Home Credit Default Risk

Predicting Home Credit Customer Default Risk using XGBoost and LightGBM with Domain Features and Aggregated Features

Yufan Li, Runan Cao, Will Missling, Bailey Hill

Table of Contents (Hyperlinked)

Introduction

Background and Problem Statement

Home Credit is a consumer finance organization with operations in nine countries. They offer point-of-sale (POS) loans, cash loans, and revolving credit options available through the company's online and physical network. Through the company's POS loans they offer a point of entry to consumers who struggle to get loans due to non-existent or deficient credit history. The company's business model strives to provide a financially inclusive experience for the underserved community within the banking industry.

In order to foster a positive loan experience for consumers, Home Credit is interested in using a variety of data to predict clients repayment abilities and the risk of defaulting on their loans. Home Credit believes that with a better system of predictions they can be able to decrease the chances for a client capable of repaying their loan being rejected and set up more clients for financial success.

Data Sources

All of the data used for this analysis came from a previous Kaggle competition. Overall, there were 8 total tables with different information about prior Home Credit applicants.

1. Application (train): this is the main table that has information about the loan and loan applicant at the time of the application. Each row in the table is unique to a loan, and a Target column is included which notes '1' if the client had payment difficulties, and '0' if not, which means this is a binary classification problem. Payment difficulties is defined as the client had a late payment "more than X days on at least one of the first Y installments of the loan in the sample."
2. Application (test): this table has the exact same columns as the train table, minus the Target column.
3. Bureau: for clients who have a loan in the sample, this table includes all of their previous credits reported to the Credit Bureau by other financial institutions before the client's application date.
4. Bureau Balance: this table includes monthly balances of previous credits in the Credit Bureau with one row for every month of previous credit history.
5. POS Cash Balance: for point of sale and cash loans through Home Credit, this table shows the applicants' previous monthly balances.
6. Credit Card Balance: this table shows the monthly balances for applicants' previous credit cards through Home Credit.

7. Previous Applications: for clients with loans in the sample, this table displays all prior applications for Home Credit loans.
8. Installments Payments: this table displays the repayment history of previously disbursed credits with separate rows for each payment and missed payment. Also, a row could include either one payment of one installment or one installment corresponding to one payment of a previous credit from Home Credit.

Analysis

Methodology

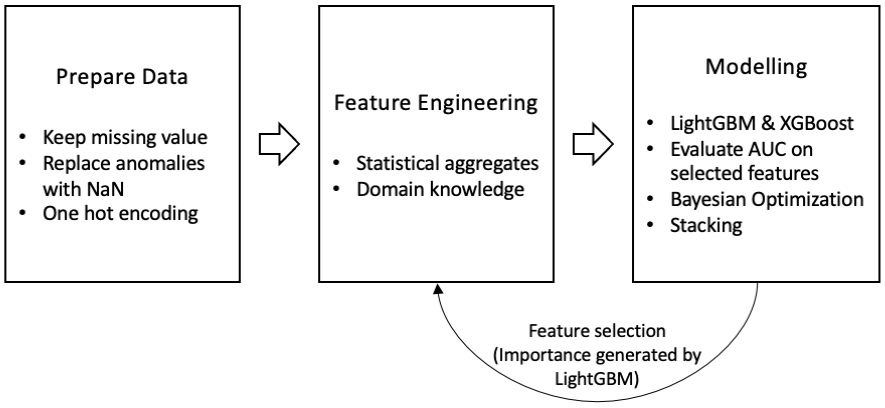
The first step in the analysis process was to start with a cursory exploration of the data so as to better understand the problem of interest and data available–this is essential as predictive models are fueled by the quality of data which is fed to the algorithms. Following this initial exploration was the pre-processing and data cleaning step. During this step the data is examined for noisy data points, extreme outliers, and missing values. However, considering the actual meanings of missing values, we did not do any processing on them.

