Home Credit Default Risk Prediction

MSBA 6420: PREDICTIVE ANALYTICS GROUP PROJECT

Submitted by: Yu Chun Peng, Chien-Chu Hsu, Chia-Yen Ho, Devansh Bhasin

Contents

Background & Context

It has always been a challenging task for financial institutions to determine the financial strength of a potential customer to repay the loan amount within a defined time period. With the right approach, financial organizations can avoid losses and make huge profits.

Home Credit is an international non-bank financial institution, focusing on installment lending primarily to people with little or no credit history. Home Credit presented a Kaggle challenge to identify who is able to repay the loan based on loan application, demographic and historical credit behavior data, and other alternative data.

Problem Statement

There are some people who don't have a specific credit history or even a bank account, but they need a loan for some reasons. However, without adequate credit history, it may be difficult for Home Credit to provide loans to these individuals, as these loans can be highly risky. In this case, some lenders even tend to exploit borrowers by demanding exorbitant interest rates, putting those borrowers in an unfair and dangerous situation.

Another scenario is when the loan applicant does have a credit history, but the records are scattered across different organizations, and reviewing this historical data can be time-consuming. Especially as the number of applicants increases, this will become a serious problem for Home Credit.

For these cases, we will try to build models to predict how capable each applicant is of repaying a loan, thus the process can be simplified and effective for both the Home Credit and applicants. This is a classification problem where the label is a binary variable. 0 represents the applicant will repay the loan on time, while 1 represents that the applicant will have difficulty repaying the loan.

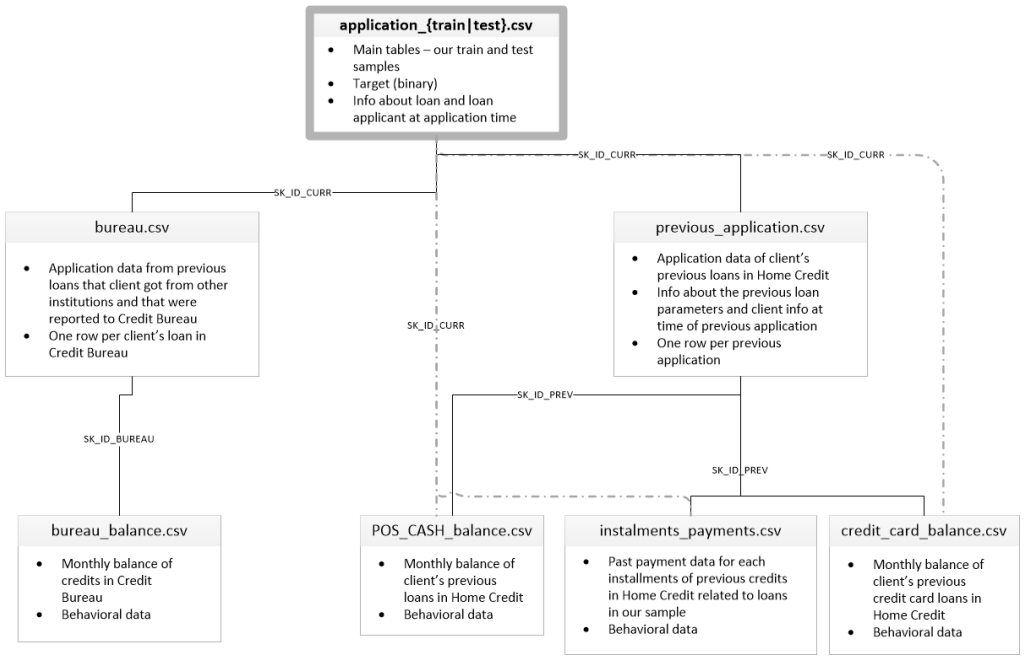
Result Summary

Our best model is a blend of 3 popular Gradient Boosted Tree methods: XGBoost, LightGBM and Catboost and achieves an ROC AUC score of **0.79316.** The link for the codes of implementation and our best file for submission can be found in the appendix section of the report.

Data Sources

The data is provided by Home Credit in a Kaggle Challenge. There are 8 datasets in total. The link to the original datasets is here.

