Home Credit Default Risk
Prediction on How Capable Each Applicant is of Repaying a Loan

Zhengyuan Ding(`zd415`)[†], Chenqin Yang(`cy1355`)[†], Ziyu Lei(`zl2350`)[†], Jiayao Liu(`jl9875`)[†], and Zian Chen(`zc674`)[†]

[†]Center for Data Science, New York University

***Abstract*— In this project, we mainly focused on helping Home Credit to predict whether a loan application will default or not. We first did feature engineering and tried out several strategies on dealing with imbalanced dataset including changing evaluation metrics and oversampling using SMOTE algorithm. We finally applied multiple models including Naive Bayes, Support Vector Machine, Random Forest and LightGBM to train a good performing classifier. After doing hyperparameter tuning on LightGBM, we acquired the best classifier with AUC score of 0.7216 and recall score of 0.8229. From the best model, we also extracted top 10 important features that can serve as good indicators on the default prediction.**

I. BACKGROUND

Insufficient credit history, these three words can freeze your finance when you apply to a loan or credit from bank. Many people under this situation often struggle to get loans and are vulnerable to untrustworthy lenders. To help alleviate this issue, Home Credit aims to provide financial support to this underserved population with better borrowing experience. By making use of a variety of alternative data including telco and transactional information, Home Credit hopes to predict their clients' repayment abilities.

II. PROBLEM STATEMENT

Our goal is to use the applicant information to predict how capable each applicant is of repaying a loan, in order to help Home Credit serve the unbanked population safely and successfully. We set this as a supervised binary classification problem using machine learning methods. To optimize the performance efficiency, we would like to apply several methods including Concurrency, Itertools, Numba and MPI etc. in the process of new features generation, data preprocessing and modeling implementation.

III. DATASET DESCRIPTION

A. Dataset

Our dataset mainly comes from Kaggle Competition called "Home Credit Default Risk". Overall, six tables are provided for this project including "application"," bureau", "bureau_balance", "previous application", "credit_card_balance", and "installments_payments". Among them the first four tables with more than 260 variables in total are used most extensively. [1]. "SK_ID_CURR" and "SK_ID_BUREAU" are the keys used when merging these tables. Moreover, our main table is "application_train" with 307511 instances, which includes personal background, income, and housing information about each loan application at Home Credit. Every loan has its own row and identified by the feature "SK_ID_CURR".

[1]Dataset Source: https://www.kaggle.com/c/home-credit-default-risk/data

B. Target Distribution

As is expected, the distribution of target variable where "0" represents "not default" and "1" represents "default" is imbalanced. The ratio of non-default versus default is approximately 11:1 displayed in Figure 1. Besides, we also encounter the problem of missing value in the provided dataset.

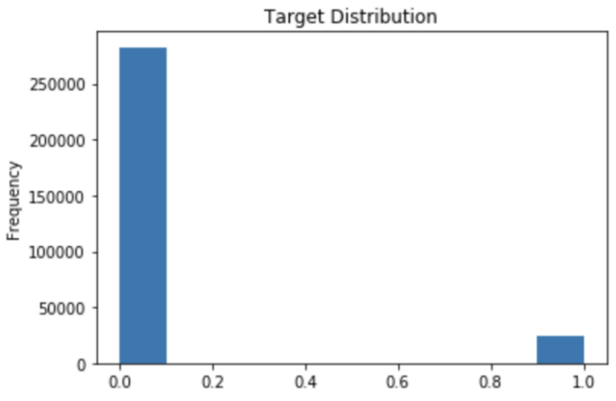


Fig. 1. Target Distribution

IV. DATA PREPROCESSING

A. Missing Values

To start with, we first inspect the categorical and numerical variables based on the data type and description of the columns. We notice that some of the columns with data type of "int64" but actually serve as indicator variables that should be included in categorical groups. After separating different types of columns, we proceed to deal with missing values. Based on the percentage of missing values for each column, we decide to use 60% as cutting threshold to ensure that we won't loss too much information. Visualization for missing value counts in each column are displayed in Figure 2 with horizontal line drawn at 60% of the total number of data points.

