GREAT YIELDS
a boutique investment house

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

Peer to peer lending project
Working Paper № 3 – Team A

Introduction

In the previous stage (stage **2**), we filtered out loans with irrelevant instances, such as loans with a status different from *Fully Paid* and *Charged Off* and loans that were given for 60 months. We saved the relevant loans, in all quarters combined into a CSV file for further and easier use in the steps ahead. Although we had filtered out 92 features in the previous stage, we have decided to start this phase by using all 150 columns (excluding "*URL",* col.19). Throughout this stage we investigated and tested different models, so we can decide our modeling approach and did more data preprocessing, making it more suitable for the model(s). For this stage, we will use a **classification model** to predict whether a loan will have a realized return higher or lower than 2%. 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.

Three important things which affect the modeling approach that we considered:

1. **The Target Variable:** Which variable should we use as the target? We decided to create a new column: "realized_return_2%" - this is a categorical column that was created by using an if-else statement on the "realized_return" column[1]. If the realized return is higher than 2%, then assign the value "1", else, assign "0". The reason for choosing this as our target variable is quite simple: we wish to provide the company with loans that their realized return is higher than 2%, therefore we need to classify loans as "expected realized return higher than 2%" or "expected realized return smaller than 2%". See in the appendix *Note 1* for more approaches we considered and our plan B (In case this target variable approach doesn't work).

2. **Selecting the predictors + Preprocessing:** Which variables should we use as the predictors? In stage 2 we filtered 92 columns, leaving us with 59 columns (See working paper 2). In this stage, we filtered more columns that are not needed and also tried to use columns that we removed in stage 2. In fact, we tried to retrieve features that had 40% or fewer NAs and observed how this affected our model's performance. In this notion, when retrieving columns that have missing values - we had to further preprocess those columns and decide the right way to deal with NAs[2].

3. **Selecting the type of the Model:** Which model to use? If the target variable is categorical, then we can use Decision Tree / Random Forest / XGBoost / Logistic Regression, etc. Otherwise, if the target variable is numeric (for example - the realized return column), then we need to use a regression model. As of now, our target variable is categorical, therefore we used classification models. We tried using several models and

[1] See working paper 2 for the calculation method of the realized return.

[2] We created 2 functions for dealing with NAs in the preprocess part in python(NA count: function that outputs the columns with NAs and the number of NAs in each; dealing with NAs: filter all the loans with NA values in specific columns and replace NA with the mean value for other specific columns), see the attached python file.

compared their performance. **Note:** for comparing the different models we used the following metrics: Accuracy, Precision, Recall, F1 Score, AUC, number of False Positive cases[3], number of True Positive cases[4], and weighted average of the realized return gained from the Suggested portfolio. (See ***Note 2*** in the appendix for an elaborate explanation of the last metric).

To deal with those three considerations together, we worked in an organized way, mainly by building a python script that, given a target column, it aims to find the predictors, preprocess them and select the best type of model (the best model can be the one that yields the highest precision and weighted average realized return in the suggested portfolio, see *Note 2*). We will now present shortly what we did in each step in our python code[5].

Description of the work done:

I. **Loading the data:** loading the cleaned table (after the cleaning of part 2 – taken from R)

II. **Pre-processing:**

- Creating new features - Feature engineering; (we wish to address this more in stage 4, in this stage we only created the column "realized_return_2%" - our new target variable).
- Selecting columns to use in the model: We used the function filter_columns. This function enables filtering columns that we don't want to include in our model and allows trying different combinations of features and observe how those affected the metrics for evaluating the model's performance (see in section 4 – scoring). We documented those attempts in a research log[6] .
- Dealing with NAs: We used the function na_count to compute the number of NAs in each column (and thus understand which columns we need to address in terms of dealing with NAs), then we used the function dealing_with_NAs to **deal** with those columns that have NA values. There are two ways to do that:

  1. **Removing rows with NA values** in the column
  2. **Replace NA values with another value** (for example – the mean/median/mode).

- Dealing with Outliers: After some data exploration, we found variables with outliers. In the next stage, we may use the IQR method to treat them.
- One Hot Key encoding: To use the categorical columns in our model, we need to convert them into dummy variables. We used the function dummy_variables which converts the categorical variables into dummy variables.
- Last Columns filtering: In this part, we filtered columns that weren't removed at the beginning (at "selecting columns to use in the model"), and finish all the pre-processing before splitting the data into train and test sets. The variable df_2019_clean_final_before_splitting is the final variable after all the cleaning, ready to be split into train and test sets.