- **Application Train:** The main table that includes the target variable. One observation represents one loan application. We use this as the train data.
- **Application Test:** The main table but excludes the target variable. We use this as the test data.
- **Bureau Data:** The data on previous loans of clients provided by other financial institutions that were reported to the Credit Bureau. It is possible that there are many rows of credits the client had in the Credit Bureau.
- **Bureau Balance Data:** The monthly balances of previous credits in the Credit Bureau.
- **Previous Application:** Clients' previous loan applications with Home Credit.
- **Cash Balance:** Monthly balance of previous POS (point of sales) and cash loans that the client had with Home Credit.
- **Installments Payments:** Clients' repayment data for each installment of credit with Home Credit.
- **Credit Card Balance:** Monthly balance of previous credit cards loans that the client had with Home Credit.



Overview of the Kaggle database schema along with joining keys

Software & Tools

We are using the most popular tools and libraries of the Python Data Science stack to achieve our results.

Environment Specifications:

Anaconda 2021.11: Manages working environment

Python 3.9: Main coding/scripting language

Pandas 1.4.2: Used for dataframe operations

Numpy 1.22.3: Used for linear programming

Sklearn 1.0.2: Main machine learning library

LightGBM 3.3.2: Library for implementing LightGBM

XGBoost 1.5.2: Library for implementing XGBoost

Catboost 0.24.1: Library for implementing Catboost

The models were built and computed on both personal computers and online Kaggle notebooks. Following are the specifications we used:

Personal Computer Specifications:	**Kaggle Kernel Specifications:**
CPU: Intel Core i7 - 1065G7 (25W TDP) GPU: Intel Iris Plus Graphics RAM: 16GB DDR4	CPU: Intel(R) Xeon(R) CPU @ 2.00GHz GPU: Tesla P100-PCIE-16GB RAM: 16GB (Max 13GB Available)

Data Processing and Exploration

A) IMPORTING DATA

This project contains 8 datasets. Each dataset can be joined together using the key id 'SK_ID_CURR'. So, we do the data preprocessing independently for each dataset and then join them together using the key id.

B) DATA PREPROCESSING

We create a function to contain the preprocessing part for every dataset. For example, the basic data preprocessing contains these steps:

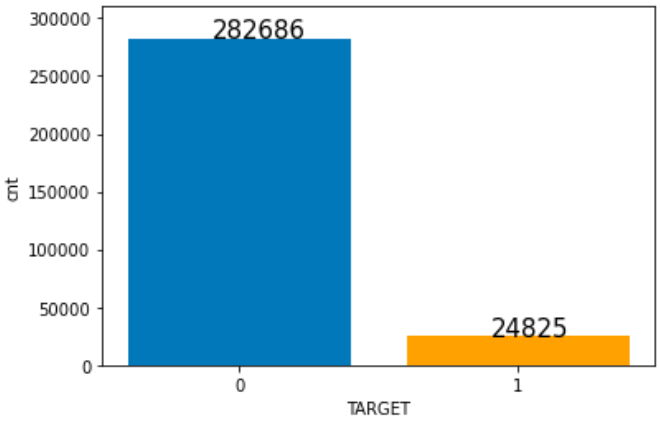
1. Turn categorical features into OneHotEncoder
2. Impute missing values
3. We created some valuable features based on our understanding of the dataset and company background. For example, we create some polynomial features for 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'DAYS_BIRTH', because we found that those are important features and there might be some higher dimension relation with the dependent variables. Another example is we create 'CREDIT_INCOME_PERCENT', which is 'AMT_CREDIT' divided by 'AMT_INCOME_TOTAL' because the percentage sounds more reasonable than the single number when evaluating one's payment ability
4. Some percent features by dividing a quantity part by the total quantity were also created
5. For each dataset, we grouped by key id and aggregated numerical features using statistical summaries, such as mean, min, max, standard deviation, and variance
6. Join data together using key id. We get a single table containing all the features

C) FEATURES ENGINEERED

Our combined dataset comprising all files contains over 320 features by default. Over this, we have generated more than 290 features to improve the performance of our models. We noticed a noticeable improvement in the performance of our models when more features were added.

D) IMBALANCE DATA PROBLEM

The target variable is what we want to predict: 0 represents that the applicant will repay the loan on time, while 1 represents that the applicant will have difficulty repaying the loan. We first examined the number of loans in each category in the application train dataset.



From the graph, we can see that 282,686 (91.93%) loans were repaid on time and 24,825 (8.07%) loans were not repaid. Therefore, the data is imbalanced. There are more loans that were repaid on time than loans that were not repaid.

