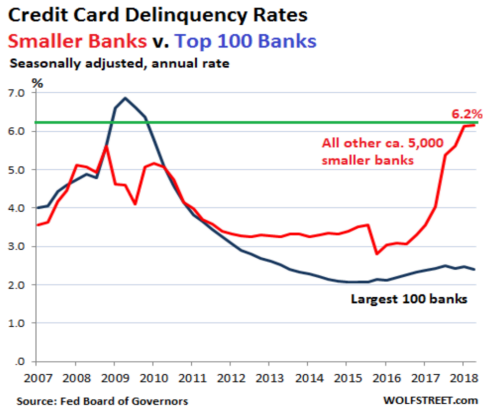
Capstone Project

Machine Learning Engineer Nanodegree September 20th, 2018

Definition

Project Overview

In the field of credit lending, people hope to solve some short-term funding gaps by using loans. The lender wants to make a profit with a small amount of interest, but there is currently no 100% way to determine/predict the credit repayment ability. This may result in people who are unable to repay being used by bad lenders, therefore leads to bankruptcy of credit, which indirectly causes certain public security/economic problems [1]. Various federal agencies also required lenders to assess a consumer's ability to repay a home loan, and the creditor must consider and evaluate at least eight factors, including “Current employment status”, “Monthly payments on a simultaneous loan”, “Monthly debt-to-income ratio or residual income”, and so on [1]. Fig 1[13]. shows smaller banks have been more seriously affected than top 100 banks.



Several improvements have been developed in the credit scoring domain. In [2], an empirical comparison of various resampling techniques has been discussed in order to deal with imbalanced data. [3] shows that ensembles of classifiers achieve better results for credit risk assessment, and [4] demonstrates that Bayesian hyper-parameter optimization performs better than random search, grid search. Recently, “Home-Credit” has launched a machine learning competition “Home Credit Default Risk” on Kaggle [5].

Problem Statement

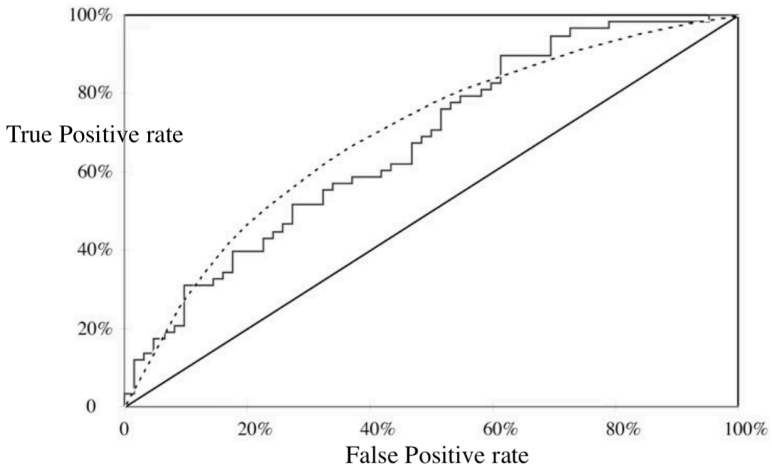
The goal is to build a prediction model to classify the clients’ repayment abilities. By using a variety of alternative data--including telco and transactional information, if we can accurately predict the clients who have ability to repay, but with insufficient or non-existent credit histories, then we can expand the business for the company and avoid untrustworthy lending to this population. This problem is a machine learning **classification** task, TARGET=0 means the client will repay on time and TARGET=1 means the client have some difficulty

repaying loan. And because the targets for training are already known and labeled, **supervised machine learning algorithms including logistic regression and gradient-boosting(e.g. XGBoost, LightGBM) are potential solutions.**

Metrics

The evaluation metric used is "**Area under the ROC curve**"(also called AUC) [12], where the ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. AUC is between 0 and 1(the higher AUC, the better model). The AUC can be calculated by the equation below, where T is the threshold parameter and X1(X0) is the score for a positive(negative) instance.

$$AUC = \int_{\infty}^{-\infty} TPR(T)FPR'(T)\,dT = P(X_1 > X_0)$$



Analysis

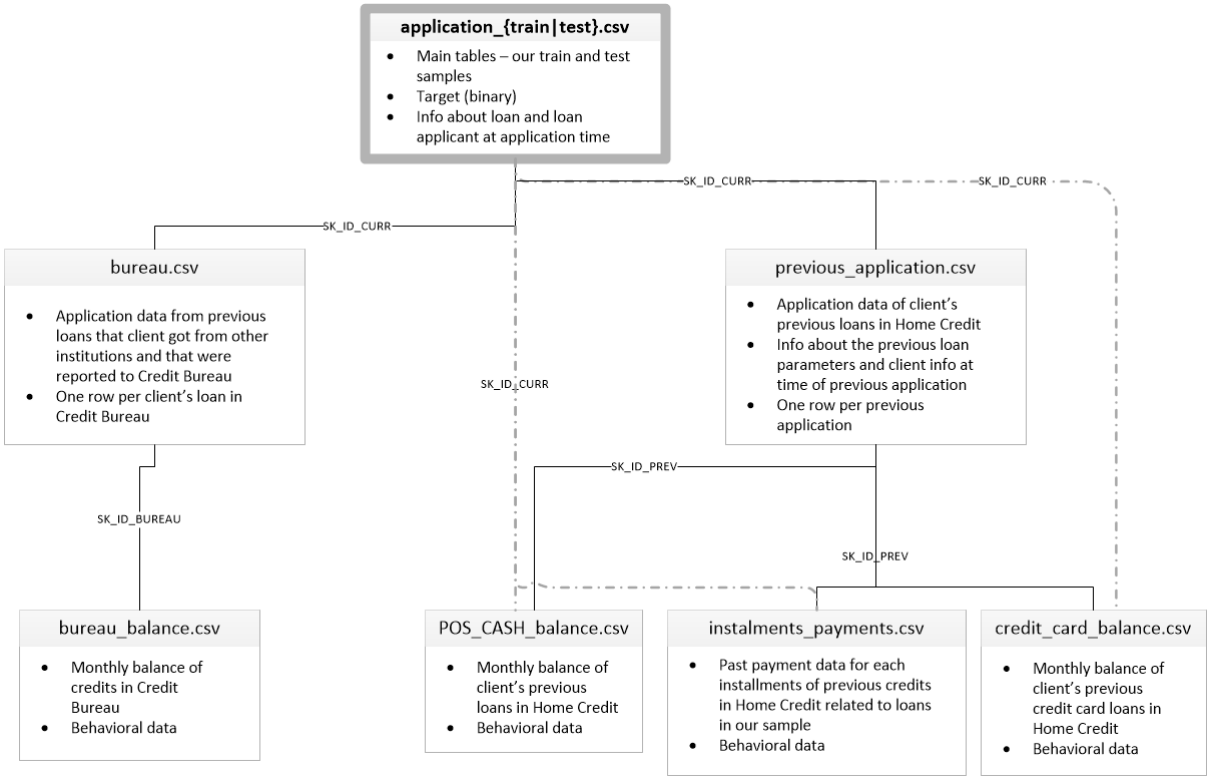
Data Exploration

The datasets are provided by Home-Credit-Group for use in the competition, and are available for download on the Kaggle competition data page [6]. There are mainly 9 CSV files where "HomeCredit_columns_description.csv" contains all descriptions and "application_{train|test}.csv" contains all applications' static data(**307511 records for training and 48744 records for testing**) for the use of training and testing. Besides "TARGET", there are **121 features** in application_{train|test}.csv, it contains **105 numerical and 16 categorical** features. And 282686 examples' TARGET equal 0, 24825 examples' TARGET equal 1, Which means the percentage of "TARGET==1" is only **8.1%**. In order to deal with this **imbalanced** data, we would resampling from the dataset or assign misclassification costs to force the classifier to concentrate on the minority classes. We will split the training data into training and validation sets to evaluate/optimize our model, then apply the optimized model on the testing data provided to get the final prediction.

- The other 6 CSV files contain additional informations briefly described below:

bureau.csv	All client's previous credits provided by other financial institutions that were reported to Credit Bureau.
bureau_balance.csv	Monthly balances of previous credits in Credit Bureau.
POS_CASH_balance.csv	Monthly balance snapshots of previous POS (point of sales) and cash loans that the applicant had with Home Credit.
credit_card_balance.csv	Monthly balance snapshots of previous credit cards that the applicant has with Home Credit.
previous_application.csv	All previous applications for Home Credit loans of clients who have loans in our sample.
installments_payments.csv	Repayment history for the previously disbursed credits in Home Credit related to the loans

- The diagram below shows the relationships between all of the dataset:



All these 8 tables generally use prefix(or suffix) to represent different groups of features, and mainly contains the following fields:

- "AMT_" represents amount like INCOME, PRICE, ANNUITY, etc.
- "DAYS_" represents time-related features like BIRTH, EMPLOYED, FIRST/LAST DUE, etc.

- "NAME_" represents categorical features like INCOME TYPE, EDUCATION TYPE, FAMILY STATUS, etc.
- "FLAG_" represents binary features like Y/N or 1/0.
- "_ID" represents ID features like CURRENT, PREVIOUS, and BUREAU.

There are some other features not in the groups above, for example, in the application_train.csv, 'CODE_GENDER' represents 'Gender of the client', and it would be 'F' for female, 'M' for male, and 'XNA' for unknown(missing). we will discuss and deal with those if need.

These 8 relational database CSV files contain 3 groups of clients' static data(e.g. gender, occupation type, etc.) and dynamic data(e.g. credit history, repayment history, etc.), that is, **current application, previous application, and bureau report**. Tables below shows each datasets' sample size, # of categorical feaures, # of numerical features, and so on. We will use SK_ID_PREV for grouping 4 previous application tables, SK_ID_BUREAU for grouping 2 bureau reports, and use SK_ID_CURR to group them all.

1. Current application

File name	samples(rows)	features(columns)				
		Categorical	Numerical	ID	TARGET	Total
application_train	307511	16	104	1	1	122
application_test	48744	16	104	1	0	121

applcation_train

SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	...	REGION_POPULATION_RELATIVE	DAYS_BIRTH	DAYS_EMPLOYED	DAYS_REGISTRATION
100002	1	Cash loans	M	...	0.018801	-9461	-637	-3648.0
100003	0	Cash loans	F	...	0.003541	-16765	-1188	-1186.0
100004	0	Revolving loans	M	...	0.010032	-19046	-225	-4260.0
100006	0	Cash loans	F	...	0.008019	-19005	-3039	-9833.0
100007	0	Cash loans	M	...	0.028663	-19932	-3038	-4311.0

applcation_test

SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	...	DAYS_BIRTH	DAYS_EMPLOYED	DAYS_REGISTRATION
100001	Cash loans	F	N	...	-19241	-2329	-5170.0
100005	Cash loans	M	N	...	-18064	-4469	-9118.0
100013	Cash loans	M	Y	...	-20038	-4458	-2175.0
100028	Cash loans	F	N	...	-13976	-1866	-2000.0
100038	Cash loans	M	Y	...	-13040	-2191	-4000.0

2. Previous application

File name	samples(rows)	features(columns)			
		Categorical	Numerical	ID	Total
previous_application	1670214	16	19	2	37
credit_card_balance	3840312	1	20	2	23
POS_CASH_balance	10001358	1	5	2	8
installments_payments	13605401	0	6	2	8

previous_application

SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	...	DAYS_LAST_DUE_1ST_VERSION	DAYS_LAST_DUE	DAYS_TERMINATION
1369693	100001	Consumer loans	3951.000	...	-1499.0	-1619.0	-1612.0
1038818	100002	Consumer loans	9251.775	...	125.0	-25.0	-17.0
1810518	100003	Cash loans	98356.995	...	-386.0	-536.0	-527.0
2396755	100003	Consumer loans	6737.310	...	-1980.0	-1980.0	-1976.0
2636178	100003	Consumer loans	64567.665	...	-647.0	-647.0	-639.0

credit_card_balance

SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	...	CNT_INSTALMENT_MATURE_CUM	NAME_CONTRACT_STATUS	SK_DPD	SK_DPD_DEF
1489396	100006	-3	0.00	...	0.0	Active	0	0
1489396	100006	-2	0.00	...	0.0	Active	0	0
1489396	100006	-1	0.00	...	0.0	Active	0	0
1843384	100011	-75	189000.00	...	NaN	Active	0	0
1843384	100011	-74	184568.85	...	1.0	Active	0	0

POS_CASH_balance

SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALMENT_FUTURE	NAME_CONTRACT_STATUS	SK_DPD	SK_DPD_DEF
1369693	100001	-54	4.0	1.0	Active	0	0
1369693	100001	-53	4.0	0.0	Completed	0	0
1851984	100001	-96	4.0	2.0	Active	0	0
1851984	100001	-95	4.0	1.0	Active	7	7
1851984	100001	-94	4.0	0.0	Active	0	0

installments_payments

SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_INSTALMENT	DAYS_ENTRY_PAYMENT	AMT_INSTALMENT
1369693	100001	1.0	1	-1709.0	-1715.0	3951.00
1369693	100001	1.0	2	-1679.0	-1715.0	3951.00
1369693	100001	2.0	4	-1619.0	-1628.0	17397.90
1369693	100001	1.0	3	-1649.0	-1660.0	3951.00
1851984	100001	1.0	2	-2916.0	-2916.0	3982.05

3. Bureau report

File name	samples(rows)	features(columns)			
		Categorical	Numerical	ID	Total
bureau	1716428	3	12	2	17
bureau_balance	27299925	1	1	1	3

bureau

SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	...	AMT_CREDIT_MAX_OVERDUE	CNT_CREDIT_PROLONG	AMT_CREDIT_SUM
100001	5896630	Closed	currency 1	...	NaN	0	112500.0
100001	5896631	Closed	currency 1	...	NaN	0	279720.0
100001	5896632	Closed	currency 1	...	NaN	0	91620.0
100001	5896633	Closed	currency 1	...	NaN	0	85500.0
100001	5896634	Active	currency 1	...	NaN	0	337680.0

bureau_balance

SK_ID_BUREAU	MONTHS_BALANCE	STATUS
5001709	-2	C
5001709	-1	C
5001709	0	C
5001710	-82	X
5001710	-81	X

For such complex and large-scale data sets, many missing values and outliers are usually included, as described above. The number of columns containing missing values(could be in the categorical or numerical columns) for each file is shown in the table below.