After the pre-processing step was completed, feature engineering was conducted. Feature engineering involves creating statistical aggregates and domain knowledge features by merging various tables. The detailed methodologies and explanations can be found in the feature engineering section below.

Following the feature engineering, the central modeling process began with two different algorithms: XGBoost and LightGBM. These algorithms were implemented on several different feature sets as a means to capture best model performance measured on the Area Under the ROC Curve metric or AUC.

For feature selection, we used LightGBM's feature importance tool to look at feature importances set at certain thresholds to produce a reduced set of features. Additionally, the effect of correlated features was examined.

Finally in order to further improve the performance of the model, Bayesian optimization was used for hyperparameter tuning, and a stacking technique was also tried.



Data Preparation

- In this case, our team replaced the extreme outliers in the DAYS_EMPLOYED variable with the NaN.
- We did one-hot encoding on categorical data.
- There are many ways to deal with missing value. We tried to impute median to columns with missing values but didn't see good performance. So we decided to choose models which can handle missing values- XGBoost and LightGBM.
- We noticed that the dataset is imbalanced so we added class 'weight='balanced'' in the models' parameter setting.
- We did exploratory analysis- generating KDE plot to look at the discriminative ability of features to the target variable

Feature Engineering

Feature engineering consisted of two main focuses: Aggregate features and Domain features. Our hope with creating the aggregated features is to provide greater tidiness and representation to each row or instance of the data. Meanwhile, the hope with Domain features was to establish variables which could offer predictive value in the modeling process while also having a somewhat intuitive nature to them. The tables below offer explanations of the added features.

Aggregated Features (some further explanation below table)

Numeric Aggregation	Provides aggregation for existing numeric features based on Count, Min, Max, Mean, and Sum.
Categorical Aggregation	Creates an id for the column type and provides dummy variables to then aggregate based on mean, count, and sum.

Client Aggregation	Provides aggregation for tables with loan-level and client-level data. This applies to the Cash, Credit, and Installments/Payments tables.

Client Aggregation as outlined in the table above was applied to the Cash, Credit, and Installments/Payments table. This function works by receiving arguments for the table of interest, grouping variables to identify clients and associated loans (SK_ID_PREV, SK_ID_CURR), and names to call the resulting columns. What it does is perform categorical and numeric aggregation at the loan level first and then concatenating the results to the client id so as to provide comprehensive loan related information for each client in a single instance which provides better data for training the model.

Added Domain Features (some further explanation below table)

Name	Explanation	Formula	Table
DAYS_EMPLOYED_PERC	Age-adjusted current employed days	DAYS_EMPLOYED / DAYS_BIRTH	Main
INCOME_CREDIT_PERC	Income to credit ratio	AMT_INCOME_TOTAL / AMT_CREDIT	Main
INCOME_PER_PERSON	Current income adjusted for household size	AMT_INCOME_TOTAL / CNT_FAM_MEMBERS	Main
ANNUITY_INCOME_PERC	Annuity to income ratio	AMT_ANNUITY / AMT_INCOME_TOTAL	Main
PAYMENT_RATE	Annuity to credit ratio	AMT_ANNUITY / AMT_CREDIT	Main
debt_to_credit	Debt to credit ratio; current debt amount on Credit Bureau credit over the current credit amount on the Credit Bureau credit	AMT_CREDIT_SUM_DEBT / AMT_CREDIT_SUM	Bureau
unused_credit	The ratio of total credit limit to total used credit	AMT_CREDIT_SUM_LIMIT / AMT_CREDIT_SUM	Bureau
over_due_ratio	The ratio of total overdue to total utilized credit	AMT_CREDIT_SUM_OVERDUE / AMT_CREDIT_SUM	Bureau