Before directly dropping these columns, we also check the correlation between the features and the target variable and notice that though one variable named "EXT_SOURCE_1_" has large portion missing, it is relatively more correlated with the target. Therefore, we decide to keep this variable. The correlation heat map is shown in Figure 3 and Figure 4.

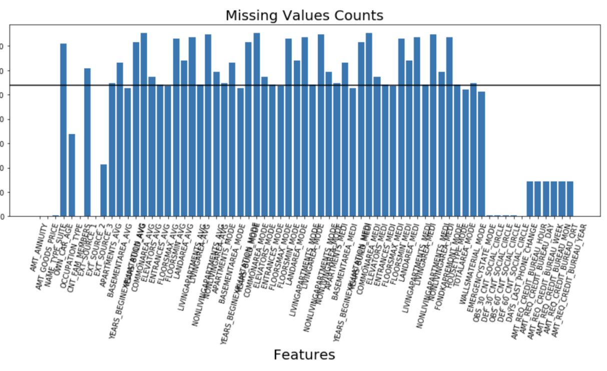


Fig. 2. Missing Value Counts

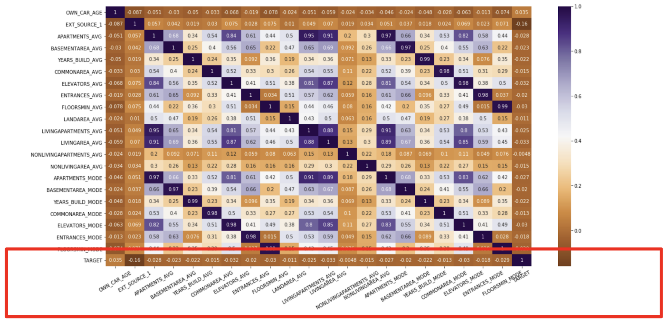


Fig. 3. Correlation Table - 1

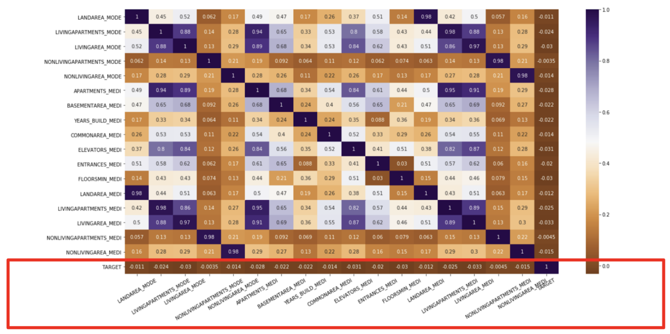
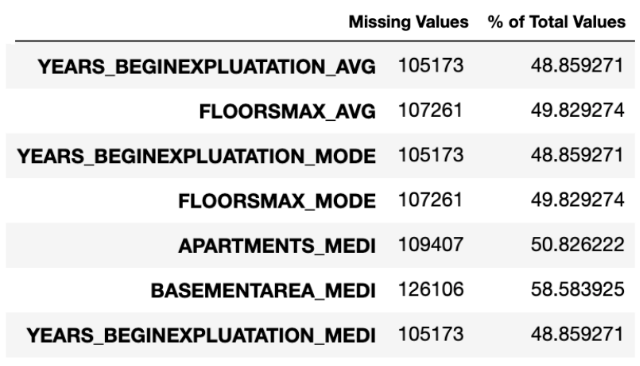


Fig. 4. Correlation Table - 2

After dropping columns with percentage of missing values greater than 60%, we move on to deal with the missing values in the remaining columns. For the remaining numerical features, it is found that the features concerned with housing information mainly consist of median, mean and mode of apartment size, common area, living area, age of building, etc (Figure 5), for which we determine to keep the median values only and drop the mean and mode values. We fill the missing values of the retained median variables with the median values as there exists data skewness problem by checking the histogram of median variables.