File name	# of columns contain missing values (# of columns)	Missing value percentange
application_train|test	68(121)	16 features > 60%
previous_application	16(37)	RATE_INTEREST_PRIMARY and RATE_INTEREST_PRIVILEGED > 99%
credit_card_balance	9(23)	8~20%
POS_CASH_balance	2(8)	0.3%
installments_payments	2(8)	0.02%
bureau	7(17)	AMT_CREDIT_MAX_OVERDUE and AMT_ANNUITY > 60%
bureau_balance	0(3)	NaN

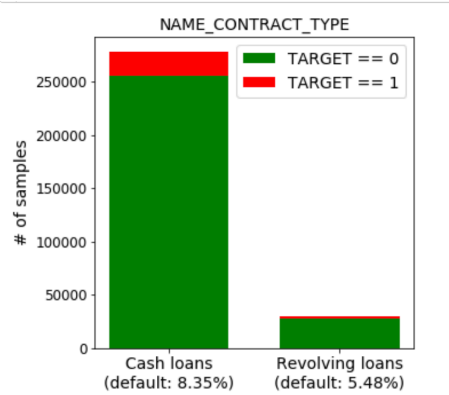
Some values are outliers and anomalies with no doubt, for example, in the previous_application.csv, 'DAYS_LAST_DUE' means 'last due date of the previous application relative to the application', so like 365243(1000 years in the future...) such a value is definitely an anomaly. In the bureau.csv, 'DAYS_CREDIT_ENDDATE' means 'remaining duration of CB credit relative to the application', so a value like -41847(115 years before...) is definitely anomaly. Sometimes it's hard to known if a big number is impossible('anomaly') or just a special client('outlier'). We must check more about this client to see if certain values are outliers or anomalies. For example, in the application_train.csv, 'AMT_INCOME_TOTAL' means 'Income of the client', and there is one client's total income is "117 millions". It could be correct for this client, but if we check some static datas related to her, we have more confident to think it's just a typo.

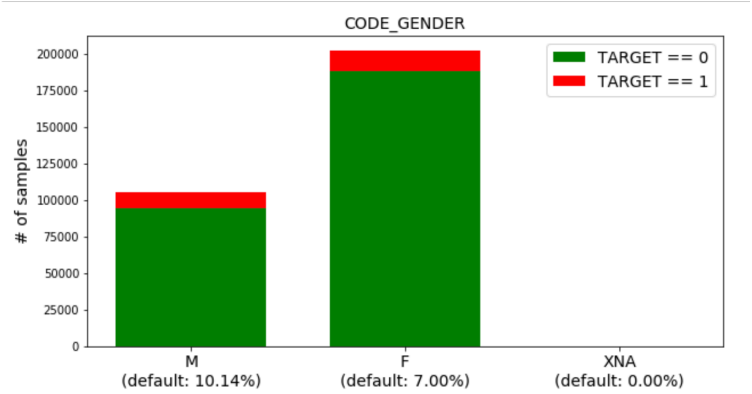
	OCCUPATION_TYPE	NAME_EDUCATION_TYPE	AGE	NAME_TYPE_SUITE	AMT_INCOME_TOTAL
12840	Laborers	Secondary / secondary special	34	Unaccompanied	117000000.00

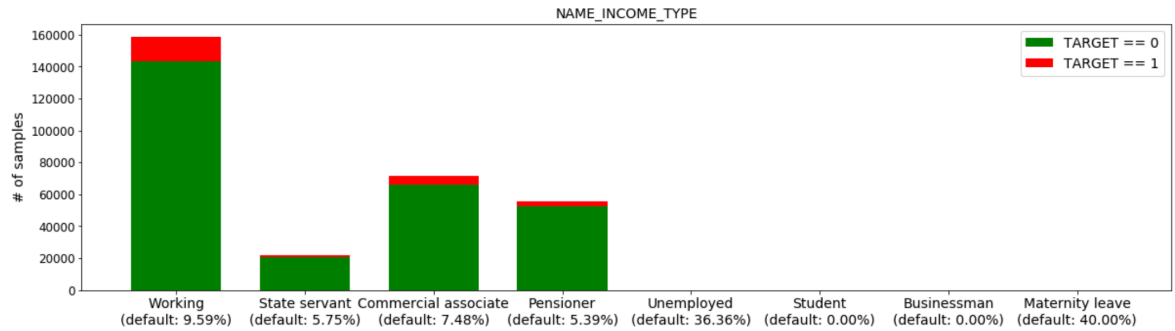
Exploratory Visualization

- **Categorical features**

To categorical features, we check the repayment/default percentage for each different category. For example, according to the charts below, we know that there are more clients have cash loan contracts than revolving loans. The bad news is that their loans are easy to default('Cash loans' has 8.35% TARGET==1, and 'Revolving loans' has 5.48%). The good news is, we have more female clients(235126 > 121125) and their default rate is lower(7.00% < 10.14%). If we check the clients' income type, we will find that if she is on maternity leave, or he/she is unemployed, then he/she is likely to default on the loan(~40%). Since they are just a small group and the situation wouldn't take too long, we don't have to pay too much attention on them. Based on all these observations, we will understand the meaning of each categorical features, and then use TARGET-encoding to deal with them.



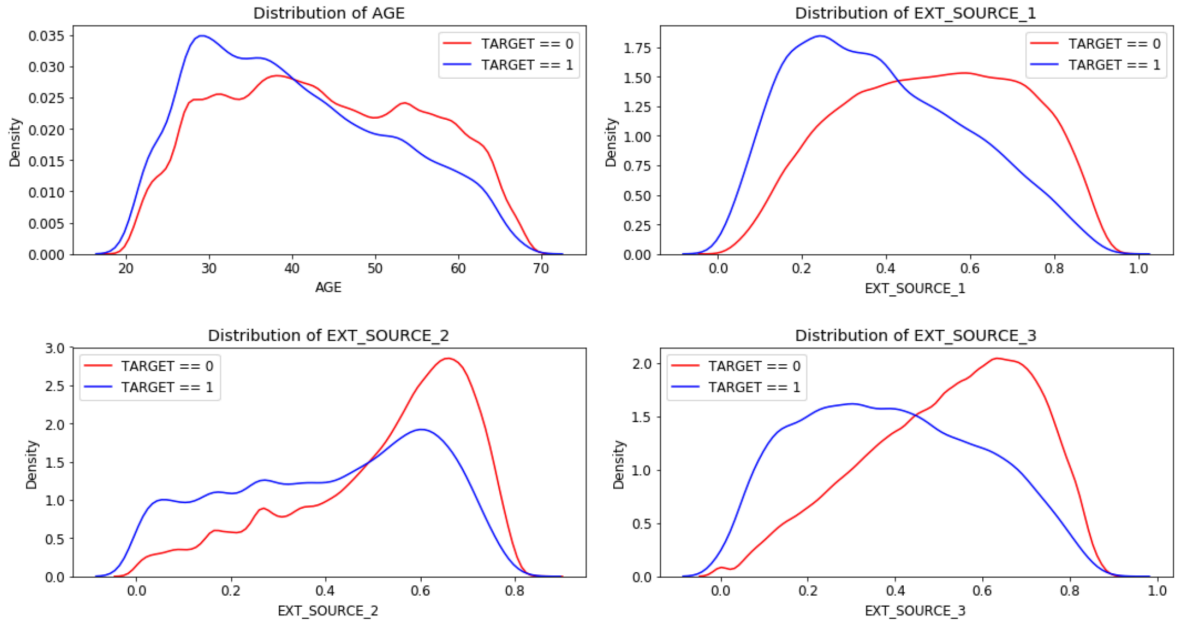




- **Numerical features**

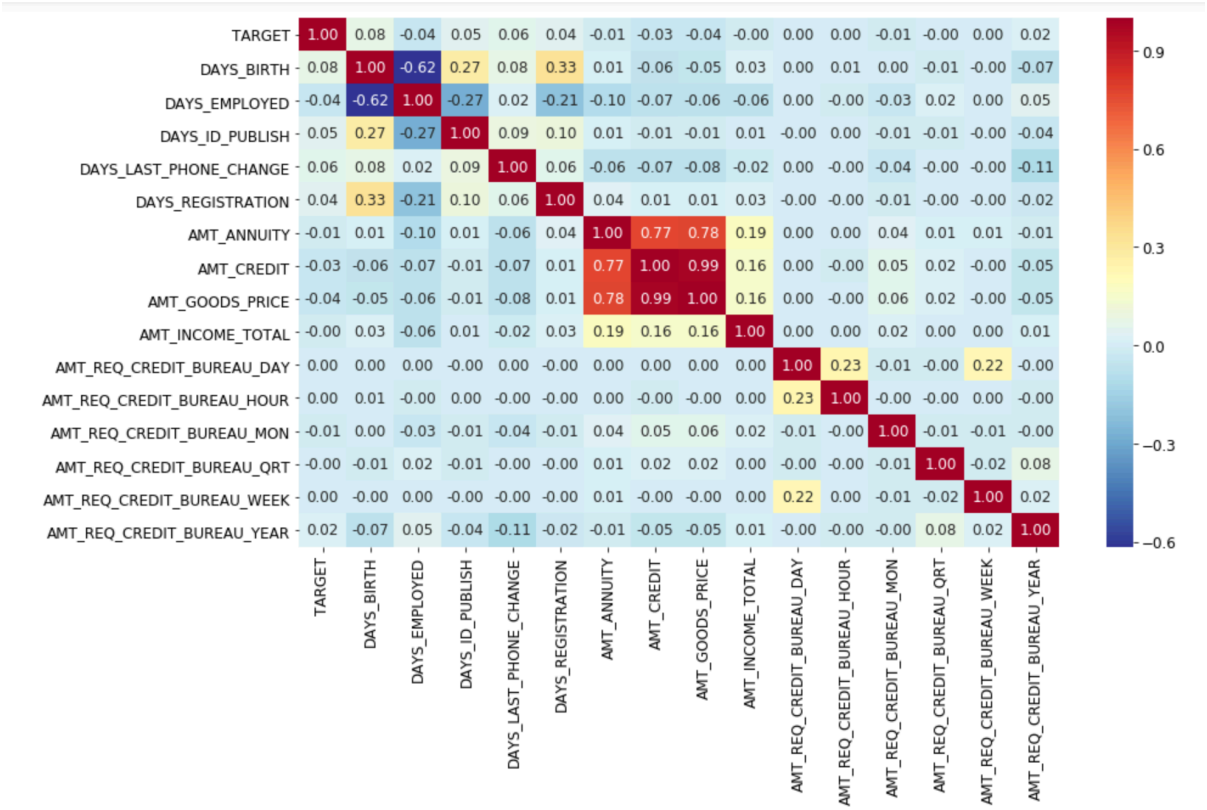
KDE(kernel density estimate) plot

To numerical features, we use KDE(kernel density estimate) plots for checking the distribution of each features by the TARGET value. For example, according to the charts below, we know that if the client is younger or has lower normalized score from external data source 'EXT_SOURCE_1', then he/she may have higher default rate. If the client has higher normalized scores from external data source 'EXT_SOURCE_2' and/or 'EXT_SOURCE_3', he/she might repay on time.