annuity_to_credit	Annuity to credit ratio in the Credit Bureau; captures the payment rate for previous loans	AMT_ANNUITY / AMT_CREDIT_SUM	Bureau
CREDIT_approved_ ratio	Approved Credit Ratio	AMT_CREDIT / AMT_APPLICATION	Previous Application
application_to_ annuity	Application to annuity ratio; amount applied for to the amount of annuity	AMT_APPLICATION / AMT_ANNUITY	Previous Application
DOWN_PAYMENT_ to_credit	An initial payment to credit ratio	AMT_DOWN_PAYMENT / AMT_CREDIT	Previous Application
AMT_GOODS_ PRICE_to_credit	A Goods Price to Credit ratio; the price of the good(s) on the previous application over the total credit received on the previous application	AMT_GOODS_PRICE / AMT_CREDIT	Previous Application
INSTALMENT_to_ balance	Previous installments to monthly balance ratio	CNT_INSTALMENT / MONTHS_BALANCE	Cash
INSTALMENT_past _future	Previous installments to future installments ratio	CNT_INSTALMENT / CNT_INSTALMENT_ FUTURE	Cash
balance_credit_ratio	Balance amount to Credit limit ratio; monthly balance of previous credit over the total monthly credit card limit	AMT_BALANCE / AMT_CREDIT_LIMIT_ ACTUAL	Credit
balance_receivable_ ratio	Total amount receivable from previous credit to monthly balance of previous credit ratio	AMT_TOTAL_ RECEIVABLE / AMT_BALANCE	Credit
payment_to_ receivable	Ratio of Monthly total amount paid on previous credits to total amount receivable from previous	AMT_PAYMENT_TOTAL _CURRENT / AMT_TOTAL_ RECEIVABLE	Credit

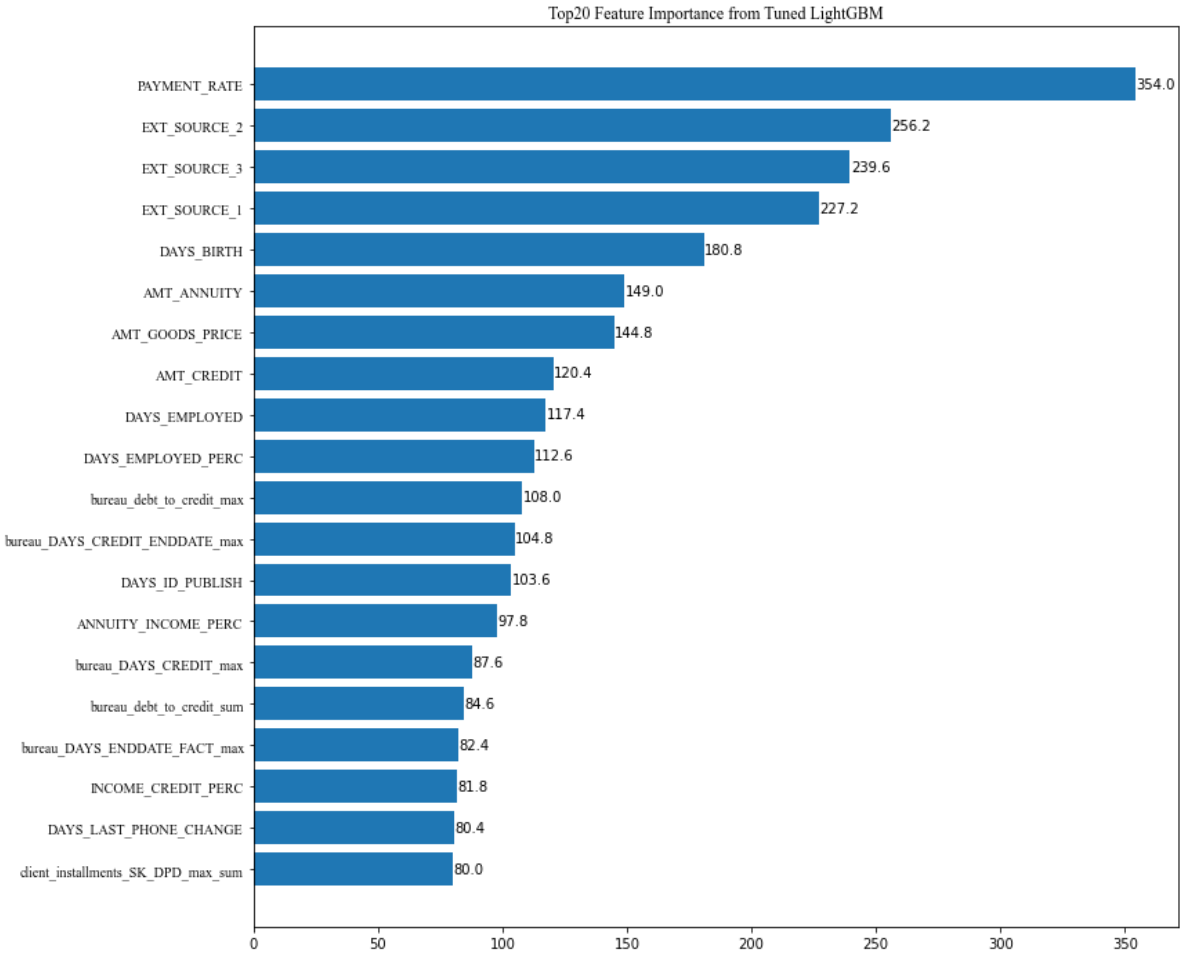
	credit		
drawing_to_credit	Ratio of monthly credit amount drawn at ATMs to monthly credit card limit	AMT_DRAWINGS_ATM_CURRENT / AMT_CREDIT_LIMIT_ACTUAL	Credit
AMT_DIFF	Difference between installment amount and payment amount	AMT_INSTALMENT - AMT_PAYMENT	Installments
AMT_RATIO	Amount payment to amount installment ratio with + 1 correction to prevent zero division error	(AMT_PAYMENT +1)/ (AMT_INSTALMENT + 1)	Installments
SK_DPD	Creates a Days Past Due column similar to that of other tables	DAYS_ENTRY_PAYMENT- DAYS_INSTALMENT	Installments
INS_IS_DPD	Creates a binary variable for whether or not the installment is past due	Variable is set equal to 1 for yes and 0 for no	Installments
INS_IS_DPD_UNDER_120	Creates a binary variable for whether or not the installment is under 120 days past due	Variable is set equal to 1 for yes and 0 for no	Installments
INS_IS_DPD_OVER_120	Creates a binary variable for whether or not the installment is over 120 days past due	Variable is set equal to 1 for yes and 0 for no	Installments