	Missing Values	% of Total Values
YEARS_BEGINEXPLUATATION_AVG	105173	48.859271
FLOORSMAX_AVG	107261	49.829274
YEARS_BEGINEXPLUATATION_MODE	105173	48.859271
FLOORSMAX_MODE	107261	49.829274
APARTMENTS_MEDI	109407	50.826222
BASEMENTAREA_MEDI	126106	58.583925
YEARS_BEGINEXPLUATATION_MEDI	105173	48.859271

Fig. 5. Missing values for numerical features

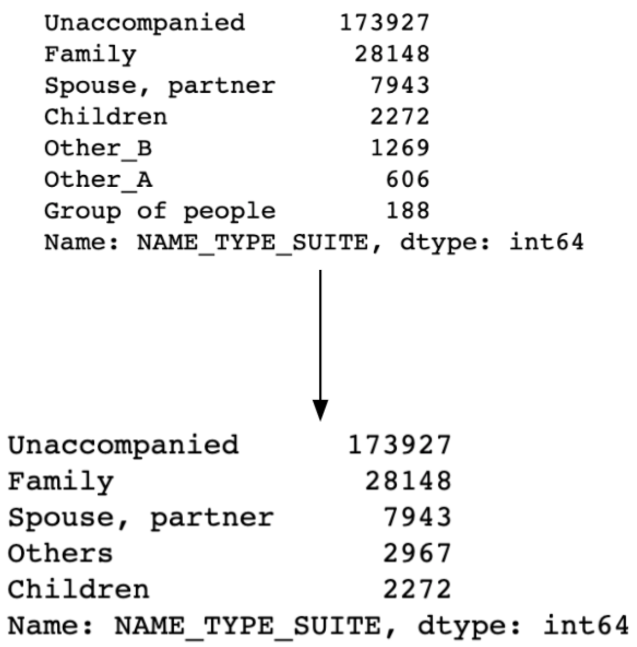
In terms of the categorical features, mainly three features are picked out — NAME_TYPE_SUITE with 904 missing, OCCUPATION_TYPE with 67553 missing and EMERGENCYSTATE with 102265 missing. For the "NAME_TYPE_SUITE" which describes "Who was accompanying client when he/she was applying for the loan", we choose to group the last three minority groups "Other_A", "Other_B", "Group of People" and the missing values and relabel them as "Others" so that it can be more representative (Figure 6). For the "OCCUPATION_TYPE", since a large portion of the values is missing, to avoid losing much information, we decide to flag the missing ones with "Unknown". For the "EMERGENCYSTATE", we decide to drop the column since it has low correlation with the target variable.

B. Dummy Variables

Till this stage, all features now contain zero missing values. Now we can create dummy variables for categorical variables so that they can be directly used during the modeling stage.

C. Anomaly Detection

Before moving to the modeling stage, it is also important to check any anomalies inside the data. For instance, in a variable named "DAYS_EMPLOYED" which represents the total days of employment, some observations take values of 365243. This is more than 1000 years of employment, which is obviously abnormal. Therefore, we flag those cells with 365243 as null and fill them with the column median. The histogram is shown in Figure 7.



```
Unaccompanied      173927
Family              28148
Spouse, partner      7943
Children             2272
Other_B              1269
Other_A               606
Group of people       188
Name: NAME_TYPE_SUITE, dtype: int64
```

```
Unaccompanied      173927
Family              28148
Spouse, partner      7943
Others               2967
Children             2272
Name: NAME_TYPE_SUITE, dtype: int64
```

Fig. 6. Grouping on 'NAME_TYPE_SUITE'

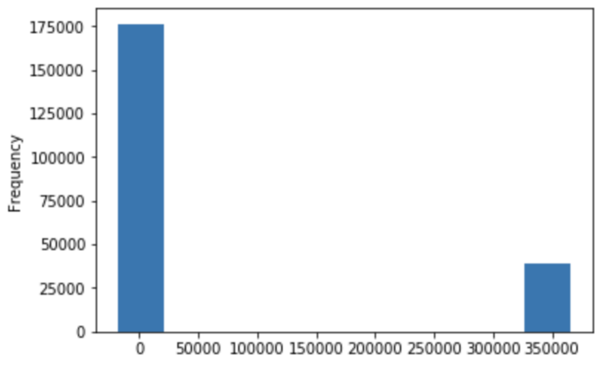


Fig. 7. Histogram of DAYS_EMPLOYED

D. Domain Knowledge Feature Extraction

In addition to the standard preprocessing on the features, we generate some features based on domain knowledge. These features prove to be useful since they rank high in the feature importance and boost the performance of the model.