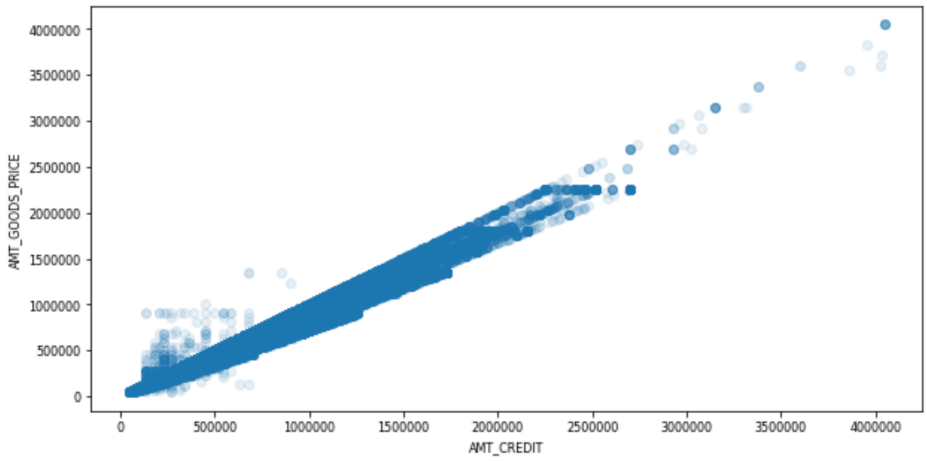


Correlation heatmap

From the correlation heatmap, we could more understand the relationships between each feeatures. For example, according to the heatmap below, we can see that DAYS_BIRTH is positively correlated with TARGET(0.08). It indicates that as the value of DAYS_BIRTH increases(younger age), the client is more likely to default on the loan(TARGET==1). i.e. if the client is 20 years old, then DAYS_BIRTH = -20*365 = -8030, the negative sign means 8030 days before the application day.

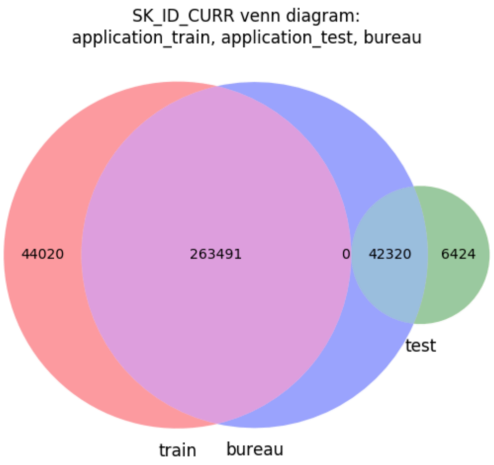


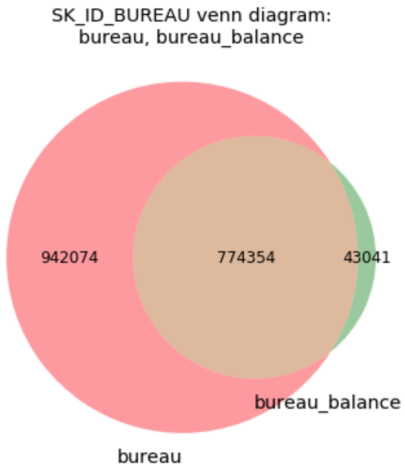
We can also see that 'AMT_CREDIT' has a strong possitive correlationship with 'AMT_GOODS_PRICE'(0.99), meaning that if 'the price of the goods for which the loan is given' increases, 'the credit amount of the loan' will also increases. If we examine the scatter plot, we can see that there is a roughly linear relationship between 'AMT_CREDIT' and 'AMT_GOODS_PRICE'. Based on these observations, we can fill-in missing values by multiplying a constant from another column if there are strong linear relationship between them. We can also do some feature-engineering to generate new handcrafted features by using these linear relationships.



- **ID**

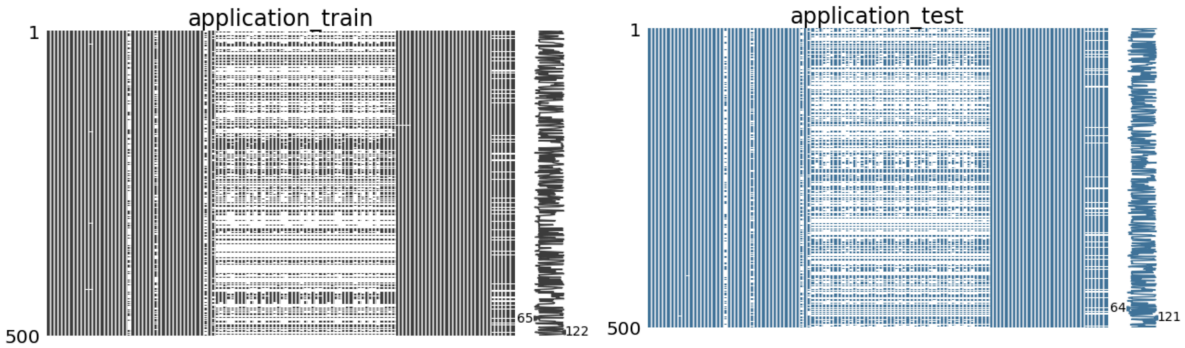
Because we have to combine all the relational datasets into one file to train the model, it is good to know how many samples in one dataset have the same 'ID' in the other. For example, according to the Venn diagrams below, we can see that in the bureau.csv file, 263491 samples can be combined into application_train.csv with the same 'SK_ID_CURR', and 42320 samples can be combined into application_test.csv. In addition, we can see that there are 43041 samples in bureau_balance.csv that have no common 'SK_ID_BUREAU' with bureau.csv.





- **Visualize missing values**

The data-dense diagram below shows that 'application_train' and 'application_test' have similar missing value patterns.



Algorithms and Techniques

There are many relational databases for this problem, and file types contain only numbers and strings. The goal is to use the training set to predict another test set. Based on this situation, we will use a typical supervised machine learning algorithm to solve this problem. The step-by-step workflow is listed below:

1. For each relational databases, including 'application_train', 'application_test', 'bureau', 'bureau_balance', 'previous_application', 'credit_card_balance', 'installments_payments', and 'POS_CASH_balance', we will:

    1.1. **Anomalies**: Find and replace into 0 or NaN. (If possible, replace into some specific reasonable value)

    1.2. **Categorical** features: Transform into numerical features by using 'Likelihood Encoding' or 'One-Hot-Encoding' or 'Label-Encoding' depending on different situations.

    1.3. **New hand-crafted features**: Find useful relationships, such as differences between two features, ratios (one column divided by another), or multiply by two columns to generate new hand-crafted features.

2. **Feature Engineering**: Group all relational databases by using 'SK_ID_CURR', 'SK_ID_PREV', and 'SK_ID_BUREAU'. Manually create some specific features like mins, maxes, sums, etc.

3. Use Gradient-Boosting models like XGBoost, LightGBM to make predictions, one advantage is these models will internally deal with missing values.

4. Hyperparameter Tuning: we will use GridSearch to find the better hyperparameters' value, and split the whole dataset into training and validation subsets to prevent overfitting.

5. **Stacking**: Using different models' predictions as features for second level model to make the final prediction.

Benchmark

We use a **logistic regression classifier with default hyperparameters** from scikit-learn library as the benchmark model. The benchmark model has an **Kaggle privateScore** of **0.61507(AUC)** on the testing set. In the benchmark model, we only use one file for training, which is "application_train.csv". Necessary but simplest "Data cleaning and formatting" methods, such as one-hot encoding, missing values handling, and scaling, are applied in order to run the logistic regression model.

Methodology

Data Preprocessing

For each relational dataset, we will preprocess outliers/exceptions, transform the categorical features into numbers, create handcrafted features, perform feature engineering, and finally group them into new csv files for later model training.

• bureau_balance.csv and bureau.csv

bureau_balance

- Hand-crafted features

```
bureau_bal_ = bureau_bal.copy()
bureau_bal_['STATUS'].replace('X', 0, inplace=True)
bureau_bal_['STATUS'].replace('C', 0, inplace=True)
bureau_bal_['STATUS'].replace('0', 0, inplace=True)
bureau_bal_['STATUS'].replace('1', 1, inplace=True)
bureau_bal_['STATUS'].replace('2', 2, inplace=True)
bureau_bal_['STATUS'].replace('3', 3, inplace=True)
bureau_bal_['STATUS'].replace('4', 4, inplace=True)
bureau_bal_['STATUS'].replace('5', 5, inplace=True)
bureau_bal_['MONTHS_BALANCE_abs'] = bureau_bal['MONTHS_BALANCE'].abs() + 1
bureau_bal['STATUS_weight'] = bureau_bal_['STATUS'] / bureau_bal_['MONTHS_BALANCE_abs']
```

- Feature engineering

```
bureau_bal = pd.get_dummies(bureau_bal)
features_months = bureau_bal.groupby(['SK_ID_BUREAU'])['MONTHS_BALANCE'].agg({'max', 'min', 'size'}).reset_index().\
rename(columns={'max': 'BUREAU_BAL_month_bal_max', 'min': 'BUREAU_BAL_month_bal_min', 'size': 'Duration'})

status_features = ['STATUS_0', 'STATUS_1', 'STATUS_2', 'STATUS_3', 'STATUS_4',\
                   'STATUS_5', 'STATUS_C', 'STATUS_X', 'STATUS_weight']
features_status = bureau_bal.groupby(['SK_ID_BUREAU'])[status_features].\
agg({'sum', 'mean', 'std', pd.DataFrame.kurt}).reset_index()
features_status.columns = pd.Index(['SK_ID_BUREAU']+\
                                   ['BUREAU_BAL_' + e[0] + "_" + e[1] for e in features_status.columns.tolist()[1:]])
```

bureau

- Outliers and anomalies

```
bureau['CREDIT_TYPE'].replace(['Another type of loan', 'Unknown type of loan',\
                               'Loan for working capital replenishment', 'Cash loan (non-earmarked)',\
                               'Real estate loan', 'Loan for the purchase of equipment',\
                               'Loan for purchase of shares (margin lending)', 'Mobile operator loan',\
                               'Interbank credit'], 'Other_loan', inplace=True)

bureau.loc[bureau.DAYS_CREDIT_ENDDATE < -50*365, 'DAYS_CREDIT_ENDDATE'] = np.nan
bureau.loc[bureau.DAYS_ENDDATE_FACT < -50*365, 'DAYS_ENDDATE_FACT'] = np.nan
bureau.loc[bureau.AMT_CREDIT_MAX_OVERDUE > 10000000, 'AMT_CREDIT_MAX_OVERDUE'] = np.nan
bureau.loc[bureau.AMT_CREDIT_SUM > 100000000, 'AMT_CREDIT_SUM'] = np.nan
bureau.loc[bureau.AMT_CREDIT_SUM_DEBT < 0 , 'AMT_CREDIT_SUM_DEBT'] = np.nan
bureau.loc[bureau.AMT_CREDIT_SUM_DEBT > 100000000, 'AMT_CREDIT_SUM_DEBT'] = np.nan
bureau.loc[bureau.AMT_ANNUITY > 10000000 , 'AMT_ANNUITY'] = np.nan
```

```
# merge bureau balance by SK_ID_BUREAU
bureau = bureau.merge(features_months, how='outer', on='SK_ID_BUREAU')
bureau = bureau.merge(features_status, how='outer', on='SK_ID_BUREAU')
```

- Label encoding

```
bureau['CREDIT_TYPE'].replace('Consumer credit', 0, inplace=True)
bureau['CREDIT_TYPE'].replace('Credit card', 1, inplace=True)
bureau['CREDIT_TYPE'].replace('Car loan', 2, inplace=True)
bureau['CREDIT_TYPE'].replace('Mortgage', 3, inplace=True)
bureau['CREDIT_TYPE'].replace('Microloan', 4, inplace=True)
bureau['CREDIT_TYPE'].replace('Other_loan', 5, inplace=True)
bureau['CREDIT_TYPE'].replace('Loan for business development', 6, inplace=True)

bureau['CREDIT_CURRENCY'].replace('currency 1', 0, inplace=True)
bureau['CREDIT_CURRENCY'].replace('currency 2', 1, inplace=True)
bureau['CREDIT_CURRENCY'].replace('currency 3', 2, inplace=True)
bureau['CREDIT_CURRENCY'].replace('currency 4', 3, inplace=True)

bureau['CREDIT_ACTIVE'].replace('Closed', 0, inplace=True)
bureau['CREDIT_ACTIVE'].replace('Active', 1, inplace=True)
bureau['CREDIT_ACTIVE'].replace('Sold', 2, inplace=True)
bureau['CREDIT_ACTIVE'].replace('Bad debt', 3, inplace=True)
```