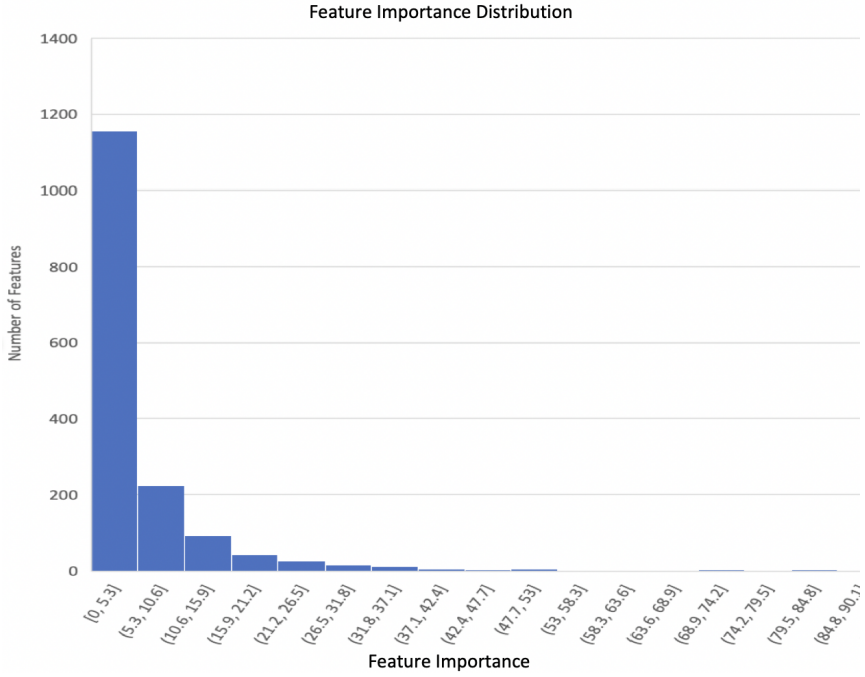
In engineering the features, we sought to capture as much information as possible by capturing the relationships between many of the features. A majority of the features in the table above are made up of ratios. We created these ratios intuitively to help illustrate the way that variables interact together within these tables. For any features that seemed even remotely related, we wanted to represent the interaction as its own feature to give the model as much information as possible to learn from. While many of the above created features are fairly straightforward and don't need further explanation as to why they were created, we have further elaborated on some of the less clear ones inside the table. Similarly, the binary variables related to days past due at the bottom of the table were just created to represent whether an installment was past due and whether or not it was under 120 days past due or over 120 days. This variable was broken down into these levels to be able to gather information first on whether payments were late or not and then to gauge the severity of the lateness with the 120 day threshold.