- Interaction Terms:
  When two variables by themselves may not have a strong influence on the target, interaction terms can capture the interaction between two variables and the combination may show some relationship with the target. Therefore, we created some interaction terms such as (EXT_SOURCE, DAYS_BIRTH) to see whether we can further capture underlying relationship between the predictors and target.
- Groupby Features:
  In the tables such as "bureau.csv" and "previous_application.csv", one loan ID corresponds to several records that contain previous loan information of the clients. Therefore we compute the aggregation values including the sum, mean, min and max grouped by loan ID, which to some extent integrates the feature distributions information of each loan ID into the model.
- Domain Knowledge Features:
  Referring to the popular credit scoring index "FICO score", we generate several features as a measure of how risky the client is. FICO score takes into account five aspects, including payment history and the amount of debt relative to credit limits. Inspired by this, we create several features like "late_payment", "credit_utilization" and "debt_ratio".
  - Late Payment: "late_payment" indicates the delayed days of payment. Specifically, it is the difference between the date that previous credit paid was supposed to be paid and the date that the previous credit was actually paid. The larger is the value of this feature, the more likely this client will default on the loan.
  - Credit Utilization: Credit utilization can be calculated by dividing Credit card balance by credit card limit. It measures how much credit card limit a client has used. It's reasonable to add this feature since a client using up all of his/her credit card limit sounds to be more risky. After experimenting, we find that more recent credit utilization data adds more value to the model. Thus we only calculate the utilization index based on the data of the last 2 months before application. Additionally, since this feature is computed by a division of data, we add 1 to the denominator to smooth the results.
  - Debt Ratio: Debt ratio follows a similar logic as the credit utilization and it's the ratio of total debt to total credit amount.

E. Feature Normalization

In our dataset, we have variables in two digits such as ages and also variables in more than five digits such as income. Since we will apply Support Vector Machine models, it is important for features to be normalized to the same scale.

V. Evaluation Metrics

AUC and Recall, rather than accuracy, are adopted to evaluate the model performance because of the imbalanced distribution of the target variable.

A. AUC

The ROC curve shows the true positive rate (TRP) versus false positive rate (FPR) at various classification thresholds. AUC refers to the area under the receiver operating characteristic curve, which ranges from 0 to 1. The higher the AUC, the better the capability of the model to distinguish between positive and negative class. Additionally, it is also the evaluation metric of the Home Credit Default Risk project.

B. Recall

Considering that the cost of missing a default loan may outweigh the cost of rejecting a potentially legit loan, Recall is also selected as an evaluation metric of model performance. Recall, also called TRP or sensitivity, is defined as $\frac{TP}{(TP+FN)}$. One interpretation of recall is the fraction of relevant instances identified over the total number of relevant instances. In terms of home credit dataset, recall measures the proportion of loans that are predicted to be default out of all default loans.

VI. Modeling

A. Naive Bayes

We use Naive Bayes as our baseline model because it is a simple and fast classifier based on the Bayes Theorem with the assumption of conditional independence between every pair of features given the target value. It classifies a new instance by estimating the probability of this instance belonging to each class and reports the predicted class with the highest probability. Intuitively, the performance of Naive Bayes model is not good enough with the 0.61 AUC score,(Figure 8) since the predictors are not completely independent. For example, clients with higher income are more likely to have higher values of "AMT_INCOME_TOTAL"and "LANDAREA_AVG". Thus, we consider other more reasonable models to improve the performance.

```
AUC score for Naive Bayes is: 0.6100048677235163
```