Feature Engineering

- diff features engineering

```
bureau['diff_DAYS_CREDIT'] = bureau['DAYS_CREDIT_UPDATE'] - bureau['DAYS_CREDIT']
bureau['diff_ENDDATE'] = bureau['DAYS_CREDIT_ENDDATE'] - bureau['DAYS_ENDDATE_FACT']
bureau['diff_OVERDUE_UPDATE'] = bureau['DAYS_CREDIT_UPDATE'] - bureau['CREDIT_DAY_OVERDUE']
bureau['diff_CREDIT_DEBT'] = bureau['AMT_CREDIT_SUM'] - bureau['AMT_CREDIT_SUM_DEBT']
bureau['diff_ANNUITY_OVERDUE'] = bureau['AMT_ANNUITY'] - bureau['AMT_CREDIT_SUM_OVERDUE']
bureau['diff_ANNUITY_MAX_OVERDUE'] = bureau['AMT_ANNUITY'] - bureau['AMT_CREDIT_MAX_OVERDUE']
bureau['diff_ANNUITY_CREDIT_SUM'] = bureau['AMT_ANNUITY'] - bureau['AMT_CREDIT_SUM']
bureau['diff_ANNUITY_CREDIT_SUM_DEBT'] = bureau['AMT_ANNUITY'] - bureau['AMT_CREDIT_SUM_DEBT']

diff_features = ['diff_DAYS_CREDIT', 'diff_ENDDATE', 'diff_OVERDUE_UPDATE', 'diff_CREDIT_DEBT',\
                 'diff_ANNUITY_OVERDUE', 'diff_ANNUITY_MAX_OVERDUE', 'diff_ANNUITY_CREDIT_SUM',\
                 'diff_ANNUITY_CREDIT_SUM_DEBT']

features_diff = bureau.groupby(['SK_ID_CURR'])[diff_features].agg({'max', 'min', 'mean', 'median'}).reset_index()
features_diff.columns = pd.Index(['SK_ID_CURR']+\
                                 ["BUREAU_" + e[0] + "_" + e[1] for e in features_diff.columns.tolist()[1:]])
```

- ratio/multiply features engineering

```
bureau['ratio_CREDIT_OVERDUE'] = bureau['DAYS_CREDIT'] /(1 + bureau['CREDIT_DAY_OVERDUE'])
bureau['ratio_OVERDUE'] = bureau['AMT_CREDIT_SUM_OVERDUE'] /(1 + bureau['AMT_CREDIT_MAX_OVERDUE'])
bureau['ratio_OVERDUE_DEBT'] = bureau['AMT_CREDIT_SUM_OVERDUE'] /(1 + bureau['AMT_CREDIT_SUM_DEBT'])
bureau['ratio_OVERDUE_ANNUITY'] = bureau['AMT_CREDIT_SUM_OVERDUE'] /(1 + bureau['AMT_ANNUITY'])
bureau['ratio_OVERDUE_CREDIT'] = bureau['AMT_CREDIT_SUM_OVERDUE'] /(1 + bureau['AMT_CREDIT_SUM'])
bureau['ratio_CREDIT_LIMIT_ANNUITY'] = bureau['AMT_CREDIT_SUM_LIMIT'] /(1 + bureau['AMT_ANNUITY'])
bureau['ratio_DEBT'] = bureau['AMT_CREDIT_SUM_DEBT'] /(1 + bureau['AMT_CREDIT_SUM'])
bureau['ratio_DEBT_LIMIT'] = bureau['AMT_CREDIT_SUM_DEBT'] /(1 + bureau['AMT_CREDIT_SUM_LIMIT'])
bureau['ratio_LIMIT_SUM'] = bureau['AMT_CREDIT_SUM_LIMIT'] /(1 + bureau['AMT_CREDIT_SUM'])

bureau['multiply_OVERDUE'] = bureau['CREDIT_DAY_OVERDUE'] * bureau['AMT_CREDIT_MAX_OVERDUE']
bureau['multiply_PROLONG_OVERDUE'] = bureau['CNT_CREDIT_PROLONG'] * bureau['AMT_CREDIT_MAX_OVERDUE']

ratio_features = ['ratio_CREDIT_OVERDUE', 'ratio_OVERDUE', 'ratio_OVERDUE_DEBT', 'ratio_OVERDUE_ANNUITY',\
                  'ratio_OVERDUE_CREDIT', 'ratio_CREDIT_LIMIT_ANNUITY', 'ratio_DEBT', 'ratio_DEBT_LIMIT',\
                  'ratio_LIMIT_SUM']
multiply_features = ['multiply_OVERDUE', 'multiply_PROLONG_OVERDUE']

features_ratio = bureau.groupby(['SK_ID_CURR'])[ratio_features + multiply_features].agg({'max', 'var'}).reset_index()
features_ratio.columns = pd.Index(['SK_ID_CURR'] +\
                                  ['BUREAU_' + e[0] + "_" + e[1] for e in features_ratio.columns.tolist()[1:]])
```

- Other numerical features

```
bureau_ = bureau.drop(columns = diff_features + ratio_features + multiply_features + ['SK_ID_BUREAU'])
features_numerical = bureau_.groupby(['SK_ID_CURR']).agg({'max', 'mean'}).reset_index()
features_numerical.columns = pd.Index(['SK_ID_CURR'] +\
                                      ['BUREAU_numerical_' + e[0] + "_" + e[1]\
                                       for e in features_numerical.columns.tolist()[1:]])
```

- Merge

```
features_BUREAU = features_diff.merge(features_ratio, on='SK_ID_CURR',how='left')
features_BUREAU = features_BUREAU.merge(features_numerical, on='SK_ID_CURR',how='left')
features_BUREAU.to_csv('features_BUREAU_final.csv', index=False)
```

• previous_application.csv

previous_application

- Outliers and anomalies

```
# AMT_CREDIT==NaN
previous.drop(previous[previous['AMT_CREDIT'].isnull()].index, inplace=True)
previous.reset_index(drop=True, inplace=True)
# AMT_DOWN_PAYMENT < 0, RATE_DOWN_PAYMENT < 0
previous.loc[previous.AMT_DOWN_PAYMENT < 0, 'AMT_DOWN_PAYMENT'] = 0
previous.loc[previous.RATE_DOWN_PAYMENT < 0, 'RATE_DOWN_PAYMENT'] = 0
# SELLERPLACE_AREA < 0
previous['SELLERPLACE_AREA_ANOMALY'] = 0
anomalous_indices = previous[previous['SELLERPLACE_AREA'] < 0].index
previous.loc[anomalous_indices, 'SELLERPLACE_AREA_ANOMALY'] = 1
previous['SELLERPLACE_AREA'].replace(-1, 0, inplace=True)
# DAYS_ > 0
previous['DAYS_FIRST_DRAWING_ANOMALY'] = 0
previous['DAYS_FIRST_DUE_ANOMALY'] = 0
previous['DAYS_LAST_DUE_1ST_VERSION_ANOMALY'] = 0
previous['DAYS_LAST_DUE_ANOMALY'] = 0
previous['DAYS_TERMINATION_ANOMALY'] = 0
anomalous_indices = previous[previous.DAYS_FIRST_DRAWING > 0].index
previous.loc[anomalous_indices, 'DAYS_FIRST_DRAWING_ANOMALY'] = 1
previous['DAYS_FIRST_DRAWING'].replace(365243, 0, inplace=True)
anomalous_indices = previous[previous.DAYS_FIRST_DUE > 0].index
previous.loc[anomalous_indices, 'DAYS_FIRST_DUE_ANOMALY'] = 1
previous['DAYS_FIRST_DUE'].replace(365243, 0, inplace=True)
anomalous_indices = previous[previous.DAYS_LAST_DUE_1ST_VERSION > 0].index
previous.loc[anomalous_indices, 'DAYS_LAST_DUE_1ST_VERSION_ANOMALY'] = 1
previous['DAYS_LAST_DUE_1ST_VERSION'].replace(365243, 0, inplace=True)
anomalous_indices = previous[previous.DAYS_LAST_DUE > 0].index
previous.loc[anomalous_indices, 'DAYS_LAST_DUE_ANOMALY'] = 1
previous['DAYS_LAST_DUE'].replace(365243, 0, inplace=True)
anomalous_indices = previous[previous.DAYS_TERMINATION > 0].index
previous.loc[anomalous_indices, 'DAYS_TERMINATION_ANOMALY'] = 1
previous['DAYS_TERMINATION'].replace(365243, 0, inplace=True)
# XNA/XAP
for cat in cat_features:
    previous[cat].replace('XNA', np.nan, inplace=True)
    previous[cat].replace('XAP', np.nan, inplace=True)
```

- Hand-crafted features

```
# log transform
previous['log_SELLERPLACE_AREA'] = previous['SELLERPLACE_AREA'].map(lambda x: np.log(x + 1))
```

- Feature engineering

```
# diff features engineering
previous['diff_ANNUITY_APPLICATION'] = previous['AMT_ANNUITY'] - previous['AMT_APPLICATION']
previous['diff_ANNUITY_CREDIT'] = previous['AMT_ANNUITY'] - previous['AMT_CREDIT']
previous['diff_APPLICATION_CREDIT'] = previous['AMT_APPLICATION'] - previous['AMT_CREDIT']
previous['diff_PRICE_PAYMENT'] = previous['AMT_GOODS_PRICE'] - previous['AMT_DOWN_PAYMENT']
previous['diff_PRICE_CREDIT'] = previous['AMT_GOODS_PRICE'] - previous['AMT_CREDIT']
previous['diff_CREDIT_DOWN_PAYMENT'] = previous['AMT_CREDIT'] - previous['AMT_DOWN_PAYMENT']
previous['diff_RATE'] = previous['RATE_INTEREST_PRIMARY'] - previous['RATE_INTEREST_PRIVILEGED']
previous['diff_FIRST_DRAWING_DUE'] = previous['DAYS_FIRST_DRAWING'] - previous['DAYS_FIRST_DUE']
previous['diff_FIRST_DRAWING_LAST_DUE'] = previous['DAYS_FIRST_DRAWING'] - previous['DAYS_LAST_DUE_1ST_VERSION']
previous['diff_LAST_DUE'] = previous['DAYS_LAST_DUE_1ST_VERSION'] - previous['DAYS_LAST_DUE']
previous['diff_TERMINATION_DECISION'] = previous['DAYS_TERMINATION'] - previous['DAYS_DECISION']
previous['diff_TERMINATION_LAST_DUE'] = previous['DAYS_TERMINATION'] - previous['DAYS_LAST_DUE']
previous['diff_TERMINATION_LAST_DUE_1ST'] = previous['DAYS_TERMINATION'] - previous['DAYS_LAST_DUE_1ST_VERSION']

diff_features = ['diff_ANNUITY_APPLICATION', 'diff_ANNUITY_CREDIT', 'diff_APPLICATION_CREDIT',\
                 'diff_PRICE_PAYMENT', 'diff_PRICE_CREDIT', 'diff_CREDIT_DOWN_PAYMENT', 'diff_RATE',\
                 'diff_FIRST_DRAWING_DUE', 'diff_FIRST_DRAWING_LAST_DUE', 'diff_LAST_DUE',\
                 'diff_TERMINATION_DECISION', 'diff_TERMINATION_LAST_DUE', 'diff_TERMINATION_LAST_DUE_1ST']
features_diff = previous.groupby(['SK_ID_CURR'])[diff_features].agg({'max', 'min', 'mean'}).reset_index()
features_diff.columns = pd.Index(['SK_ID_CURR'] +\
                                 ["PREVIOUS_" + e[0] + "_" + e[1] for e in features_diff.columns.tolist()[1:]])
```

```
# ratio features engineering
previous['ratio_APPLICATION_ANNUITY'] = previous['AMT_APPLICATION'] / (1 + previous['AMT_ANNUITY'])
previous['ratio_CREDIT_APPLICATION'] = previous['AMT_CREDIT'] / (1 + previous['AMT_APPLICATION'])
previous['ratio_CREDIT_ANNUITY'] = previous['AMT_CREDIT'] / (1 + previous['AMT_ANNUITY'])
previous['ratio_DOWN_PAYMENT_PRICE'] = previous['AMT_DOWN_PAYMENT'] / (1 + previous['AMT_GOODS_PRICE'])
previous['ratio_PRICE_ANNUITY'] = previous['AMT_GOODS_PRICE'] / (1 + previous['AMT_ANNUITY'])
previous['ratio_DOWN_PAYMENT_ANNUITY'] = previous['AMT_DOWN_PAYMENT'] / (1 + previous['AMT_ANNUITY'])
previous['ratio_RATE'] = previous['RATE_INTEREST_PRIMARY'] / (1 + previous['RATE_INTEREST_PRIVILEGED'])
previous['ratio_DOWN_PAYMENT_RATE'] = previous['RATE_DOWN_PAYMENT'] / (1 + previous['AMT_DOWN_PAYMENT'])
previous['ratio_AMT_CREDIT_CNT'] = previous['AMT_CREDIT'] / (1 + previous['CNT_PAYMENT'])
previous['ratio_APPLICATION_log_AREA'] = previous['AMT_APPLICATION'] / (1 + previous['log_SELLERPLACE_AREA'])

ratio_features = ['ratio_APPLICATION_ANNUITY', 'ratio_CREDIT_APPLICATION', 'ratio_CREDIT_ANNUITY',\
                  'ratio_DOWN_PAYMENT_PRICE', 'ratio_PRICE_ANNUITY', 'ratio_DOWN_PAYMENT_ANNUITY',\
                  'ratio_RATE', 'ratio_DOWN_PAYMENT_RATE', 'ratio_AMT_CREDIT_CNT', 'ratio_APPLICATION_log_AREA']
features_ratio = previous.groupby(['SK_ID_CURR'])[ratio_features].agg({'max', 'min', 'mean', 'std'}).reset_index()
features_ratio.columns = pd.Index(['SK_ID_CURR'] +\
                                  ["PREVIOUS_" + e[0] + "_" + e[1] for e in features_ratio.columns.tolist()[1:]])
```

```
# multiply features engineering
previous['RATE_INTEREST_PRIMARY'].fillna(1.0, inplace=True)
previous['RATE_INTEREST_PRIVILEGED'].fillna(1.0, inplace=True)
previous['RATE_DOWN_PAYMENT'].fillna(0.0, inplace=True)
previous['AMT_ANNUITY'].fillna(0.0, inplace=True)
previous['AMT_DOWN_PAYMENT'].fillna(0.0, inplace=True)

previous['multiply_ANNUITY_RATE_PRIMARY'] = previous['AMT_ANNUITY'] * previous['RATE_INTEREST_PRIMARY']
previous['multiply_ANNUITY_RATE_PRIVILEGED'] = previous['AMT_ANNUITY'] * previous['RATE_INTEREST_PRIVILEGED']
previous['multiply_DOWN_PAYMENT_RATE'] = previous['AMT_DOWN_PAYMENT'] * previous['RATE_DOWN_PAYMENT']

features_multiply = previous.groupby(['SK_ID_CURR'])\
['multiply_ANNUITY_RATE_PRIMARY', 'multiply_ANNUITY_RATE_PRIVILEGED', 'multiply_DOWN_PAYMENT_RATE'].\
agg({'max', 'min', 'mean', 'std'}).reset_index()
features_multiply.columns = pd.Index(['SK_ID_CURR'] +\
                                  ["PREVIOUS_" + e[0] + "_" + e[1] for e in features_multiply.columns.tolist()[1:]])
```

```
# num_avg features engineering
previous['YEARS_LAST_DUE_1ST_VERSION'] = previous['DAYS_LAST_DUE_1ST_VERSION']/365
previous['YEARS_DECISION'] = previous['DAYS_DECISION']/365

num_avg_features = ['AMT_ANNUITY', 'YEARS_LAST_DUE_1ST_VERSION', 'AMT_DOWN_PAYMENT',\
                    'HOUR_APPR_PROCESS_START', 'YEARS_DECISION', 'log_SELLERPLACE_AREA']

features_avg = previous.groupby(['SK_ID_CURR'])[num_avg_features].agg({'mean'}).reset_index()
features_avg.columns = pd.Index(['SK_ID_CURR'] +\
                                ["PREVIOUS_" + e[0] + "_" + e[1] for e in features_avg.columns.tolist()[1:]])
```