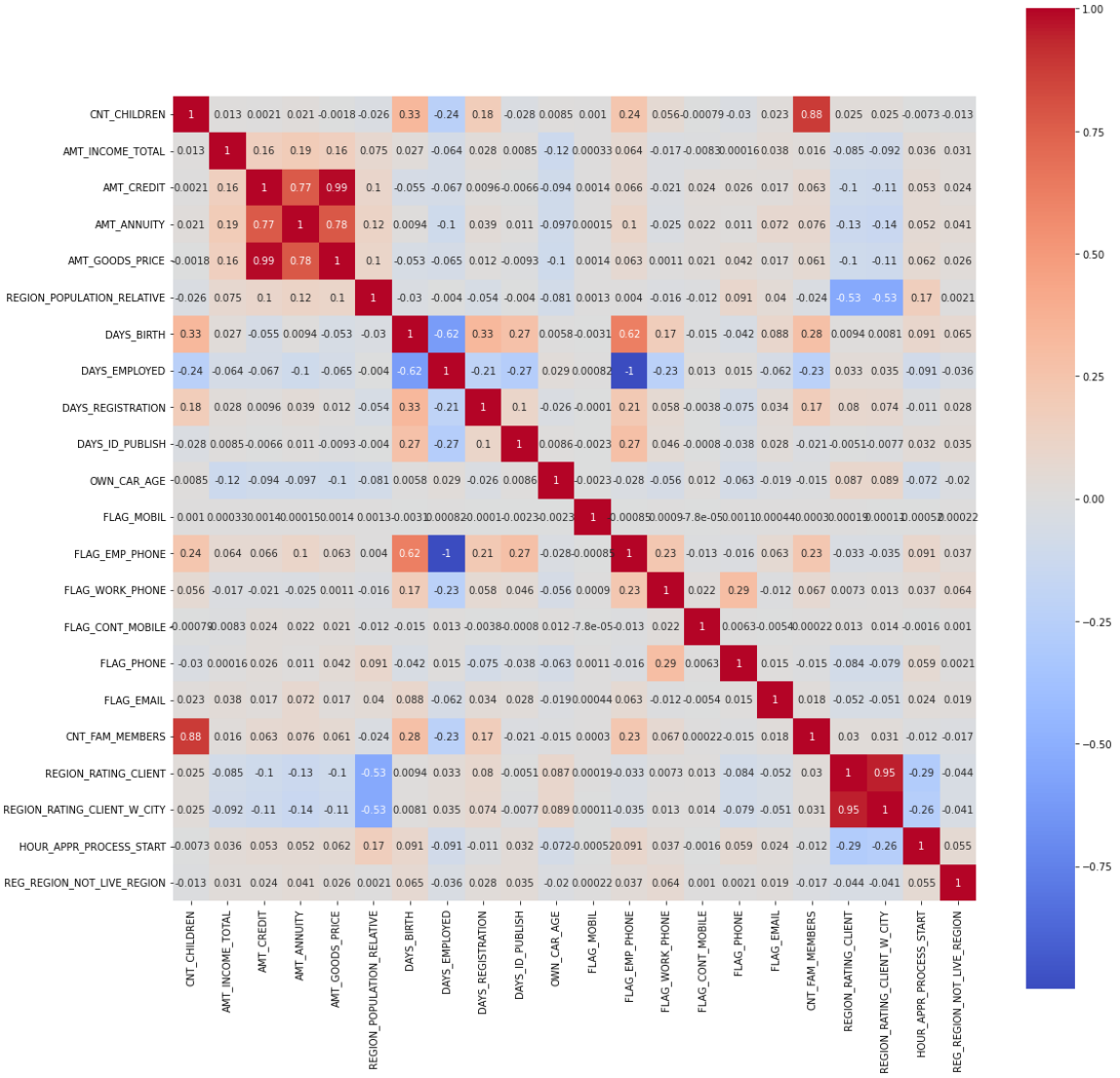
E) MISSING VALUE ANALYSIS

Just showing the first 20 features and their missing value percentages. 12 out of 20 have more than 50% of data points missing values.

	missing vlaue pct
AMT_ANNUITY	0.00
AMT_GOODS_PRICE	0.00
NAME_TYPE_SUITE	0.00
OWN_CAR_AGE	0.66
OCCUPATION_TYPE	0.31
CNT_FAM_MEMBERS	0.00
EXT_SOURCE_1	0.56
EXT_SOURCE_2	0.00
EXT_SOURCE_3	0.20
APARTMENTS_AVG	0.51
BASEMENTAREA_AVG	0.59
YEARS_BEGINEXPLUATATION_AVG	0.49
YEARS_BUILD_AVG	0.66
COMMONAREA_AVG	0.70
ELEVATORS_AVG	0.53
ENTRANCES_AVG	0.50
FLOORSMAX_AVG	0.50
FLOORSMIN_AVG	0.68
LANDAREA_AVG	0.59
LIVINGAPARTMENTS_AVG	0.68

F) CORRELATION ANALYSIS

Use a part of the features to plot out the correlation. Features such as AMT_CREDIT, AMT_ANNUITY, AMT_GOODS_PRICE have high relations with each other. But other features have a relatively low correlation.