[3] We are most concerned about type I error, known as FP errors = loans that our model suggested to invest in, however we, in fact, shouldn't invest in them – Those can be costly mistakes, so we wish to **minimize** the amount of FP our model generates.

[4] As we wish to minimize the FP cases, we also wish to maximize the TP cases.

[5] See the attached python code - *Stage 3 – Model. Team A*

[6] See the attached excel file "Research log part 3 - Team A"

- Train-Test Split[7]: We split the data into **training (30%)** and **testing (70%)** sets.
- Scaling - normalizing the data: We used the function scaling_train_test to scale the training and test sets. The function can perform different methods of scaling. We decided to use the method of Min-Max scaling: $x_i\ new = \frac{x_i - mean(X)}{max(X) - \min(X)}$

III. **Model selection:** Building a model(s): Since there is no way, at this stage, to directly determine which type of model to use and which additional columns to use, we decided to examine different possible models and features. We examined Random Forest, Logistic Regression, and XGBoost[8]. All the results are summarized in a Research Log[9]. In stage 4 we will finalize the selection of the model type.

IV. **Scoring:** We used the function model_all to gain several metrics to evaluate our model's performance: Accuracy, Precision, Recall, F1 Score, AUC, Number of FP, Number of TP, Realized return SP (The weighted average of the realized return of the Suggested Portfolio – see note 2). We wish to mainly use the Precision and the Realized return SP. The precision is an important metric since we want to minimize the number of FP and at the same time increase the number of TP. In addition, the Realized return SP (SP = Suggested Portfolio) is also an important metric as it reflects the expected return of the company (If it invests in all the loan in the suggested portfolio). Those metrics will help us decide which model performs the best.
It's important to mention that the amount of loans in the Suggested portfolio is influenced by the threshold of the classification – if we increase the threshold, we anticipate getting fewer loans, however, get a higher weighted average of realized return (as there are less FP). To find the best threshold, we built a function profit_curve (see in the appendix – profit curve illustration).
At the same time, we also compared the ROC Curves of the models and the PrecisionXRecall Curves, thus we built 2 more functions: roc_curve_RAR for generating the ROC Curves, and function Precision_VS_Recall_RAR for generating the PrecisionXRecall Curves (see in the appendix – ROC Curve and Precision X Recall Curve).

Plans for the next step:

In the following step, we will choose our final model – finalize the selection of a target variable and the modeling approach. Then we will perform **feature selection** on the columns that were chosen in this stage so we can find which variables can increase our model's performance (i.e. increase the weighted average of the realized return in the suggested portfolio set). Finally, we will use **parameter optimization** to find the best parameters for the type of the chosen model. Hopefully, we will get better results regarding the metrices: Precision and "weighted average of the realized return gained from the Suggested Portfolio".

[7] Needs to be done before scaling as we want to define a scaler and fit it according to the train set.

[8] We also tried using a KNN model and Naïve Bayes classifier. Regarding the KNN model - the computation time was too high (when predicting the labels for the test set), and regarding the Naïve Bayes – it performed poorly on the test set, so we decided not to use them.

[9] See the attached excel file "Research log part 3 - Team A".

Appendix

Note 1: More modeling approaches/plan B:

We also tried modeling the “realized return” and the “loan_status” column.

- Regarding the “realized return”: as it yielded a very small R square (0.016) we decided to leave this approach for now.
- Regarding the loan_status: The Accuracy of a model using this column as the target variable was quite high(~0.97), thus we may wish to use this column if our “Plan A” (Model the new column we created: “realized_return_2%”) doesn't work.

Plan B: If Plan “A” (Model the new column we created: “realized_return_2%”) doesn’t work, we can use two models together:

1. A **classification** model, where the target variable is “**loan_status**”. The model will classify loans to be “Fully Paid” or “Charged Off”.
2. A **regression** model, where the target variable is “**realized return**”. The model will predict the realized return of each loan

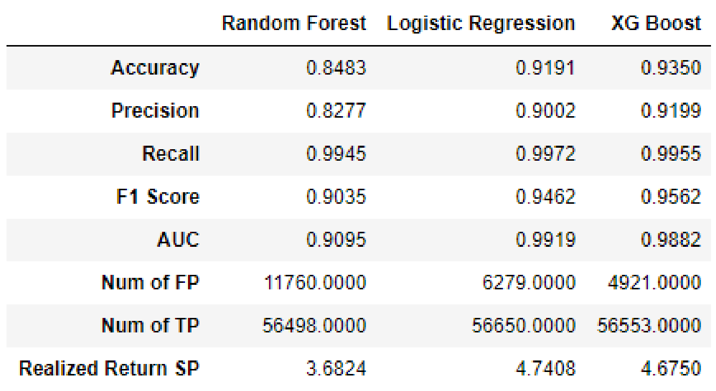
We will first use the classification model to classify loans to “Fully Paid” or “Charged Off”. Then we will insert all the loans that were **predicted** as “**Fully Paid**” into the regression model and predict their realized return. Finally, all the loans that their **predicted** realized return will be higher than 2%, will be included in the “suggested portfolio” (A list of all the loans that we will suggest the company to invest in).