- Merge

```
features_previous = features_diff.merge(features_ratio)
features_previous = features_previous.merge(features_multiply)
features_previous = features_previous.merge(features_avg)
features_previous.to_csv('features_previous_final.csv', index=False)
```

• credit_card_balance.csv

credit_card_balance

- Outliers and anomalies

```
credit.fillna(0, inplace=True)
credit['AMT_DRAWINGS_ATM_CURRENT'].replace(-6827.31, 0, inplace=True)
credit['AMT_DRAWINGS_CURRENT'].replace(-519.57, 0, inplace=True)
credit['AMT_DRAWINGS_CURRENT'].replace(-1687.50, 0, inplace=True)
credit['AMT_DRAWINGS_CURRENT'].replace(-6211.62, 0, inplace=True)
```

- Feature engineering

```
# Drawings features engineering
credit['AMT_CNT_ratio_ATM_CURRENT'] = credit['AMT_DRAWINGS_ATM_CURRENT'] / (1 + credit['CNT_DRAWINGS_ATM_CURRENT'])
credit['AMT_CNT_ratio_CURRENT'] = credit['AMT_DRAWINGS_CURRENT'] / (1 + credit['CNT_DRAWINGS_CURRENT'])
credit['AMT_CNT_ratio_OTHER_CURRENT']=credit['AMT_DRAWINGS_OTHER_CURRENT']/(1 + credit['CNT_DRAWINGS_OTHER_CURRENT'])
credit['AMT_CNT_ratio_POS_CURRENT'] = credit['AMT_DRAWINGS_POS_CURRENT'] / (1 + credit['CNT_DRAWINGS_POS_CURRENT'])

features_drawings = credit.groupby(['SK_ID_CURR', 'SK_ID_PREV'])\
['AMT_DRAWINGS_ATM_CURRENT', 'CNT_DRAWINGS_ATM_CURRENT', 'AMT_DRAWINGS_CURRENT', 'CNT_DRAWINGS_CURRENT',\
'AMT_DRAWINGS_OTHER_CURRENT', 'CNT_DRAWINGS_OTHER_CURRENT', 'AMT_DRAWINGS_POS_CURRENT', 'CNT_DRAWINGS_POS_CURRENT',\
'AMT_CNT_ratio_POS_CURRENT', 'AMT_CNT_ratio_ATM_CURRENT', 'AMT_CNT_ratio_CURRENT', 'AMT_CNT_ratio_OTHER_CURRENT'].\
agg({'max', 'min', 'mean', 'std'}).reset_index()
features_drawings.columns = pd.Index(['SK_ID_CURR', 'SK_ID_PREV'] +\
                                  ["CREDIT_" + e[0] + "_" + e[1] for e in features_drawings.columns.tolist()[2:]])
```

```
# Time-related features engineering
features_time = credit.groupby(['SK_ID_CURR', 'SK_ID_PREV'])['MONTHS_BALANCE'].\
agg({'max', 'min', 'mean', 'size'}).reset_index().\
rename(columns={'max': 'CREDIT_month_bal_max', 'min': 'CREDIT_month_bal_min',
                'mean': 'CREDIT_month_bal_mean', 'size': 'CREDIT_Duration'})

features_DPD = credit.groupby(['SK_ID_CURR', 'SK_ID_PREV'])['SK_DPD', 'SK_DPD_DEF'].\
agg({'max', 'min', 'mean'}).reset_index()
features_DPD.columns = pd.Index(['SK_ID_CURR', 'SK_ID_PREV'] +\
                                ["CREDIT_" + e[0] + "_" + e[1] for e in features_DPD.columns.tolist()[2:]])
```

```
# diff features engineering
credit['diff_PAYMENT_REGULARITY'] = credit['AMT_PAYMENT_TOTAL_CURRENT'] - credit['AMT_INST_MIN_REGULARITY']
credit['diff_RECEIVABLE'] = credit['AMT_RECEIVABLE_PRINCIPAL'] - credit['AMT_RECIVABLE']

features_diff = credit.groupby(['SK_ID_CURR', 'SK_ID_PREV'])['diff_PAYMENT_REGULARITY', 'diff_RECEIVABLE'].\
agg({'max', 'min', 'mean'}).reset_index()
features_diff.columns = pd.Index(['SK_ID_CURR', 'SK_ID_PREV'] +\
                                 ["CREDIT_" + e[0] + "_" + e[1] for e in features_diff.columns.tolist()[2:]])
```

```
# ratio features engineering
credit['AMT_ratio_PAYMENT_CURRENT'] = credit['AMT_PAYMENT_CURRENT'] / (1 + credit['AMT_PAYMENT_TOTAL_CURRENT'])
credit['AMT_ratio_RECIVABLE'] = credit['AMT_RECIVABLE'] / (1 + credit['AMT_TOTAL_RECEIVABLE'])
credit['ratio_BALANCE_LIMIT'] = credit['MONTHS_BALANCE'] / (1 + credit['AMT_CREDIT_LIMIT_ACTUAL'])

features_ratio = credit.groupby(['SK_ID_CURR', 'SK_ID_PREV'])\
['AMT_ratio_PAYMENT_CURRENT', 'AMT_ratio_RECIVABLE', 'ratio_BALANCE_LIMIT'].\
agg({'max', 'min', 'mean', 'std'}).reset_index()
features_ratio.columns = pd.Index(['SK_ID_CURR', 'SK_ID_PREV'] +\
                                  ["CREDIT_" + e[0] + "_" + e[1] for e in features_ratio.columns.tolist()[2:]])
```

```
# CUM features engineering
features_CUM = credit.groupby(['SK_ID_CURR', 'SK_ID_PREV'])['CNT_INSTALMENT_MATURE_CUM'].\
agg({'max', 'min', 'mean'}).reset_index().\
rename(columns={'max': 'CREDIT_CNT_INSTALMENT_MATURE_CUM_max', 'min': 'CREDIT_CNT_INSTALMENT_MATURE_CUM_min',
                'mean': 'CREDIT_CNT_INSTALMENT_MATURE_CUM_mean'})
```

- Merge

```
features_credit = features_drawings.merge(features_time)
features_credit = features_credit.merge(features_DPD)
features_credit = features_credit.merge(features_diff)
features_credit = features_credit.merge(features_ratio)
features_credit = features_credit.merge(features_CUM)

features_credit.to_csv('features_credit_final.csv', index=False)
```

• installments_payments.csv

installments_payments

- Feature engineering

```
# diff features engineering
install['diff_early_pay'] = install['DAYS_INSTALMENT'] - install['DAYS_ENTRY_PAYMENT']
install['diff_AMT_pay'] = install['AMT_INSTALMENT'] - install['AMT_PAYMENT']

features_diff = install.groupby(['SK_ID_CURR', 'SK_ID_PREV'])['diff_early_pay', 'diff_AMT_pay'].\
agg({'max', 'min', 'mean'}).reset_index()
features_diff.columns = pd.Index(['SK_ID_CURR', 'SK_ID_PREV'] +\
                                 ["INSTALL_" + e[0] + "_" + e[1] for e in features_diff.columns.tolist()[2:]])

# ratio features engineering
install['ratio_early_pay'] = install['diff_early_pay'] / (1 + install['DAYS_ENTRY_PAYMENT'])
install['ratio_AMT_pay'] = install['diff_AMT_pay'] / (1 + install['AMT_INSTALMENT'])

features_ratio = install.groupby(['SK_ID_CURR', 'SK_ID_PREV'])['ratio_early_pay', 'ratio_AMT_pay'].\
agg({'max', 'min', 'mean', 'std'}).reset_index()
features_ratio.columns = pd.Index(['SK_ID_CURR', 'SK_ID_PREV'] +\
                                  ["INSTALL_" + e[0] + "_" + e[1] for e in features_ratio.columns.tolist()[2:]])
```

- Merge

```
features_install = features_diff.merge(features_ratio)
features_install.to_csv('features_install_final.csv', index=False)
```

• POS_CASH_balance.csv

POS_CASH_balance

- Feature engineering

```
# diff features engineering
pos_cash['DPD_diff'] = pos_cash['SK_DPD'] - pos_cash['SK_DPD_DEF']
pos_cash['CNT_diff'] = pos_cash['CNT_INSTALMENT'] - pos_cash['CNT_INSTALMENT_FUTURE']

features_diff = pos_cash.groupby(['SK_ID_CURR', 'SK_ID_PREV'])['DPD_diff', 'CNT_diff'].\
agg({'max', 'min', 'mean'}).reset_index()
features_diff.columns = pd.Index(['SK_ID_CURR', 'SK_ID_PREV'] +\
                                 [e[0] + "_" + e[1] for e in features_diff.columns.tolist()[2:]])

# time features engineering
features_time = pos_cash.groupby(['SK_ID_CURR', 'SK_ID_PREV'])['MONTHS_BALANCE'].\
agg({'max', 'min', 'mean', 'size'}).reset_index().\
rename(columns={'max': 'month_bal_max', 'min': 'month_bal_min', 'mean': 'month_bal_mean', 'size': 'Duration'})

# DPD features engineering
features_DPD = pos_cash.groupby(['SK_ID_CURR', 'SK_ID_PREV'])['SK_DPD', 'SK_DPD_DEF'].\
agg({'max', 'min', 'mean'}).reset_index()
features_DPD.columns = pd.Index(['SK_ID_CURR', 'SK_ID_PREV'] +\
                                [e[0] + "_" + e[1] for e in features_DPD.columns.tolist()[2:]])

# CNT features engineering
features_CNT = pos_cash.groupby(['SK_ID_CURR', 'SK_ID_PREV'])['CNT_INSTALMENT', 'CNT_INSTALMENT_FUTURE'].\
agg({'max', 'min', 'mean'}).reset_index()
features_CNT.columns = pd.Index(['SK_ID_CURR', 'SK_ID_PREV'] +\
                                [e[0] + "_" + e[1] for e in features_CNT.columns.tolist()[2:]])
```