Feature Selection

While both algorithms are capable of generating feature importance outputs, due to the greater efficiency of LightGBM in the context of this problem we conducted the feature selection based on LightGBM's feature importance generator. This was conducted by setting the threshold to 6, which means that all features with an importance lower than 6 were removed. We chose six because, from feature importance distribution, we see that dropping features whose importance is under six can help us reduce a considerable amount of features and also keep the more important features.

Figure 1: Feature Importances


Top20 Feature Importance from Tuned LightGBM
PAYMENT_RATE
EXT_SOURCE_2
EXT_SOURCE_3
EXT_SOURCE_1
DAYS_BIRTH
AMT_ANNUITY
AMT_GOODS_PRICE
AMT_CREDIT
DAYS_EMPLOYED
DAYS_EMPLOYED_PERC
bureau_debt_to_credit_max
bureau_DAYS_CREDIT_ENDDATE_max
DAYS_ID_PUBLISH
ANNUITY_INCOME_PERC
bureau_DAYS_CREDIT_max
bureau_debt_to_credit_sum
bureau_DAYS_ENDDATE_FACT_max
INCOME_CREDIT_PERC
DAYS_LAST_PHONE_CHANGE
client_installments_SK_DPD_max_sum
354.0
256.2
239.6
227.2
180.8
149.0
144.8
120.4
117.4
112.6
108.0
104.8
103.6
97.8
87.6
84.6
82.4
81.8
80.4
80.0
0
50
100
150
200
250
300
350


Feature Importance Distribution
Number of Features
1400
1200
1000
800
600
400
200
0
(0, 5.3]
(5.3, 10.6]
(10.6, 15.9]
(15.9, 21.2]
(21.2, 26.5]
(26.5, 31.8]
(31.8, 37.1]
(37.1, 42.4]
(42.4, 47.7]
(47.7, 53]
(53, 58.3]
(58.3, 63.6]
(63.6, 68.9]
(68.9, 74.2]
(74.2, 79.5]
(79.5, 84.8]
(84.8, 90.1]
Feature Importance

Modeling

Setup

Before being able to proceed with the modeling process, we created a function which helps set up the data to be used in the machine learning techniques. In this step we also prepared for future submission by establishing ids from the test dataset and dropping this from the test set. We additionally used this function to spit up the features and labels or data and target to be able to train the models. Additionally, the training data was split to create a set of validation features and labels as well.

Modeling

Predicting default risk is at its core a classification problem as we are seeking to generate a binary target variable and be able to generate associated probabilities for a client's risk of defaulting. As such, XGBoost and LightGBM classification models were selected for the modeling process due to their robust nature in handling missing values in addition to both algorithms' efficiency and accuracy. Additionally, a stacking method which creates an ensemble effect of the two algorithms was employed. Table 1 below shows an overview of the model performances including best scores based on Kaggle submission scoring.