Note 2: Explaining the metric “Weighted average of the realized return gained from the Suggested portfolio”:

After building a model, for example - random forest, we predicted the **labels** of all the loans in the test set (The labels will be either “Realized return higher than 2%” = 1 OR “Realized return not higher than 2%” = 0) and also predicted the **probability** of each loan to be classified as “Realized return higher than 2%” = 1. Then filtered all the loans with a probability that is lower than the **threshold**, by default the threshold is 0.5. All the remaining loans are defined as our **“suggested portfolio”**, meaning the loans we would have suggested the company to invest in. Finally, we performed a weighted average on the column “realized return” using the column “funded_amnt_inv” as the weights. This provided us with the weighted average of the realized return gained by a specific model with a specific threshold while considering the amount the investor invested in the loan.

Note: we created a function “model all” that calculates, for each model, for a given threshold, its Accuracy, Precision, Recall, F1 Score, AUC, number of FP, number of TP, and the weighted average of the realized return (Called in python: “Realized return SP“ [SP = suggested portfolio]) and the confusion matrix.

Example of combining 3 outputs of this function for 3 different models:



	Random Forest	Logistic Regression	XG Boost
Accuracy	0.8483	0.9191	0.9350
Precision	0.8277	0.9002	0.9199
Recall	0.9945	0.9972	0.9955
F1 Score	0.9035	0.9462	0.9562
AUC	0.9095	0.9919	0.9882
Num of FP	11760.0000	6279.0000	4921.0000
Num of TP	56498.0000	56650.0000	56553.0000
Realized Return SP	3.6824	4.7408	4.6750

Profit curve – illustration*:*

The function profit_curve gets **6 arguments**:

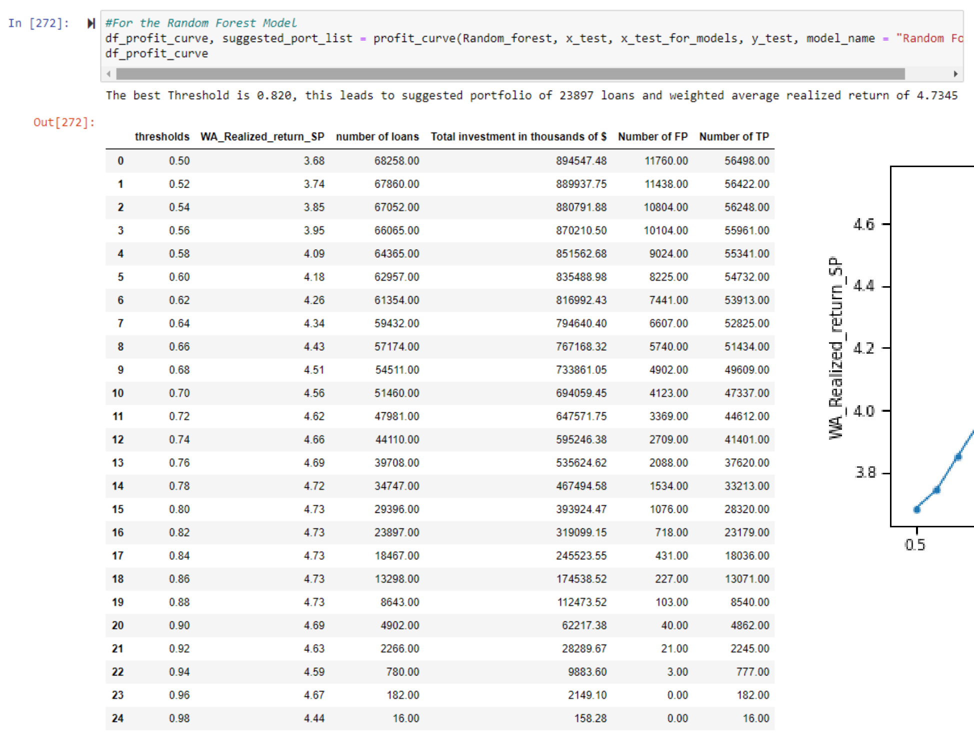
1. **model** - A model **After fitting** to x_train_for_models (the original x_train without the columns: "id", "funded_amnt_inv", "realized_return" (those columns are needed for calculating the actual realized return gained by the suggested portfolio)
2. **x_test** - the original x_test (with all the columns)
3. **x_test_for_models** - the original x_test without the columns: "id", "funded_amnt_inv", "realized_return" (those columns are needed for calculating the actual realized return gained by the suggested portfolio)
4. **y_test** - y_test array
5. **display_df_profit_curve** - a boolean. default = False. If you wish to see a data frame with the different thresholds, the weighted average realized return (of the Suggested Portfolio that was chosen as a result of the threshold), the number of loans that were left in the suggested portfolio (as a result of the threshold), The total investment in dollars (If the company would have invested in all the loans in the suggested portfolio for a given threshold it would have invested the amount of money in this column) and the amount of FP and TP.
6. **model_name** - a string specifying the name of the model (only for convenience). If not given, the model will use the model argument for the name.