G) EXPLORATORY DATA ANALYSIS

First, we look at the distribution of each variable towards the target variable.

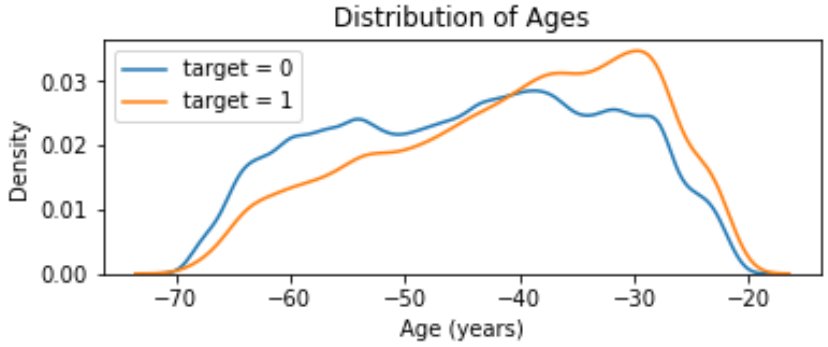
Ages: The target = 1 curve skews towards the younger end of the range.

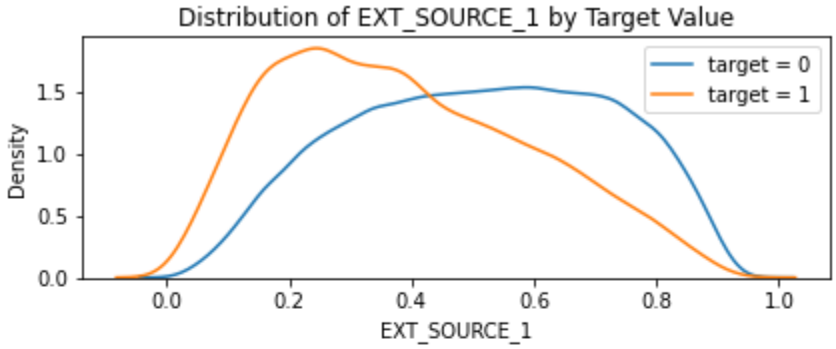
EXT_SOURCE_1: The target = 1 curve skews towards smaller EXT SOURCE 1 values.

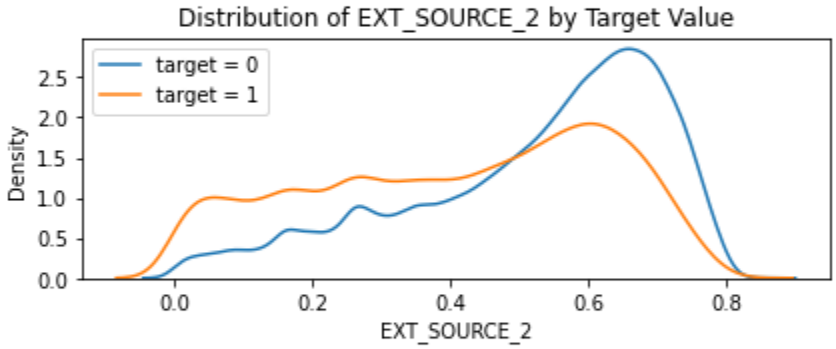
EXT_SOURCE_2: The target = 1 curve skews towards larger EXT SOURCE 2 values.

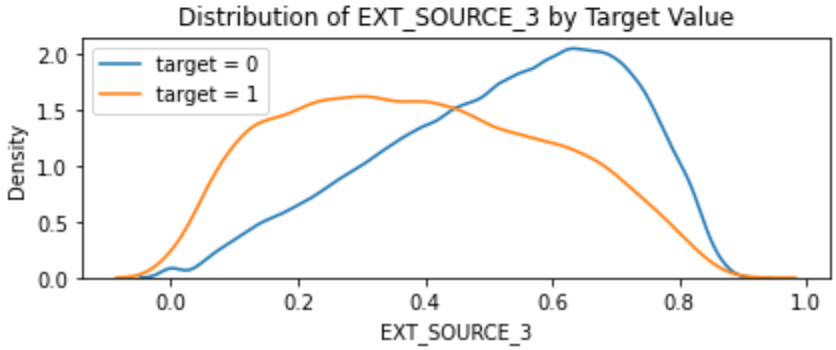
EXT_SOURCE_3: The target = 1 curve skews towards smaller EXT SOURCE 3 values.