Table 1: Model Performance Overview

Model Type	**Best Score AUC**
LightGBM	**0.79863**
XGBoost	**0.79530**
Stacking (XGBoost & LightGBM)	**0.79133**

Modeling followed a seemingly iterative process used to find and produce a "Best Model". This consisted of building the model with features and followed by model evaluation. The models were evaluated based on the different sets of features: Main Table, Combined tables with Aggregate and domain features, feature importance based feature selection set, and feature correlation (See Table 2 in "Results" for Corresponding Model Performance).

Base models were first constructed for the two algorithms using hyperparameters as seen in the images below.

XGBoost Base Model

```
# Create the model
model = xgb.XGBClassifier(n_estimators = 1000, objective = "binary:logistic",
          booster = "gbtree",
          eval_metric = "auc",
          nthread = 4,
          eta = 0.05,
          max_depth = 6,
          min_child_weight = 30,
          gamma = 0,
          subsample = 0.75,
          colsample_bytree = 0.6,
          colsample_bylevel = 0.65,
          alpha = 0,
          nrounds = 750)

# Train the model
model.fit(train_features, train_labels, eval_metric = 'auc',
          eval_set = [(valid_features, valid_labels), (train_features, train_labels)],
          early_stopping_rounds = 100, verbose = 200)
```

LightGBM Base Model

```
# Create the model
model = lgb.LGBMClassifier(n_estimators=10000, objective = 'binary',
                           class_weight = 'balanced', learning_rate = 0.05,
                           reg_alpha = 0.1, reg_lambda = 0.1,
                           subsample = 0.8, n_jobs = -1, random_state = 50)

# Train the model
model.fit(train_features, train_labels, eval_metric = 'auc',
          eval_set = [(valid_features, valid_labels), (train_features, train_labels)],
          eval_names = ['valid', 'train'], categorical_feature =  'auto',
          early_stopping_rounds = 100, verbose = 200)
```

As this project was dealing with a large dataset with many features we were dealing with the issue of tuning hyperparameters for each model and being able to produce the best overall model. Grid Search Cross Validation proved to be either too slow or to cause memory usage errors for our computing machines, thus Bayesian Optimization was selected for optimizing and tuning the models as seen below. Ultimately, this was only possible to be performed on the LightGBM model due to technical constraints (RAM constraint and time consuming for XGBoost).

Bayesian Optimization Structure

```
import lightgbm as lgb
from bayes_opt import BayesianOptimization
from sklearn.metrics import roc_auc_score
```

```
def lgb_eval(num_leaves, feature_fraction, bagging_fraction, max_depth, lambda_l1,
             lambda_l2, min_split_gain, min_child_weight):
    params = {'application':'binary','num_iterations':4000, 'learning_rate':0.05,
              'early_stopping_round':100, 'metric':'auc'}
    params["num_leaves"] = round(num_leaves)
    params['feature_fraction'] = max(min(feature_fraction, 1), 0)
    params['bagging_fraction'] = max(min(bagging_fraction, 1), 0)
    params['max_depth'] = round(max_depth)
    params['lambda_l1'] = max(lambda_l1, 0)
    params['lambda_l2'] = max(lambda_l2, 0)
    params['min_split_gain'] = min_split_gain
    params['min_child_weight'] = min_child_weight
    cv_result = lgb.cv(params, train_data, nfold=5, seed=6, stratified=True, verbose_eval =200, metrics=['auc'])
    return max(cv_result['auc-mean'])
```

```
lgbBO = BayesianOptimization(lgb_eval, {'num_leaves': (24, 45),
                                        'feature_fraction': (0.1, 0.9),
                                        'bagging_fraction': (0.8, 1),
                                        'max_depth': (5, 8.99),
                                        'lambda_l1': (0, 5),
                                        'lambda_l2': (0, 3),
                                        'min_split_gain': (0.001, 0.1),
                                        'min_child_weight': (5, 50)}, random_state=0)
```

```
init_round=15
opt_round=25
train_data = lgb.Dataset(data=X_train, label=Y_train,free_raw_data=False)
lgbBO.maximize(init_points=init_round, n_iter=opt_round)
```

The results of the Bayesian Optimization yielded a set of hyperparameters that we were able to use to build a fine-tuned LightGBM model to be evaluated on the different sets of features.