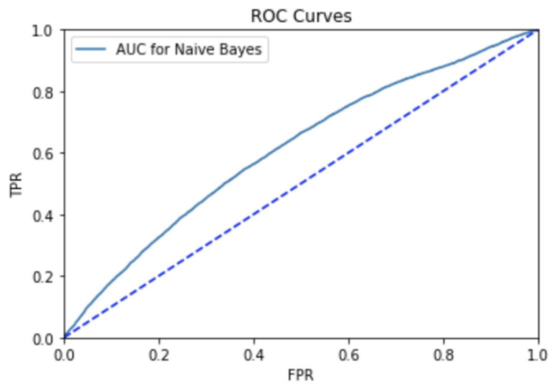


Fig. 8. AUC for Naive Bayes Classifier

B. Support Vector Machine

SVM (Support Vector Machines) is usually a classification method that tries to find a hyperplane in an N-dimensional (n-features) space that can classify the samples. The optimal hyperplane is determined by maximizing the margin which is the minimum distance between data points of different classes to the hyperplane. Hinge loss is extensively selected as the loss function to penalize the wrongly classified samples. We also add a regularization parameter to the cost function to balance the loss and complexity. However, during our implementation, running SVM in Python is too time consuming so we decide to use PySpark for the implementation. Since PySpark now only supports linear kernel, we cannot use polynomial or RBF kernel to well capture non-linear relationships of the variables for classification. The model performance is relatively weak with an AUC Score of 0.58, which is lower than our baseline model Naive Bayes. Thus, in our next step, we will turn to tree-based methods.

C. Random Forest

Random Forest is an ensemble learning algorithm that combines multiple decision trees and averages the results of the decision trees to improve accuracy and reduce variance. When building a decision tree, a sub-sample of training sets are drawn with replacement to fit a tree. Moreover, at each split node, only a subset of features are selected randomly to be split. These two techniques add randomness to forest growing. The following three hyper-parameters have been tuned to improve the performance:

- `n_estimators`: [50, 100, 150, 200, 250]
- `max_depth`: [5,10,20,30]
- `min_samples_split`: [10,20,30,40]

The best hyper-parameter combination is n_estimators=150, max_depth=5, min_samples_split=10 and the corresponding AUC score is 0.71 which is higher than the AUC of Naive Bayes model.
Compared to SVM, we have seen some improvement by random forest which is a parallel ensemble method and next we proceed to explore the capability of sequential ensemble model - LightGBM.

```
AUC score for Random Forest is: 0.7156391040680428
```

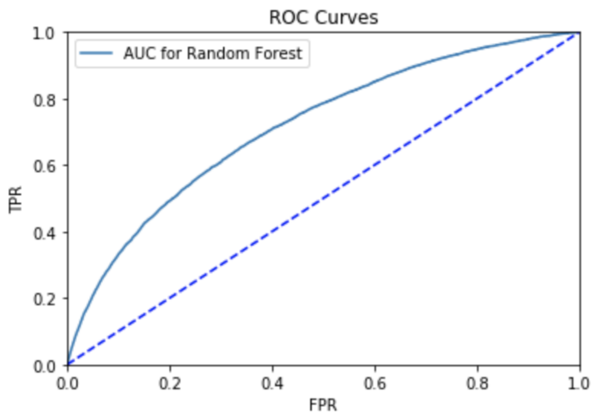


Fig. 9. AUC for Random Forest Classifier

D. LightGBM

LightGBM is a decision-tree based gradient boosting learning algorisms [1]. Compared with traditional gradient boosting methods such as Gradient Boosting Decision Tree(GBDT) and XGBoost, it utilizes the leaf-wise growth strategy so that in each iteration more loss is reduced than level-wise strategy that is adopted in most traditional gradient boosting methods does. In this case, with the same number of leaf splits, it can achieve higher precision [2]. Though it may face the issue of overfitting, we can control it by limiting the depth of the trees it grows. Another advantage of LightGBM is that it supports all data types including categorical without one-hot encoding, which brings faster training speed and higher efficiency. What's more, we can also get the feature importance rankings from tree-based algorithms so that we can help Home Credit figure out what can be strong indicators towards whether the loan will be default or not. However, we also need to point out the disadvantage of LightGBM. Similar to random forest, it is a black-box method that may lack interpretability. Overall, it is a black-box but powerful tool.