We can see that the distributions are different for target 1 and target 0.

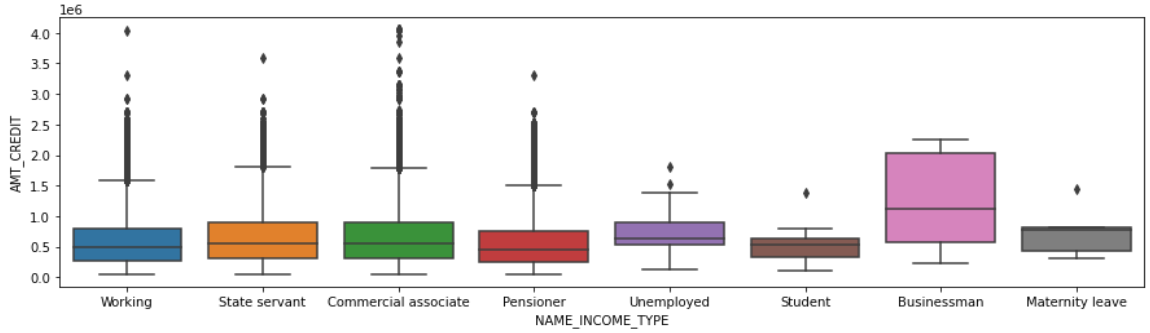

Distribution of Ages
target = 0
target = 1
Density
0.03
0.02
0.01
0.00
−70
−60
−50
−40
−30
−20
Age (years)


Distribution of EXT_SOURCE_1 by Target Value
target = 0
target = 1
Density
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0
EXT_SOURCE_1

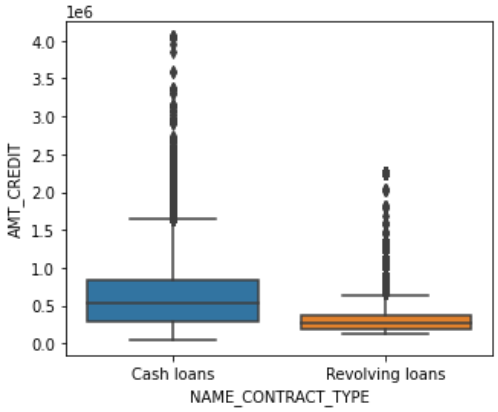

Distribution of EXT_SOURCE_2 by Target Value
target = 0
target = 1
Density
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
EXT_SOURCE_2


Distribution of EXT_SOURCE_3 by Target Value
target = 0
target = 1
Density
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0
EXT_SOURCE_3

Secondly, we use boxplots to see each income type distribution with AMT_CREDIT. Businessmen have a higher amount of credit while students have less amount of credit. Besides, people applying for cash loans also have a higher amount of credit versus those applying for revolving loans.

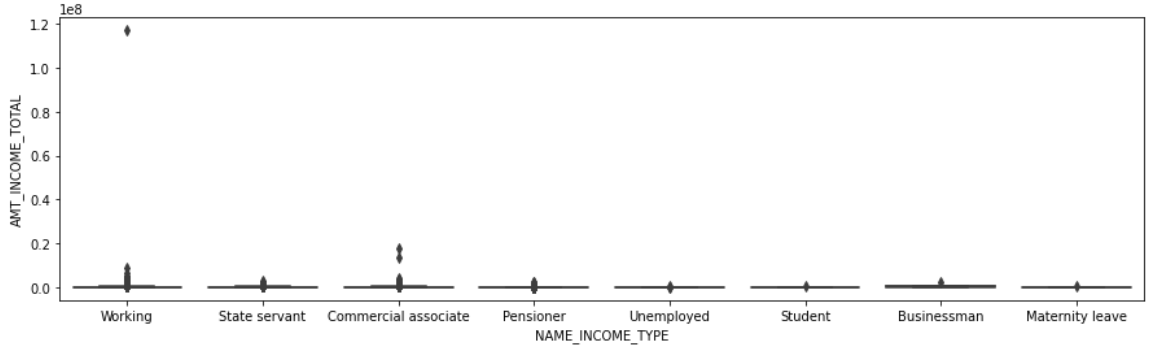


Credit Amount vs Income Type



Credit Amount vs Contract Type

We can see that there are outliers in AMT_INCOME_TOTAL while the customer's income type is working or commercial associate. But for other income types, there seem to be no outliers.