LightGBM (w/ Bayesian Optimization Hyperparameters)

```
# Create the model
model = lgb.LGBMClassifier(
                           n_estimators=10000, objective = 'binary',
                            class_weight = 'balanced', learning_rate = 0.05,
                            reg_alpha = 4.898, reg_lambda = 2.968,
                            max_depth=7,min_child_weight=11.84,
                            min_split_gain=0.03673,num_leaves=31,
                            feature_fraction=0.1926,bagging_fraction=0.8898,
                            subsample = 0.8, n_jobs = -1, random_state = 50)

# Train the model
model.fit(train_features, train_labels, eval_metric = 'auc',
          eval_set = [(valid_features, valid_labels), (train_features, train_labels)],
          eval_names = ['valid', 'train'], categorical_feature = 'auto',
          early_stopping_rounds = 100, verbose = 200)
```

We also tried to employ an ensemble method of the two algorithms by creating a stacking model of XGBoost and LightGBM even using the optimized hyperparameters as seen below.

Stacking Model

```
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression


estimators = [('lightgbm',LGBMClassifier(n_estimators=10000, objective = 'binary',
                                  class_weight = 'balanced', learning_rate = 0.05,
                                  reg_alpha = 4.898, reg_lambda = 2.968, max_depth=7,min_child_weight=11.84,
                                  min_split_gain=0.03673,num_leaves=31,feature_fraction=0.1926,
                                  bagging_fraction=0.8898,
                                  subsample = 0.8, n_jobs = -1, random_state = 50)),
    ("xgb", XGBClassifier(n_estimators = 1000, objective = "binary:logistic",
                  booster = "gbtree",
                  eval_metric = "auc",
                  nthread = 4,
                  eta = 0.05,
                  max_depth = 6,
                  min_child_weight = 30,
                  gamma = 0,
                  subsample = 0.75,
                  colsample_bytree = 0.6,
                  colsample_bylevel = 0.65,
                  alpha = 0,
                  nrounds = 750))]
clf = StackingClassifier(
    estimators=estimators, final_estimator=LogisticRegression()
)
```

Results and Future Steps

Results

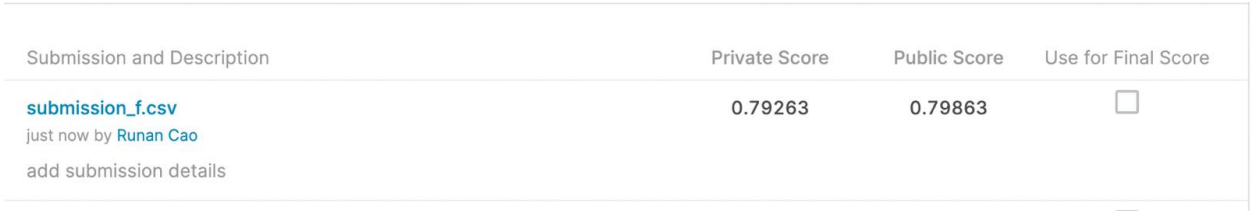
Overall, we saw strong model performance for XGBoost and LightGBM with the best set of results coming with the set of combined tables with domain and aggregate features proving to be the best set of features for producing top performance. The very best model was LightGBM using the hyperparameters provided by Bayesian optimization.