This function returns 2 values + Profit Curve:

1. **df_profit_curve** = a data frame with the different thresholds, the weighted average realized return (of the Suggested Portfolio that was chosen as a result of the threshold), the number of loans that were left in the suggested portfolio (as a result of the threshold), the total money (in thousands of dollars) the company would need to invest, and the number of the FP and TP cases for each threshold.
2. **suggested_port_list**: a list containing all the suggested portfolios that we got because of each and every threshold.

Note: The function **outputs a profit curve**, **the green dot** represents the **best** threshold (the one that gained the highest realized return). The function will print that threshold, the number of loans that participated in the calculation of the realized return and it will also print that highest realized return.

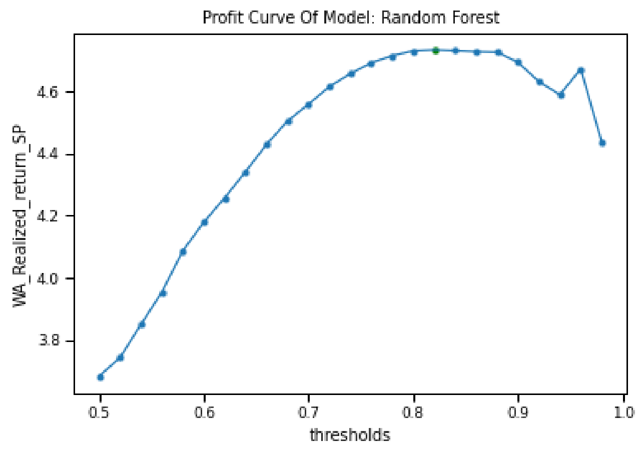
Example:



```
In [272]:  #For the Random Forest Model
           df_profit_curve, suggested_port_list = profit_curve(Random_forest, x_test, x_test_for_models, y_test, model_name = "Random Fo
           df_profit_curve

           The best Threshold is 0.820, this leads to suggested portfolio of 23897 loans and weighted average realized return of 4.7345
Out[272]:
```

	thresholds	WA_Realized_return_SP	number of loans	Total investment in thousands of $	Number of FP	Number of TP
0	0.50	3.68	68258.00	894547.48	11760.00	56498.00
1	0.52	3.74	67860.00	889937.75	11438.00	56422.00
2	0.54	3.85	67052.00	880791.88	10804.00	56248.00
3	0.56	3.95	66065.00	870210.50	10104.00	55961.00
4	0.58	4.09	64365.00	851562.68	9024.00	55341.00
5	0.60	4.18	62957.00	835488.98	8225.00	54732.00
6	0.62	4.26	61354.00	816992.43	7441.00	53913.00
7	0.64	4.34	59432.00	794640.40	6607.00	52825.00
8	0.66	4.43	57174.00	767168.32	5740.00	51434.00
9	0.68	4.51	54511.00	733861.05	4902.00	49609.00
10	0.70	4.56	51460.00	694059.45	4123.00	47337.00
11	0.72	4.62	47981.00	647571.75	3369.00	44612.00
12	0.74	4.66	44110.00	595246.38	2709.00	41401.00
13	0.76	4.69	39708.00	535624.62	2088.00	37620.00
14	0.78	4.72	34747.00	467494.58	1534.00	33213.00
15	0.80	4.73	29396.00	393924.47	1076.00	28320.00
16	0.82	4.73	23897.00	319099.15	718.00	23179.00
17	0.84	4.73	18467.00	245523.55	431.00	18036.00
18	0.86	4.73	13298.00	174538.52	227.00	13071.00
19	0.88	4.73	8643.00	112473.52	103.00	8540.00
20	0.90	4.69	4902.00	62217.38	40.00	4862.00
21	0.92	4.63	2266.00	28289.67	21.00	2245.00
22	0.94	4.59	780.00	9883.60	3.00	777.00
23	0.96	4.67	182.00	2149.10	0.00	182.00
24	0.98	4.44	16.00	158.28	0.00	16.00