Income Total vs Income Type

Predictive Modeling

A) IMPLEMENTATION METHODOLOGY

Scikit-learn was the main tool we used for data preparation for defining test and train sets and bringing the data in the form . This includes OntHotEncoder, TrainTestSplit, Logistic Regression, XGBoost and Catboost. Besides, we also use the original LightGBM and BayesianOptimization library to optimize the computation.

Our performance metric is AUC. We try different feature engineering and feature selection for each model to get the best performance. After trying different models, we stack the models together to get more generalized results.

B) MODELS IMPLEMENTED

B1. Logistic Regression

Logistic Regression is one of the simplest machine learning algorithms and is easy to implement yet provides great training efficiency in some cases. It makes no assumptions about distributions of classes in feature space. It can interpret model coefficients as indicators of feature importance. We can also consider regularization techniques to avoid over-fitting. However, non-linear problems can't be solved with logistic regression because it has a linear decision surface. Also, It is tough to obtain complex relationships using logistic regression. Therefore we try on some more powerful and compact algorithms such as XGBoost to outperform this algorithm.

```
log_reg.get_params
```

```
<bound method BaseEstimator.get_params of LogisticRegression(C=0.0001)>
```

Logistic Regression Parameters

B2. XGBoost

XGBoost is an optimized library for regularized distributed gradient boosting designed to be highly efficient, flexible, and portable. This algorithm has been dominating applied machine learning and Kaggle competitions for structured or tabular data. XGBoost makes no statistical assumptions about the data and uses a more regularized approach for the

model building to prevent overfitting. It also works well with other libraries like scikit-learn in python and has integrated support for GPUs for highly efficient and parallel training. At this stage, XGBoost is viewed as a production grade software by many data scientists which is battle tested on various kinds of datasets.

```
clf_xgb = xgboost.XGBClassifier(
 n_estimators = 10000,
 learning_rate = 0.01,
 colsample_bylevel = 0.916,
 colsample_bynode = 0.5289,
 colsample_bytree = 0.52050,
 gamma = 0.85416,
 max_depth = 6,
 min_child_weight = 67,
 reg_alpha = 0.24312,
 reg_lambda = 0.104283,
 subsample = 0.854,
 tree_method = 'gpu_hist',
 gpu_id = 0,
 eval_metric='auc',
 early_stopping_rounds=200,
 random_state = 51412)
```

Tuned XGBoost Parameters

B3. LightGBM

Light GBM is a fast, distributed, high-performance gradient boosting framework that uses a tree-based learning algorithm. Light GBM splits the tree leaf-wise with the best fit whereas other boosting algorithms split the tree depth-wise or level-wise rather than leaf-wise. In other words, Light GBM grows trees vertically while other algorithms grow trees horizontally. Compared to XGBoost, it is capable of performing equally well with large datasets with a significant reduction in training time.

```
parameters:  {'bagging_fraction': 0.8, 'feature_fraction': 0.9, 'lambda_l1': 0.0, 'lambda_l2': 3.0,
'learning_rate': 0.05, 'max_depth': 8.99, 'min_child_weight': 39.99611572837116, 'min_split_gain':
0.1, 'num_leaves': 45.0, 'subsample': 0.7}
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=0.7 will be ignored. Current value: bag
ging_fraction=0.8
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of testing was 1.791438 s
econds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 86861
[LightGBM] [Info] Number of data points in the train set: 307511, number of used features: 579
[LightGBM] [Info] Start training from score 0.080729
AUC: 0.8009927366379824
CPU times: user 1h 23min 8s, sys: 3min 58s, total: 1h 27min 6s
Wall time: 1h 2min 32s
```

Tuned LightGBM Parameters

B4. Catboost

Catboost is known as Categorical Gradient Boosting. It is a new machine learning technique developed by Yandex based on gradient boosting. Catboost usually outperforms existing boosting algorithms like XGBoost and Light GBM. In addition, Catboost is easy to implement and is very powerful. It implements symmetric trees, thus helping in decreasing prediction time. In short, Catboost has the advantage of improved results and efficient training compared to XGBoost and LightGBM.