E. Hyperparameter Tuning

For the hyperparameter tuning on LightGBM, we mainly choose the following three ones for tuning[2]. We perform the grid search with a total of 125 combinations and retrieve the highest AUC score with its corresponding hyperparameter configurations Figure 10.

- `num_leaves`: [50, 100, 200, 300, 400] - max number of leaves in one tree
- `min_data_in_leaf`: [10, 20, 30, 40, 50] - minimal number of data in one leaf. Can be used to deal with over-fitting
- `max_depth`: [10, 15, 20, 50, 100] - limit the max depth for tree model

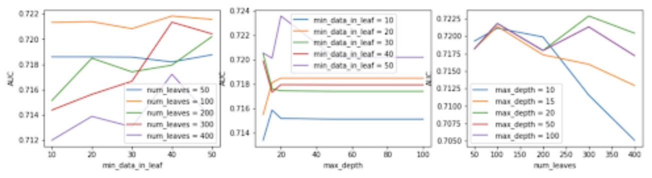


Fig. 10. Hyperparameter Tuning

F. Best Model

The best configuration we get for LightGBM model is in Table I. Under this configuration, the AUC score is 0.7216 and the recall score is 0.8229.

TABLE I
BEST MODEL

Hyperparameters	Values
`min_data_in_leaf`	50
`num_leaves`	100
`max_depth`	20

G. Feature Importance

After training with the best configuration on the training set, we get the top 10 features that have the largest contributions to the model by weighing the number of times each feature present at a splitting node. From the Figure 11, we can see that most of the important features are newly generated features including the debt_ratio and some interaction terms between external sources. And we think the top 10 important features are reasonable indicators for predicting whether a loan will be default or not. For example, applicants with large debt ratio or credit utilization ratio maybe more likely to default while the older the applicants are, the less likely they will default since they may have accumulated savings. Overall, those ten features can be good indicators for Home Credit to take as reference on predicting whether a loan will be default or not.

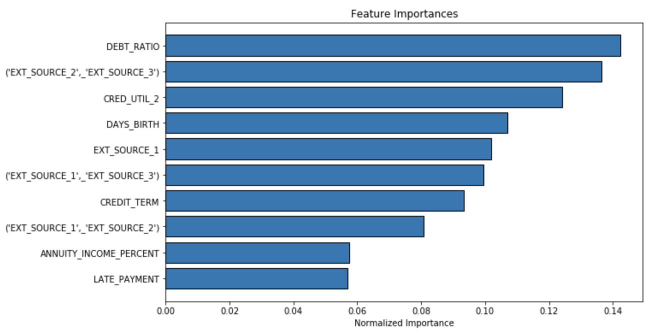


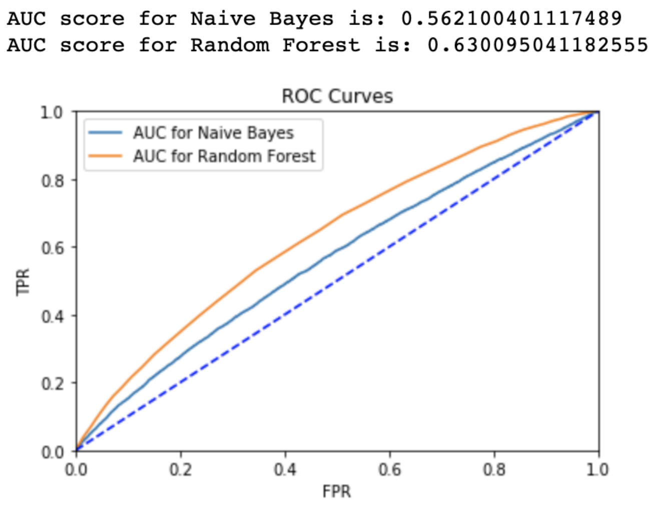
Fig. 11. Top 10 Important Features

VII. SMOTE ALGORITHM

One way to deal with imbalanced data is to choose your evaluation metrics wisely. Accuracy is not a suitable evaluation metrics since a high accuracy around 90% or even bigger may mislead you that the model performs well. However, actually the model only predicts the majority group and sacrifices the loss on the minority group.