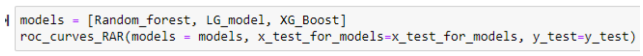
ROC Curve – illustration*:*

This function gets 3 arguments:

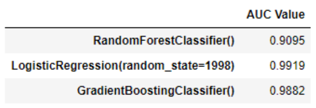
1. **models** - a list of models **after fitted to the x_train and y_train**
2. **x_test** - a **x_test_for_models** array
3. **y_test** - a y_test array (The true labels)

This function **returns ROC Curves of all the models inserted in the "models" argument** and returns the **AUC** values.

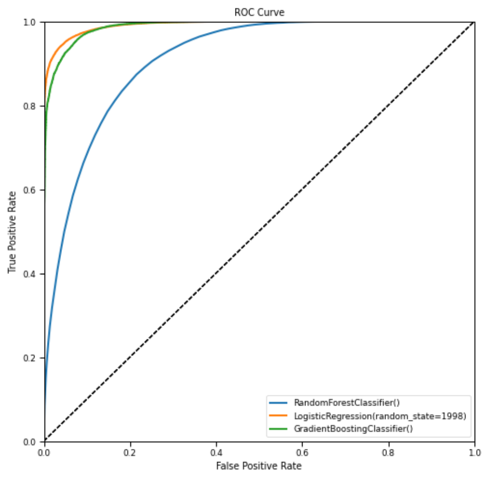
Example:



```
models = [Random_forest, LG_model, XG_Boost]
roc_curves_RAR(models = models, x_test_for_models=x_test_for_models, y_test=y_test)
```



	AUC Value
RandomForestClassifier()	0.9095
LogisticRegression(random_state=1998)	0.9919
GradientBoostingClassifier()	0.9882



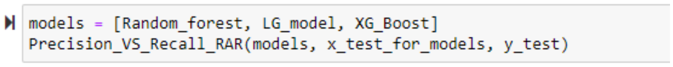
Precision_VS_Recall_RAR – illustration*:*

This function gets 3 arguments:

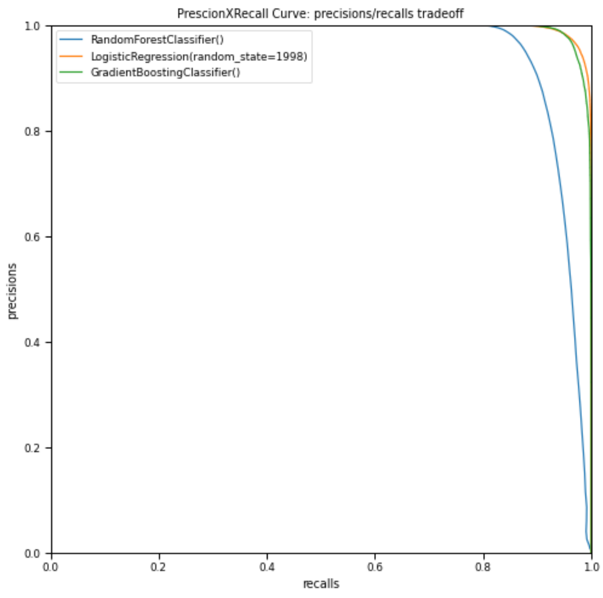
1. **models** - a list of models **after fitted to the x_train and y_train**
2. **x_test** - a **x_test_for_models** array
3. **y_test** - a y_test array (The true labels)

This function **returns Precision VS Recall Curves of all the models inserted in the "models" argument**.

Example:



```
models = [Random_forest, LG_model, XG_Boost]
Precision_VS_Recall_RAR(models, x_test_for_models, y_test)
```



List of the removed columns:

"member_id", "funded_amnt",
"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_inq",
"mths_since_recent_revol_delinq",
"num_rev_tl_bal_gt_0", "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", "realized_return_3",
#try generall:
"il_util","bc_open_to_buy", "bc_util", "mo_sin_old_il_acct",
"mths_since_recent_bc","percent_bc_gt_75","emp_length",
#try_revolving:
"open_rv_12m","open_rv_24m", "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"