After tuning the hyper-parameter, the final version of the Catboost model looks as below:

```
clf_catboost = CatBoostClassifier(iterations=2000,
                                  learning_rate=0.05,
                                  depth=6,
                                  l2_leaf_reg=40,
                                  bootstrap_type='Bernoulli',
                                  subsample=0.7,
                                  scale_pos_weight=5,
                                  eval_metric='AUC',
                                  metric_period=50,
                                  od_type='Iter',
                                  od_wait=45,
                                  random_seed=17,
                                  allow_writing_files=False)

clf_catboost.fit(X_train,y_train,verbose=True)
```

Tuned Catboost Parameters

C) HYPERPARAMETER TUNING

Hyperparameters are parameters that we set for the models. We performed hyperparameter tuning for each learning algorithm to choose the optimal set of hyperparameters. We considered methods, such as GridSearch, random search and Bayesian Optimization. Eventually, we decided to conduct Bayesian Optimization for hyperparameter tuning because Bayesian Optimization is more efficient in selecting the hyperparameters than GridSearch and random search.

D) BAYESIAN OPTIMIZATION

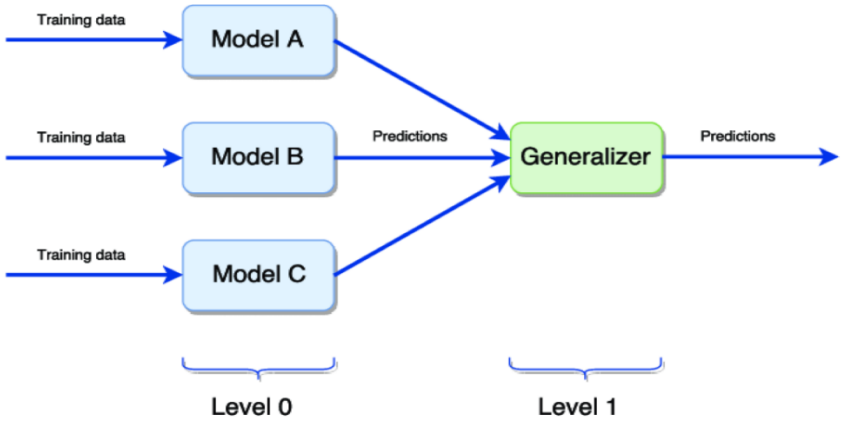
Bayesian optimization is a sequential model-based optimization algorithm that uses the results from the previous iteration to decide the next hyperparameter candidates. The process is repeated iteratively to converge to an optimal point. Therefore, Bayesian optimization is efficient in selecting hyperparameters because it selects the hyperparameters in a combined perspective. In sum, it can find the best hyperparameters in less time and in fewer iterations by prioritizing hyperparameters that appear more promising from past results.

E) FEATURE SELECTION

When using a tree-based model, we calculated and plotted the feature importances. Then select the first 30, 60, and 100 features and rerun the model. However, the model performance after feature selection is worse than the original. Therefore, in this case, we did not use any feature selection method in the final models.

F) STACKING

Stacking is a model-centric ensemble model that can combine the predictions from multiple machine learning models. It can increase model performance by reducing bias and reducing variance. We stack the results of XGBoost, LightGBM and Catboost models by applying the weightage 75% of XGBoost, 10% of Catboost and 15% LightGBM.



G) MODEL OPTIMIZATION AND DEPLOYMENT

For deploying our models we first had to select the best parameters to maximize the ROC AUC score on the test data. AUC represents the probability that a random positive point is positioned above a random negative point when all points are ordered according to their predicted probabilities. Since we used the most popular gradient boosted trees for classification our optimization hyperparameters mostly included:

1. **Learning Rate:** Rate of learning after each iteration. Very high learning rate can lead to overfitting
2. **Max Depth:** How long do we allow the trees to grow. Tree with very high depth can lead to overfitting
3. **Regularization Parameters:** The magnitude of L1 and L2 regularization done to prevent overfitting
4. **Subsampling:** Percentage of data selected with replacement in every iteration. This is used to reduce variance.
5. **Column Sampling:** Subset of columns used in the decision tree estimators. Useful for automatic feature selection while training the model

Along with the above parameters, some algorithm specific parameters were also tuned to maximize the performance (can be checked in the code snippets.)

We implemented the above discussed Bayesian Optimization method to select the best hyperparameters, while saving on computational resources at the same time. At Least 10 iterations (Random Initialization + Optimization Rounds) were used for each Gradient Boosted method to find the optimal parameters.

We also took advantage of the GPU optimizations of the gradient boosting algorithms to train the models at high speed. In our tests, using GPUs provided over 7x improvement in the training speed compared to CPU only training.