Table 2: Features and Corresponding Model Performance

Performance Score (AUC)	**LightGBM (Basic)**	**LightGBM (Bayesian optimization)**	**XGBoost**
Main Table (241 Features)	**0.74453**	**0.74625**	**0.75083**
Combined Tables w/ Domain and aggregate features (1605 Features)	**0.79336**	**0.79863**	**0.79530**

After Feature importance selection (use LightGBM, threshold=6, 425 Features)	**0.79307**	**0.79674**	**0.79373**
After Feature Correlation selection (Threshold=0.8, 254 Features)	**0.79218**	**0.79551**	**0.79110**

Kaggle Submission Photo Evidence:



Business Insights

As Home Credit seeks to create a more financially-inclusive world, their ability to understand a potential customers' risk of defaulting on their loans. Additionally, the profitability of the company in many ways hinges on their ability to serve customers who are unlikely to default. To this end we believe that some of our features we have created can help Home Credit towards achieving their goals.

- The most heavily weighted feature in our model was the "Payment Rate" feature which captures a ratio of loan annuity to the credit amount of the loan. Using this feature we believe that Home Credit can enhance their ability to assess customers' risk of defaulting.
- This problem can also be seen as a cost imbalanced classification problem, since the cost of making a type1 error and the cost of making a type2 error are different. Identifying a true defaulter as not a defaulter can cause bad debt to the company while also wasting credit resources. Identifying applicants who can repay will as defaulters and rejecting them results in a negative customer experience that reflects on both the retailer and Home Credit. These two costs usually differ. So we can adjust parameters, such as class weights in the models correspondingly. We provided a hypothetical cost and revenue matrix as below, in which we also consider the revenues resulting from correct predictions.
- Following the above discussion, when implementing the model into an unseen dataset without labels, the default cut-off value is 0.5, which means the model classifies each data point with predicted probability larger than cut-off value into 1, and lower than cut-off value into 0. However, 0.5 is not always the best cut-off value. To figure out what is the best cut-off value, we can compute a confusion matrix for each cut-off value and

generate profit curve(in this case, profit=(number of TPs*2400+number of TNs*0)-(number of FPs*12400+number of FNs*2400), and find out the largest profit and best cut-off point.

- High AUC can lead to significant business impact. After all, AUC means randomly choosing a positive and negative instance, the probability of ranking a positive instance over a negative instance. When AUC is high, the probability of ranking the customer with high default risk over the customer with low default risk is high, which means Home Credit can identify customers with high default risk more accurately and reduce unnecessary loss.

Table 2: Hypothetical Cost and Revenue Matrix

		Actual	
		Not default	Default
Predicted	Not default	+2400 Interest revenue Business goodwill	-12400 Bad debt Waste credit resource
	Default	-2400 Bad consumer experience Loss potential revenue	+0 Good risk control

Further Steps

There are some ways to further improve models' performance:

- Fine-tuning the XGBoost model using Bayesian optimization. As we mentioned before, we were not able to finish parameter tuning for XGBoost due to its time consuming and RAM related issues.
- In this case, we tried stacking technique but it didn't boost our performance. To enhance stacking performance, we can try more diverse models with good performance other than boosting models and set 'passthrough=True' to incorporate original training data.

Appendix

Table 3: Hyperparameter setting

LightGBM (Basic)	n_estimators=10000, objective = 'binary', class_weight = 'balanced', learning_rate = 0.05, reg_alpha = 0.1, reg_lambda = 0.1, subsample = 0.8, n_jobs = -1, random_state = 50
LightGBM (Bayesian optimization)	n_estimators=10000, objective = 'binary', class_weight = 'balanced', learning_rate = 0.05, reg_alpha = 4.898, reg_lambda = 2.968, max_depth=7, min_child_weight=11.84, min_split_gain=0.03673, num_leaves=31, feature_fraction=0.1926, bagging_fraction=0.8898, n_jobs = -1, random_state = 50
XGBoost	n_estimators = 1000, objective = "binary:logistic", booster = "gbtree", eval_metric = "auc", nthread = 4, eta = 0.05, max_depth = 6, min_child_weight = 30, gamma = 0, subsample = 0.75, colsample_bytree = 0.6, colsample_bylevel = 0.65, alpha = 0, nrounds = 750