- Merge

```
features_POS = features_time.merge(features_DPD)
features_POS = features_POS.merge(features_diff)
features_POS = features_POS.merge(features_CNT)

features_POS.to_csv('features_POS_final.csv', index=False)
```

• application_train.csv and appication_test.csv

Categorical

- Anomaly

```
application['CODE_GENDER'].replace('XNA', 'F', inplace=True)
```

- Target-encoding

```
cat_features = [col for col in application_train_.columns if application_train_[col].dtype == 'object']
target_encoding = dict.fromkeys(cat_features, 0)
for feature in cat_features:
    cat_ = application[feature].dropna().unique()
    default_percentage = []
    for k, v in enumerate(cat_):
        repay = application[(application[feature] == v) & (application.TARGET == 0)].shape[0]
        default = application[(application[feature] == v) & (application.TARGET == 1)].shape[0]
        default_percentage.append(100 * (default/(repay + default)))
    feature_map = {key:value for (key, value) in zip(cat_, default_percentage)}
    target_encoding[feature] = feature_map
for feat in target_encoding:
    application[feat].replace(target_encoding[feat], inplace=True)
```

Numerical

- Anomaly

```
application['REGION_RATING_CLIENT_W_CITY'].replace(-1, 2, inplace=True)
application.loc[application.AMT_INCOME_TOTAL == 117000000, 'AMT_INCOME_TOTAL'] = 117000
application.loc[application.AMT_REQ_CREDIT_BUREAU_QRT > 100, 'AMT_REQ_CREDIT_BUREAU_QRT'] = np.nan
application.loc[application.OBS_30_CNT_SOCIAL_CIRCLE > 300, 'OBS_30_CNT_SOCIAL_CIRCLE'] = np.nan
application.loc[application.OBS_60_CNT_SOCIAL_CIRCLE > 300, 'OBS_60_CNT_SOCIAL_CIRCLE'] = np.nan
# DAYS_EMPLOYED > 0
# https://www.kaggle.com/c/home-credit-default-risk/discussion/57247
application["DAYS_EMPLOYED"].replace(365243, 0, inplace=True)
# missing value
application['NONLIVINGAPARTMENTS_MODE'].fillna(value=1, inplace=True)
house_features_AVG = [col for col in application.columns if col.find('AVG')!= -1]
house_features_MODE = [col for col in application.columns if col.find('MODE')!= -1]
house_features_MEDI = [col for col in application.columns if col.find('MEDI')!= -1]
house_features = house_features_AVG + house_features_MODE + house_features_MEDI
for _ in house_features:
    application[_].fillna(0.0, inplace=True)
```

- Hand-crafted features

```
application["AGE"] = application["DAYS_BIRTH"].abs()//365
application['AMT_INCOME_TOTAL_K'] = application['AMT_INCOME_TOTAL']//1000
application['Long_employment'] = (application['DAYS_EMPLOYED'] < (-365)*5).astype(int)
application['Age_38up'] = (application['DAYS_BIRTH'] < (-365*38)).astype(int)
```

- Feature engineering

```
# Ratio feature engineering
application['ratio_annuity_income'] = application['AMT_ANNUITY'] / application['AMT_INCOME_TOTAL']
application['ratio_car_to_birth'] = application['OWN_CAR_AGE'] / application['DAYS_BIRTH']
application['ratio_car_to_employ'] = application['OWN_CAR_AGE'] / application['DAYS_EMPLOYED']
application['ratio_children'] = application['CNT_CHILDREN'] / application['CNT_FAM_MEMBERS']
application['ratio_credit_to_annuity'] = application['AMT_CREDIT'] / application['AMT_ANNUITY']
application['ratio_credit_to_goods'] = application['AMT_CREDIT'] / application['AMT_GOODS_PRICE']
application['ratio_credit_to_income'] = application['AMT_CREDIT'] / application['AMT_INCOME_TOTAL']
application['ratio_days_employed'] = application['DAYS_EMPLOYED'] / application['DAYS_BIRTH']
application['ratio_income_credit'] = application['AMT_INCOME_TOTAL'] / application['AMT_CREDIT']
application['ratio_income_per_child'] = application['AMT_INCOME_TOTAL'] / (1 + application['CNT_CHILDREN'])
application['ratio_income_per_person'] = application['AMT_INCOME_TOTAL'] / application['CNT_FAM_MEMBERS']
application['ratio_payment_rate'] = application['AMT_ANNUITY'] / application['AMT_CREDIT']
application['ratio_phone_to_birth'] = application['DAYS_LAST_PHONE_CHANGE'] / application['DAYS_BIRTH']
application['ratio_phone_to_employ'] = application['DAYS_LAST_PHONE_CHANGE'] / application['DAYS_EMPLOYED']
```

```
# Aggregation feature engineering
application['external_sources_weighted'] =\
application.EXT_SOURCE_1 * 2 + application.EXT_SOURCE_2 * 3 + application.EXT_SOURCE_3 * 4

for function_name in ['min', 'max', 'sum', 'mean', 'nanmedian']:
    application['external_sources_{}'.format(function_name)] = eval('np.{}'.format(function_name))(
        application[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3']], axis=1)

AGGREGATION_RECIPIES = [
    (['CODE_GENDER', 'NAME_EDUCATION_TYPE'],
     [('AMT_ANNUITY', 'max'), ('AMT_CREDIT', 'max'),
      ('EXT_SOURCE_1', 'mean'), ('EXT_SOURCE_2', 'mean'),
      ('OWN_CAR_AGE', 'max'), ('OWN_CAR_AGE', 'sum')]),
    (['CODE_GENDER', 'ORGANIZATION_TYPE'],
     [('AMT_ANNUITY', 'mean'), ('AMT_INCOME_TOTAL', 'mean'),
      ('DAYS_REGISTRATION', 'mean'), ('EXT_SOURCE_1', 'mean')]),
    (['CODE_GENDER', 'REG_CITY_NOT_WORK_CITY'],
     [('AMT_ANNUITY', 'mean'), ('CNT_CHILDREN', 'mean'), ('DAYS_ID_PUBLISH', 'mean')]),
    (['CODE_GENDER', 'NAME_EDUCATION_TYPE', 'OCCUPATION_TYPE', 'REG_CITY_NOT_WORK_CITY'],
     [('EXT_SOURCE_1', 'mean'), ('EXT_SOURCE_2', 'mean')]),

    (['NAME_EDUCATION_TYPE', 'OCCUPATION_TYPE'],
     [('AMT_CREDIT', 'mean'), ('AMT_REQ_CREDIT_BUREAU_YEAR', 'mean'),
      ('APARTMENTS_AVG', 'mean'), ('BASEMENTAREA_AVG', 'mean'),
      ('EXT_SOURCE_1', 'mean'), ('EXT_SOURCE_2', 'mean'), ('EXT_SOURCE_3', 'mean'),
      ('NONLIVINGAREA_AVG', 'mean'), ('OWN_CAR_AGE', 'mean'), ('YEARS_BUILD_AVG', 'mean')]),
    (['NAME_EDUCATION_TYPE', 'OCCUPATION_TYPE', 'REG_CITY_NOT_WORK_CITY'],
     [('ELEVATORS_AVG', 'mean'), ('EXT_SOURCE_1', 'mean')]),
    (['OCCUPATION_TYPE'],
     [('AMT_ANNUITY', 'mean'), ('CNT_CHILDREN', 'mean'), ('CNT_FAM_MEMBERS', 'mean'),
      ('DAYS_BIRTH', 'mean'), ('DAYS_EMPLOYED', 'mean'),
      ('DAYS_ID_PUBLISH', 'mean'), ('DAYS_REGISTRATION', 'mean'),
      ('EXT_SOURCE_1', 'mean'), ('EXT_SOURCE_2', 'mean'), ('EXT_SOURCE_3', 'mean')])]

groupby_aggregate_names = []
for groupby_cols, specs in (AGGREGATION_RECIPIES):
    group_object = application.groupby(groupby_cols)
    for select, agg in (specs):
        groupby_aggregate_name = '{}_{}_{}'.format('_'.join(groupby_cols), agg, select)
        application = application.merge(group_object[select].agg(agg).reset_index()
                                .rename(index=str, columns={select: groupby_aggregate_name})
                                [groupby_cols + [groupby_aggregate_name]], on=groupby_cols,how='left')
        groupby_aggregate_names.append(groupby_aggregate_name)
```

```
# Diff feature engineering
diff_feature_names = []
for groupby_cols, specs in (AGGREGATION_RECIPIES):
    for select, agg in (specs):
        if agg in ['mean','median','max','min']:
            groupby_aggregate_name = '{}_{}_{}'.format('_'.join(groupby_cols), agg, select)
            diff_name = '{}_diff'.format(groupby_aggregate_name)
            abs_diff_name = '{}_abs_diff'.format(groupby_aggregate_name)
            application[diff_name] = application[select] - application[groupby_aggregate_name]
            application[abs_diff_name] = np.abs(application[select] - application[groupby_aggregate_name])
            diff_feature_names.append(diff_name)
            diff_feature_names.append(abs_diff_name)
```

• Combine all relational databases and save as new data/test files

- Load application_train and applocation_test(after preprocessing)

```
application_train_ = pd.read_csv('./application_train_final.csv')
application_test_ = pd.read_csv('./application_test_final.csv')
train_size_ = application_train_.shape[0]
application = pd.concat((application_train_, application_test_)).reset_index(drop=True)
```

- Merge bureau and bureau_balance(after preprocessing)

```
features_bureau = pd.read_csv('features_BUREAU_final.csv')
application = application.merge(features_bureau, on='SK_ID_CURR',how='left')

features_previous = pd.read_csv('features_previous_final.csv')
features_previous_CURR = features_previous.groupby(['SK_ID_CURR']).\
agg({'max', 'min', 'mean', 'std'}).reset_index()
features_previous_CURR.columns = pd.Index(['SK_ID_CURR'] +\
                                          [e[0] + "_" + e[1] for e in features_previous_CURR.columns.tolist()[1:]])
application = application.merge(features_previous_CURR, on='SK_ID_CURR',how='left')
```

- Merge POS_CASH_balance(after preprocessing)

```
features_POS = pd.read_csv('features_POS_final.csv')
features_POS_CURR = features_POS.drop(columns = ['SK_ID_PREV'])
features_POS_CURR = features_POS_CURR.groupby(['SK_ID_CURR']).\
agg({'max', 'min', 'mean', 'std'}).reset_index()
features_POS_CURR.columns = pd.Index(['SK_ID_CURR'] +\
                                     [e[0] + "_" + e[1] for e in features_POS_CURR.columns.tolist()[1:]])
application = application.merge(features_POS_CURR, on='SK_ID_CURR',how='left')
```

- Merge installments_payments(after preprocessing)

```
features_install = pd.read_csv('features_install_final.csv')
features_install_CURR = features_install.drop(columns = ['SK_ID_PREV'])
features_install_CURR = features_install_CURR.groupby(['SK_ID_CURR']).\
agg({'max', 'min', 'mean', 'std'}).reset_index()
features_install_CURR.columns = pd.Index(['SK_ID_CURR'] +\
                                         [e[0] + "_" + e[1] for e in features_install_CURR.columns.tolist()[1:]])
application = application.merge(features_install_CURR, on='SK_ID_CURR',how='left')
```

- Merge credit_card_balance(after preprocessing)

```
features_credit = pd.read_csv('features_credit_final.csv')
features_credit_CURR = features_credit.drop(columns = ['SK_ID_PREV'])
features_credit_CURR = features_credit_CURR.groupby(['SK_ID_CURR']).\
agg({'max', 'min', 'mean', 'std'}).reset_index()
features_credit_CURR.columns = pd.Index(['SK_ID_CURR'] +\
                                        [e[0] + "_" + e[1] for e in features_credit_CURR.columns.tolist()[1:]])
application = application.merge(features_credit_CURR, on='SK_ID_CURR',how='left')
```

- Save as new files

```
data = application[:train_size_].copy()
test = application[train_size_:].copy()
test.drop(columns = ['TARGET'], inplace=True)

data.to_csv('data_fe.csv', index = False)
test.to_csv('test_fe.csv', index = False)
```

Implementation

After preprocessed the datasets, we will train three different models to get more model diversity, that is 'LGBMClassifier', 'CatBoostClassifier', and 'XGBClassifier'. For each model training, we use KFold to split into training and validation subsets to avoid overfitting. We then stack these predictions as features of the secondary model for final prediction. Each prediction of the Level 1 and Level 2 models will be saved.