For Kaggle Notebooks: We used a memory reduction algorithm on the datasets that uses the optimal data types wherever possible to save on RAM space. Use of this algorithm reduced the memory occupied by large datasets by upto 67% and prevented the Kaggle Kernel from running out of RAM.

Results

Following are the results of the best iteration of each model:

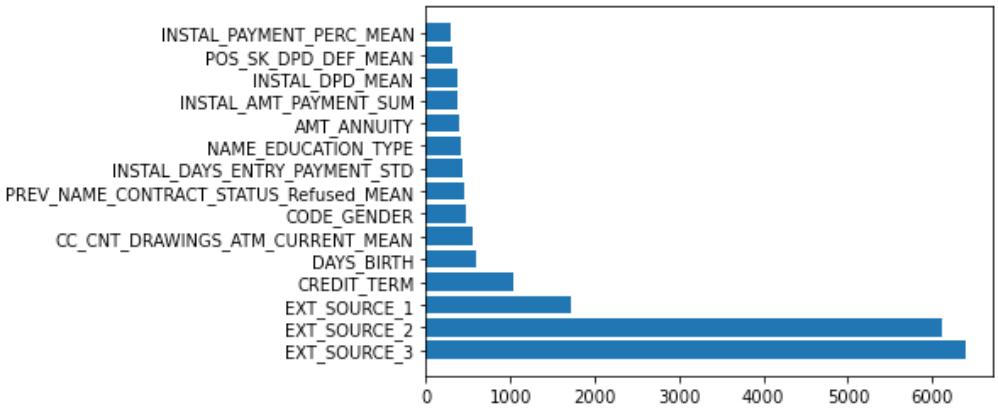
Algorithm	AUC on Kaggle(Public/Private Score)
Logistic Regression	0.71174/0.69922
XGBoost	0.79111/0.78885
LightGBM	0.77594/0.77265
Catboost	0.78638/0.78778
Stacking	**0.79316/0.79045**

Our best model is achieved by a blend of: **75% XGBoost + 15% LightGBM + 10%CatBoost**

It is to be noted that our best model is trained on only 583 features and is still achieving Considerable predictive power compared to other models on Kaggle. Because of using less features, out computation time is less that 50% compared to some of the best implementations on Kaggle which use 1000+ features.

Feature Importance

We explained the feature importance of top 10 features:



1. **EXT_SOURCE 1 to 3:** Normalized score from external data source
2. **CREDIT_TERM:** A feature we create by AMT_ANNUITY divided by AMT_CREDIT
3. **DAYS_BIRTH:** Client's age in days at the time of application

4. **CC_AMT_DRAWINGS_ATM_CURRENT:** An aggregated feature of AMT_DRAWINGS_ATM_CURRENT, which is the amount drawn at the ATM during the month of the previous credit
5. **CODE_GENDER:** Gender of the client
6. **PREV_NAME_CONTRACT_STATUS_Refused_MEAN:** The mean of the refused application of NAME_CONTRACT_STATUS, which is contract status during the month
7. **INSTAL_DAYS_ENTRY_PAYMENT_std:** The standard deviation of DAYS_ENTRY_PAYMENT, which is the time of the installments of previous credit paid actually (relative to the application date of the current loan)
8. **NAME_EDUCATION_TYPE:** Level of highest education the client achieved

Business Recommendation

Our best model has considerable predictive power to predict the customers who have higher probability of defaulting and not returning loans. This model can be used to automate the application processing part of loan approvals. It is also scene that external data sources are enhancing the decision-making process in our predictive models. More data like this can be gathered to make our model more robust.

Limitations and Future Scope

A) Implementing Neural Networks

We selected gradient boosted tree methods instead of neural network architectures because of the superior performance of the former on tabular datasets. However, use of Deep Learning Techniques and Autoencoders could be explored for automatic feature selection, dimensionality reduction, noise removal and feature transformations. The output of neural network layers can be used as an input for boosted tree algorithms to potentially improve performance.

B) Treating Missing Values

Some of the features used for building our model contained over 60% missing records. We accounted for this by using advanced algorithms that have capabilities to handle missing values inherently. However, the performance of the models could be improved if the data gathering was more complete. Additionally, some machine learning based methods could be explored to impute NA values in columns based on the context.

Appendix

Github Link: GitHub - chiayenho/Home-Credit-Default-Risk

The above GitHub link contains all our coding implementations and also the best file we used for submission

References:

1) Home Credit Default Risk | Kaggle

2) HOME CREDIT DEFAULT RISK — An End to End ML Case Study — PART 1: Introduction and EDA | by Rishabh Rao | TheCyPhy | Medium