshold 0.75:

Metrics_table_all_folds_statistics:

Statistics of the model for 5 folds with threshold of 0.75:

	Min	Max	Mean	Sd
Accuracy	0.711818	0.720076	0.715201	0.003181
Precision	0.552734	0.595699	0.569667	0.019504
Recall	0.024347	0.028246	0.026505	0.001528
F1 Score	0.046655	0.053746	0.050647	0.002811
Negative_Predictive_Value	0.713545	0.722518	0.717172	0.003413
AUC	0.629115	0.635611	0.631345	0.002548
Realized Return SP	2.207190	2.378726	2.277079	0.086071

Note how for each and every metric the standard deviation is quite low, meaning the model performs quite the same for different test folds.

base_line_all_folds:

Statistics of Realized return by grade for 5 folds with threshold of 0.75:

	Weighted Average all folds				Standard Deviation all folds				Number of Loans all folds			
	mean	min	max	sd	mean	min	max	sd	mean	min	max	sd
A	2.165756	2.080576	2.218764	0.052197	4.295975	4.182398	4.431141	0.089288	4390.6	4336	4454	52.950921
B	2.278077	2.206056	2.358258	0.066701	6.503338	6.366264	6.631370	0.107049	6428.2	6326	6556	100.385258
C	2.417272	2.005182	2.728381	0.312456	8.309501	8.088307	8.718890	0.284566	2623.2	2535	2719	71.824091
D	3.104324	2.630672	3.441793	0.344190	9.594174	9.221508	10.151234	0.392070	181.2	148	227	33.372144
E	3.765097	-3.283825	9.182111	4.935600	7.690721	2.064529	14.083771	5.495881	9.8	5	16	3.962323
Total All Suggested Portfolio	2.277079	2.207190	2.378726	0.086071	6.352716	6.352716	6.352716	0.000000	13633.2	13457	13865	155.451600

❖ ***Final List of the removed columns:***

```
columns_not_needed = ["addr_state",
                      "member_id", "funded_amnt","mths_since_recent_inq",
                      "emp_title", "verification_status",
                      "pymnt_plan", "term", "desc","title", "zip_code",
                      "fico_range_low", "mths_since_last_delinq",
                       "mths_since_last_record","out_prncp","out_prncp_inv",
                       "total_pymnt", "total_rec_prncp","total_rec_int",
                       "total_rec_late_fee", "recoveries",
                       "collection_recovery_fee","last_pymnt_amnt", "next_pymnt_d",
                       "last_credit_pull_d", "last_fico_range_high",
                       "last_fico_range_low", "mths_since_last_major_derog",
                       "policy_code", "annual_inc_joint", "dti_joint",
                       "verification_status_joint", "tot_coll_amt",
                       "mths_since_rcnt_il","avg_cur_bal",
                       "mths_since_recent_bc_dlq",
                       "mths_since_recent_revol_delinq",
                       "num_tl_120dpd_2m", "num_tl_30dpd","revol_bal_joint",
                       "sec_app_fico_range_low", "sec_app_fico_range_high",
                       "sec_app_earliest_cr_line", "sec_app_inq_last_6mths",
                       "sec_app_mort_acc", "sec_app_open_acc",
                       "sec_app_revol_util", "sec_app_open_act_il",
                       "sec_app_num_rev_accts", "sec_app_chargeoff_within_12_mths",
                       "sec_app_collections_12_mths_ex_med",
                       "sec_app_mths_since_last_major_derog", "hardship_flag",
                       "hardship_type", "hardship_reason", "hardship_status",
                       "deferral_term", "hardship_amount", "hardship_start_date",
                       "hardship_end_date", "payment_plan_start_date",
                       "hardship_length", "hardship_dpd", "hardship_loan_status",
                       "orig_projected_additional_accrued_interest",
                       "hardship_payoff_balance_amount",
                       "hardship_last_payment_amount",
                       "debt_settlement_flag", "debt_settlement_flag_date",
                       "settlement_status", "settlement_date", "settlement_amount",
                       "settlement_percentage", "settlement_term","open_il_12m",
                       "last_pymnt_d", "earliest_cr_line", "application_type",
                       "Unnamed: 0",  "issue_d", "realized_return_3", "total_pymnt_inv",
                       "il_util", "bc_util","emp_length",  "open_rv_12m",
                      "open_act_il","total_cu_tl", "all_util", "acc_open_past_24mths",
                      "num_actv_rev_tl","num_rev_tl_bal_gt_0","pct_tl_nvr_dlq"]
```