- Load datasets

```
data = pd.read_csv('./data_fe.csv')
test = pd.read_csv('./test_fe.csv')

y_train = data['TARGET']
X_train = data.drop(columns = ['SK_ID_CURR', 'TARGET'])
X_test = test.drop(columns = ['SK_ID_CURR'])

submission_lgbm = test[['SK_ID_CURR']].copy()
submission_catb = test[['SK_ID_CURR']].copy()
submission_xgbt = test[['SK_ID_CURR']].copy()
oof_lgbm = data[['SK_ID_CURR', 'TARGET']].copy()
oof_catb = data[['SK_ID_CURR', 'TARGET']].copy()
oof_xgbt = data[['SK_ID_CURR', 'TARGET']].copy()
```

- Train and predict(1st level): CatBoostClassifier

```
oof_pred = np.zeros(X_train.shape[0])
y_pred = np.zeros(X_test.shape[0])
folds = KFold(n_splits= 5, shuffle=True, random_state=RANDOM)

# missing values
imputer = Imputer(strategy='median').fit(X_train)
X_train_ = imputer.transform(X_train)
X_test_ = imputer.transform(X_test)
X_train_ = pd.DataFrame(X_train_)
X_test_ = pd.DataFrame(X_test_)

for n_fold, (train_idx, valid_idx) in enumerate(folds.split(X_train_, y_train)):
    size = train_idx.size
    oversample_idx = np.random.choice(size, size//10)
    train_idx = np.append(train_idx, oversample_idx)

    train_x, train_y = X_train_.iloc[train_idx], y_train.iloc[train_idx]
    valid_x, valid_y = X_train_.iloc[valid_idx], y_train.iloc[valid_idx]

    clf = CatBoostClassifier(eval_metric='AUC', use_best_model=True)

    clf.fit(train_x, train_y, eval_set=[(train_x, train_y), (valid_x, valid_y)],\
            verbose= 100, early_stopping_rounds= 50)
    oof_pred[valid_idx] = clf.predict_proba(valid_x)[:, 1]
    y_pred += clf.predict_proba(X_test_)[:, 1] / folds.n_splits
    print('Fold %2d AUC : %.6f' % (n_fold + 1, roc_auc_score(valid_y, oof_pred[valid_idx])))
print('Full AUC score %.6f' % roc_auc_score(y_train, oof_pred))

submission_catb['TARGET'] = y_pred
submission_catb.to_csv('submission_catb.csv', index = False)

oof_catb['PRED'] = oof_pred
oof_catb.to_csv('oof_catb.csv', index = False)
```

- Train and predict(1st level): XGBClassifier

```
oof_pred = np.zeros(X_train.shape[0])
y_pred = np.zeros(X_test.shape[0])
folds = KFold(n_splits=5, shuffle=True, random_state=RANDOM)
for n_fold, (train_idx, valid_idx) in enumerate(folds.split(X_train, y_train)):
    size = train_idx.size
    oversample_idx = np.random.choice(size, size//10)
    train_idx = np.append(train_idx, oversample_idx)

    train_x, train_y = X_train.iloc[train_idx], y_train.iloc[train_idx]
    valid_x, valid_y = X_train.iloc[valid_idx], y_train.iloc[valid_idx]

    clf = XGBClassifier()

    clf.fit(train_x, train_y, eval_set=[(train_x, train_y), (valid_x, valid_y)],\
            eval_metric='auc', verbose=100, early_stopping_rounds=50)
    oof_pred[valid_idx] = clf.predict_proba(valid_x)[:, 1]
    y_pred += clf.predict_proba(X_test)[:, 1] / folds.n_splits
    print('Fold %2d AUC : %.6f' % (n_fold + 1, roc_auc_score(valid_y, oof_pred[valid_idx])))
print('Full AUC score %.6f' % roc_auc_score(y_train, oof_pred))

submission_xgbt['TARGET'] = y_pred
submission_xgbt.to_csv('submission_xgbt.csv', index=False)

oof_xgbt['PRED'] = oof_pred
oof_xgbt.to_csv('oof_xgbt.csv', index=False)
```

- Train and predict(1st level): LGBMClassifier

```
oof_pred = np.zeros(X_train.shape[0])
y_pred = np.zeros(X_test.shape[0])
folds = KFold(n_splits= 5, shuffle=True, random_state=RANDOM)
for n_fold, (train_idx, valid_idx) in enumerate(folds.split(X_train, y_train)):
    size = train_idx.size
    oversample_idx = np.random.choice(size, size//10)
    train_idx = np.append(train_idx, oversample_idx)

    train_x, train_y = X_train.iloc[train_idx], y_train.iloc[train_idx]
    valid_x, valid_y = X_train.iloc[valid_idx], y_train.iloc[valid_idx]

    clf = LGBMClassifier(nthread=-1)
    clf.fit(train_x, train_y, eval_set=[(train_x, train_y), (valid_x, valid_y)],\
            eval_metric= 'auc', verbose= 100, early_stopping_rounds= 50)
    oof_pred[valid_idx] = clf.predict_proba(valid_x, num_iteration=clf.best_iteration_)[:, 1]
    y_pred += clf.predict_proba(X_test, num_iteration=clf.best_iteration_)[:, 1] / folds.n_splits
    print('Fold %2d AUC : %.6f' % (n_fold + 1, roc_auc_score(valid_y, oof_pred[valid_idx])))
print('Full AUC score %.6f' % roc_auc_score(y_train, oof_pred))

submission_lgbm['TARGET'] = y_pred
submission_lgbm.to_csv('submission_lgbm.csv', index = False)

oof_lgbm['PRED'] = oof_pred
oof_lgbm.to_csv('oof_lgbm.csv', index = False)
```

- Stack first-level predictions and create a new dataset for second level

```
data1 = oof_lgbm.copy()
data2 = oof_catb.drop(columns = ['TARGET'])
data3 = oof_xgbt.drop(columns = ['TARGET'])
data1.rename(columns={'PRED':'PRED_lgbm'}, inplace=True)
data2.rename(columns={'PRED':'PRED_catb'}, inplace=True)
data3.rename(columns={'PRED':'PRED_xgbt'}, inplace=True)
data = data1.join(data2.set_index('SK_ID_CURR'), on='SK_ID_CURR')
data = data.join(data3.set_index('SK_ID_CURR'), on='SK_ID_CURR')

test = pd.DataFrame()
test['SK_ID_CURR'] = submission_lgbm['SK_ID_CURR'].copy()
test['PRED_lgbm'] = submission_lgbm['TARGET']
test['PRED_catb'] = submission_catb['TARGET']
test['PRED_xgbt'] = submission_xgbt['TARGET']

y_train = data['TARGET'].copy()
X_train = data.drop(columns = ['SK_ID_CURR', 'TARGET'])
submission = test[['SK_ID_CURR']].copy()
X_test = test.drop(columns = ['SK_ID_CURR'])
```

- Train and predict(2nd level): LGBMClassifier

```
oof_pred = np.zeros(X_train.shape[0])
y_pred = np.zeros(X_test.shape[0])
folds = KFold(n_splits= 5, shuffle=True, random_state=RANDOM)
for n_fold, (train_idx, valid_idx) in enumerate(folds.split(X_train, y_train)):
    train_x, train_y = X_train.iloc[train_idx], y_train.iloc[train_idx]
    valid_x, valid_y = X_train.iloc[valid_idx], y_train.iloc[valid_idx]

    clf = LGBMClassifier(nthread=-1)

    clf.fit(train_x, train_y, eval_set=[(train_x, train_y), (valid_x, valid_y)],\
            eval_metric= 'auc', verbose= 100, early_stopping_rounds= 50)
    oof_pred[valid_idx] = clf.predict_proba(valid_x)[:, 1]
    y_pred += clf.predict_proba(X_test)[:, 1] / folds.n_splits
    print('Fold %2d AUC : %.6f' % (n_fold + 1, roc_auc_score(valid_y, oof_pred[valid_idx])))
print('Full AUC score %.6f' % roc_auc_score(y_train, oof_pred))
submission['TARGET'] = y_pred
print(submission.head())
submission.to_csv('submission_stack.csv', index = False)
```

Refinement

There are several ways to optimize hyperparamters, including GridSearch, RandomSearch, and Bayesian optimization. For convenience, we use GridSearch to perform hyperparameter optimization. As listed below, model improvements ranged from 0.005 to 0.031 AUC scores with 5-Fold Cross-validation.

Model	AUC score	
	with default hyperparameters	with optimized hyperparameters
LGBMClassifier	Fold 1 AUC : 0.789852 Fold 2 AUC : 0.787760 Fold 3 AUC : 0.787868 Fold 4 AUC : 0.791548 Fold 5 AUC : 0.787991 **Full AUC score : 0.788996**	Fold 1 AUC : 0.798936 Fold 2 AUC : 0.800173 Fold 3 AUC : 0.794746 Fold 4 AUC : 0.802849 Fold 5 AUC : 0.800740 **Full AUC score : 0.799491**
CatBoostClassifier	Fold 1 AUC : 0.789434 Fold 2 AUC : 0.792968 Fold 3 AUC : 0.787340 Fold 4 AUC : 0.792176 Fold 5 AUC : 0.793205 **Full AUC score : 0.791025**	Fold 1 AUC : 0.797625 Fold 2 AUC : 0.795580 Fold 3 AUC : 0.791579 Fold 4 AUC : 0.797508 Fold 5 AUC : 0.797496 **Full AUC score : 0.795955**
XGBClassifier	Fold 1 AUC : 0.776255 Fold 2 AUC : 0.776244 Fold 3 AUC : 0.775361 Fold 4 AUC : 0.778546 Fold 5 AUC : 0.775569 **Full AUC score : 0.776381**	Fold 1 AUC : 0.808895 Fold 2 AUC : 0.807873 Fold 3 AUC : 0.803194 Fold 4 AUC : 0.807418 Fold 5 AUC : 0.809910 **Full AUC score : 0.807435**

- LGBMClassifier GridSearch

```
data = pd.read_csv('./data_fe.csv', nrows=6000)
test = pd.read_csv('./test_fe.csv', nrows=1000)
y_train = data['TARGET']
X_train = data.drop(columns = ['SK_ID_CURR', 'TARGET'])
X_test = test.drop(columns = ['SK_ID_CURR'])

# Initialize the classifier
clf = LGBMClassifier(nthread=-1, learning_rate=0.02, silent=-1, verbose=-1)

parameters = {'num_leaves':[16, 32, 64],
              'max_depth':[7, 8, 9],
              'min_child_weight':[30, 40, 50],
              'n_estimators':[3000, 10000],
              'learning_rate': [0.02, 0.1],
              'reg_alpha': [0.04, 0.08],
              'subsample':[0.8, 0.9, 1.0],
              'min_split_gain':[0.01, 0.02, 0.03]}

scorer = make_scorer(roc_auc_score)
grid_obj = GridSearchCV(clf, parameters, scoring=scorer, n_jobs = -1, cv = 4)
grid_fit = grid_obj.fit(X_train, y_train)
best_clf = grid_fit.best_estimator_
```

- CatBoostClassifier GridSearch

```
data = pd.read_csv('./data_fe.csv', nrows=6000)
test = pd.read_csv('./test_fe.csv', nrows=1000)
y_train = data['TARGET']
X_train = data.drop(columns = ['SK_ID_CURR', 'TARGET'])
X_test = test.drop(columns = ['SK_ID_CURR'])

# Initialize the classifier
clf = CatBoostClassifier()

parameters = {'l2_leaf_reg':[3, 5, 10],
              'depth':[7, 8, 9],
              'border_count':[10, 20, 50]}

scorer = make_scorer(roc_auc_score)
grid_obj = GridSearchCV(clf, parameters, scoring=scorer, n_jobs = -1, cv = 4)
grid_fit = grid_obj.fit(X_train, y_train)
best_clf = grid_fit.best_estimator_
```

- XGBClassifier GridSearch

```
data = pd.read_csv('./data_fe.csv', nrows=6000)
test = pd.read_csv('./test_fe.csv', nrows=1000)
y_train = data['TARGET']
X_train = data.drop(columns = ['SK_ID_CURR', 'TARGET'])
X_test = test.drop(columns = ['SK_ID_CURR'])

# Initialize the classifier
clf = XGBClassifier(nthread=-1)

parameters = {'colsample_bytree':[0.6, 0.7, 0.8],
              'subsample':[0.6, 0.7, 0.85],
              'n_estimators': [1000, 1500, 2000],
              'max_depth': [7, 8, 9],
              'learning_rate': [0.02, 0.06, 0.1]}

scorer = make_scorer(roc_auc_score)
grid_obj = GridSearchCV(clf, parameters, scoring=scorer, n_jobs = -1, cv = 4)
grid_fit = grid_obj.fit(X_train, y_train)
best_clf = grid_fit.best_estimator_
```

Results

Model Evaluation and Validation

- Optimizied hyperparameters' setting(LGBMClassifier)
  - num_learves: Maximum tree leaves for base learners, 16
  - subsample: Subsample ratio of the training instance, 0.8
  - min_split_gain: Minimum loss reduction required to make a further partition on a leaf node of the tree, 0.01

[ref] https://github.com/Microsoft/LightGBM/blob/master/python-package/lightgbm/sklearn.py

```
LGBMClassifier(boosting_type='gbdt', class_weight=None, colsample_bytree=1.0,
               learning_rate=0.02, max_depth=-1, min_child_samples=20,
               min_child_weight=40, min_split_gain=0.01, n_estimators=10000,
               n_jobs=-1, nthread=-1, num_leaves=16, objective=None,
               random_state=None, reg_alpha=0.08, reg_lambda=0.0, silent=-1,
               subsample=0.8, subsample_for_bin=200000, subsample_freq=0, verbose=-1)
```

- Optimizied hyperparameters' setting(CatBoostClassifier)
  - border_count: The number of splits for numerical features, 20
  - l2_leaf_reg: L2 regularization coefficient. Used for leaf value calculation, 5
  - depth: Depth of the tree, 7

[ref] https://tech.yandex.com/catboost/doc/dg/concepts/python-reference_parameters-list-docpage/#python-reference_parameters-list

```
CatBoostClassifier(iterations=3000, border_count=20, l2_leaf_reg=5,
                   depth=7, eval_metric='AUC', use_best_model=True, random_seed=RANDOM)
```

- Optimizied hyperparameters' setting(XGBoostClassifier)
  - colsample_bytree: Subsample ratio of columns when constructing each tree, 0.7
  - subsample: Subsample ratio of the training instance, 0.7
  - max_depth: Maximum tree depth for base learners, 7
  - learning_rate: Boosting learning rate, 0.02

[ref] https://xgboost.readthedocs.io/en/latest/python/python_api.html

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bytree=0.7,
              gamma=0, learning_rate=0.02, max_delta_step=0, max_depth=7, min_child_weight=1, missing=None,
              n_estimators=2000, n_jobs=-1, nthread=-1, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None, silent=True, subsample=0.7)
```

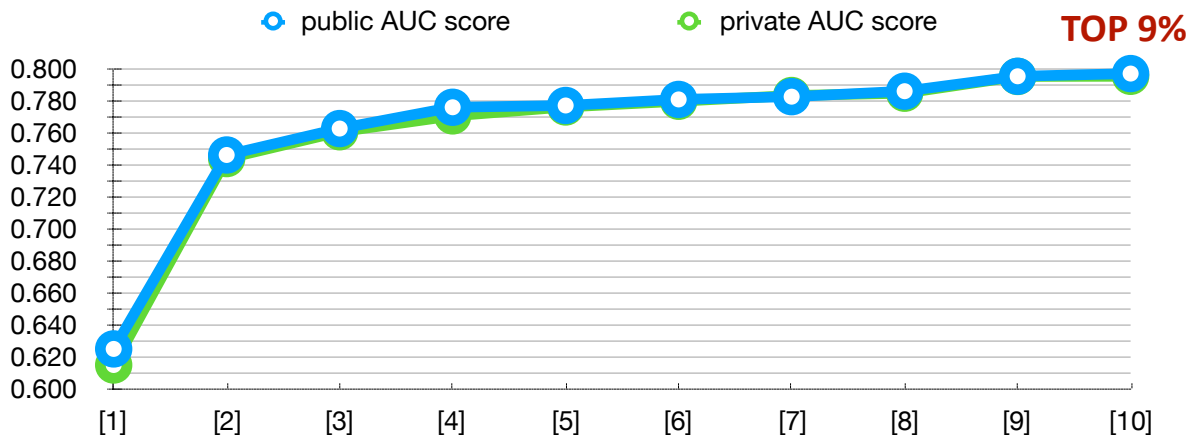
Justification

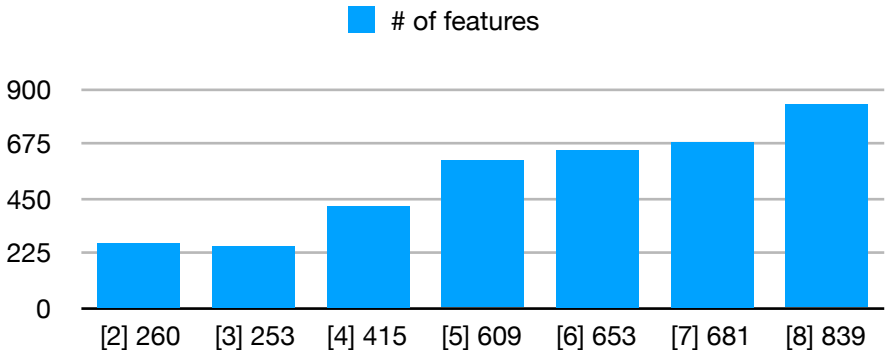
After all the hard work is done, we can submit the final prediction and evaluate Kaggle's test data. The prediction is evaluated on area under the ROC curve between the predicted probability and the observed target. The public AUC score is calculated with approximately 20% of the test data, and the private AUC score is calculated with approximately 80% of the test data. The final prediction got 0.79661 public AUC score(rank 2364th), and 0.79490 private AUC score(rank 613th, TOP 9%).

Conclusion

Free-Form Visualization

Even without any data preprocessing and feature engineering, gain-boosing models like LGBM perform much better than simple logistic regression. And as we preprocessed each relational databases and merge in the train/test sets, the total number of features grows and the the AUC score improves as well. Finally, after we optimized the hyperparameters and use stack technique, the final prediction can get TOP 9% on Kaggle's private leaderboard.





[1] (Logisticgression)Only use original application_train

[2] (LGBMClassifier)Only use original application_train

[3] (LGBMClassifier)Preprocessed application_train

[4] (LGBMClassifier)Add preprocessed bureau and bureau_balance

[5] (LGBMClassifier)Add preprocessed previous_application

[6] (LGBMClassifier)Add preprocessed POS_CASH_balance

[7] (LGBMClassifier)Add preprocessed installments_payments

[8] (LGBMClassifier)Add preprocessed credit_card_balance

[9] (LGBMClassifier)Use optimized hyperparameters

[10] After stacking technique applied.

Reflection

This is my first time to participate in Kaggle competition, which is beyond my imagination. I learned a lot from th wonderful articles and tutorials shared by talent people around the world. Although I know that the rankings on the public and private leaderboard will be different, I am still curious about why I can get good ranking on the private leaderboard(TOP 9%), but poor ranking(TOP 33%) on the public leaderboard. During the competition, I encountered some problems about computing power. In the beginning, I only used the macbook pro with 2.2GHz Intel Core i7 and 16GB DDR3 memory, it was too slow and I couldn't do anything while sleeping. Then I turned to GCE(Google compute engine) with n1-highmem-8 (8 vCPUs, 52 GB memory), which has a $300 free quota. But since the final dataset is large, if I run two files at the same time, python will raise MemoryError. I found a solution(using gc.collect()) from the Kaggle kernel, but it didn't help much. Then I used AWS(p2.8xlarge and p2.16xlarge), which is perfect but very expensive…

One interesting thing is that there are many experienced developers trying to design some AML(automatic machine learning) tools, which is very very exciting! In fact, I spent some time learning and using some AML tools like Featuretools(for automated feature engineering), SMOTE(for oversampling imbalanced data), tpot(for automated model tuning), and vecstack(for stacking). But due to time and computing power, I decided to manually design these blocks. The other interesting thing I found is that there are a lot of teams and many people don't even have any background knowledge on this application area. Next time I participate in the Kaggle competition, I will join/group a team to get more model diversity, dataset feature diversity, and brainstorming.

Improvement

Basically there are three improvements that need to be tried:

- Bayesian hyper-parameter optimization

Although after using GridSearch, we found better hyperparameter values and got better performance, those are actually not the best hyperparameters. Bayesian hyper-parameter optimization build a probability model and use it to select the most promising hyperparameters.

- Model diversity

In this project, we only use three gain-boosting model for stacking. Although I'd tried a few models like RandomForestClassifier and ExtraTreesClassifier. These models couldn't get good perforamce hence ruin the stacking. Neural Network should be a good option.

- Feature diversity

We didn't explore the dataset deeply and thoroughly, we could get much more useful features, not just 839 features.

Reference Links

1. An Overview of the Consumer Financial Protection Bureau's Ability-to-Repay and Qualified Mortgage Rule
   https://www.americanbar.org/publications/blt/2013/04/02_shatz.html
2. Anahita Namvar, Mohammad Siami, Fethi Rabhi, Mohsen Naderpour: "Credit risk prediction in an imbalanced social lending environment"
   https://arxiv.org/abs/1805.00801
3. Joaquín Abellán,Javier G.Castellano: "A comparative study on base classifiers in ensemble methods for credit scoring", Expert Systems with Applications Volume 73, 1 May 2017, Pages 1-10
   https://www.sciencedirect.com/science/article/pii/S0957417416306947?via%3Dihub
4. Xia Yufei, Liu Chuanzhe, Li YuYing, Liu Nana: "A boosted decision tree approach using Bayesian hyper-parameter optimization for credit scoring", Expert Systems with Applications Volume 78, 15 July 2017, Pages 225-241
   https://www.sciencedirect.com/science/article/pii/S0957417417301008
5. "Home Credit Default Risk", kaggle.com, 2018. [Online]
   https://www.kaggle.com/c/home-credit-default-risk/
6. "Home Credit Default Risk Data", kaggle.com, 2018. [Online]
   https://www.kaggle.com/c/home-credit-default-risk/data
7. A Gentle Introduction to Gradient Boosting
   http://www.ccs.neu.edu/home/vip/teach/MLcourse/4_boosting/slides/gradient_boosting.pdf
8. XGBoost, eXtreme Gradient Boosting (Github)
   https://github.com/dmlc/xgboost
9. LightGBM, Light Gradient Boosting Machine (Github)
   https://github.com/Microsoft/LightGBM
10. CatBoost: based on gradient boosting over decision trees (Github)

https://github.com/catboost/catboost
11. Capstone project benchmark model code (Gist)
https://gist.github.com/jo4x962k7JL/06af77c0d82da5dfbc2d82788d42659b
12. wikipedia: Receiver_operating_characteristic
https://en.wikipedia.org/wiki/Receiver_operating_characteristic
13. Wolf Street: Subprime Begins to Haunt Credit Card Balances
https://wolfstreet.com/2018/08/21/credit-card-balances-delinquencies-charge-offs/
14. Patterns of Missing Data
https://www.kaggle.com/jpmiller/patterns-of-missing-data
15. Github: vecstack
https://github.com/vecxoz/vecstack
16. Kaggle kernel：basic blends
https://www.kaggle.com/ishaan45/thank-you
17. Kaggle kernel：Different basic blends possible
https://www.kaggle.com/ashishpatel26/different-basic-blends-possible
18. Kaggle Discussion：1st Place Solution
https://www.kaggle.com/c/home-credit-default-risk/discussion/64821#380529
19. Basic Feature Engineering With Time Series Data in Python
https://machinelearningmastery.com/basic-feature-engineering-time-series-data-python/
20. Kaggle kernel：LighGBM_with_Selected_Features
https://www.kaggle.com/ogrellier/lighgbm-with-selected-features/code
21. Kaggle kernel：Good_fun_with_LigthGBM
https://www.kaggle.com/ogrellier/good-fun-with-ligthgbm/code
22. Python target encoding for categorical features
https://www.kaggle.com/ogrellier/python-target-encoding-for-categorical-features
23. A Preprocessing Scheme for High-Cardinality Categorical Attributes in Classification and Prediction Problems
https://kaggle2.blob.core.windows.net/forum-message-attachments/225952/7441/high%20cardinality%20categoricals.pdf
24. Kaggle kernel：open solution
https://www.kaggle.com/kkaczmarek/kernels?sortBy=voteCount&group=everyone&pageSize=20&userId=845888
25. Overview of the 5th solution +0.004 to CV/Private LB of your model
https://www.kaggle.com/c/home-credit-default-risk/discussion/64625
26. Kaggle discussion：＃14 solution
https://www.kaggle.com/c/home-credit-default-risk/discussion/64502
27. Kaggle kernel：updated-0-792-lb-lightgbm-with-simple-features
https://www.kaggle.com/jsaguiar/updated-0-792-lb-lightgbm-with-simple-features/code
28. How to Handle Missing Data
https://towardsdatascience.com/how-to-handle-missing-data-8646b18db0d4
29. A Conceptual Explanation of Bayesian Hyperparameter Optimization for Machine Learning

https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f