Apart from choosing AUC and recall as our evaluation metrics, our team also deploy another strategy when dealing with imbalanced data, which is oversampling. Instead of directly duplicating data points, we choose to use SMOTE algorithm, which stands for Synthetic Minority Over-sampling Technique, to generate new samples. The SMOTE algorithm will take samples of the feature space for each minority target class and its nearest neighbors to generate new examples that combine features of the target case with features of its neighbors. This approach increases the samples available to each class and makes the samples more general [3][4]. The advantage is that it alleviates overfitting caused by random oversampling as synthetic examples are generated rather than replication of instances. However, with the number of features growing, the SMOTE may lose its power on high dimensional data. In our case, the SMOTE strategy doesn't give us much improvement on the model performance. We run the Naïve Bayes and Random Forest model on the resampled dataset and retrieve the AUC results plotted in fig. 12. We reckon that the performance goes down because SMOTE may add much noise in the data when resampling.

[2]For a complete list of hyperparameters of LightGBM, see https://lightgbm.readthedocs.io/en/latest/



```
AUC score for Naive Bayes is: 0.562100401117489
AUC score for Random Forest is: 0.630095041182555
```

Fig. 12. Model Performance After Resampling

VIII. Performance Improving

A. MPI

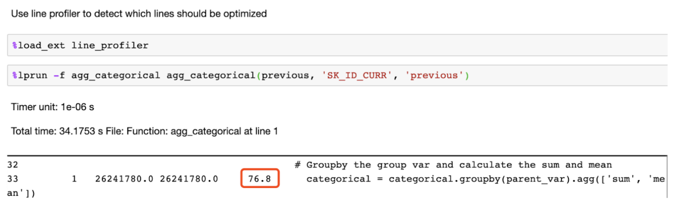
Message Passing Interface which facilitates communication between parallel computers, allows us to exploit multiple processors to solve problems that are too tremendous for traditional serial approaches. In our project, we use MPI to optimize hyper-parameter tuning for random forest models in comparison to for-loop. In one MPI program, each process with a unique rank will be distributed a certain group of hyper-parameter combinations. Each process trains the models with the given combination and returns the hyper-parameter setting which has the best evaluation metrics. The for-loop hyper-parameter tuning takes more than 2 hours while the optimization by MPI reduces the computation time by over 60% (Table II).

TABLE II
Run Time Comparison

For-Loop	MPI(n=8)
7502.6633841991425s greater than 2 hours	2696.423863172531s less than 1 hour

B. Performance tuning: Line_ profiler

Line_profiler can profile given functions, and it will detect the execution time of each individual line inside those functions. In previous_application table, it takes 34 seconds to run the aggregation function for categorical variables, which is kind of slow. Thus, we use line_profiler and find out that 76.8% of running time is spent on the line of groupby and aggregation operation, which should be optimized (Figure 13).



Use line profiler to detect which lines should be optimized

```
%load_ext line_profiler
```

```
%lprun -f agg_categorical agg_categorical(previous, 'SK_ID_CURR', 'previous')
```

Timer unit: 1e-06 s

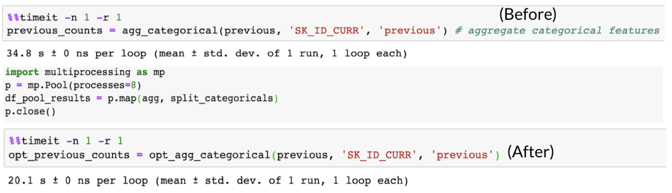
Total time: 34.1753 s File: Function: agg_categorical at line 1

```
32                                                # Groupby the group var and calculate the sum and mean
33        1   26241780.0 26241780.0      76.8       categorical = categorical.groupby(parent_var).agg(['sum', 'me
an'])
```

Fig. 13. Line_Profiler

C. Concurrency (Pools)

As mentioned in the previous part, the aggregation operations on the dataframes consume a relatively longer time shown in the results given by the line profiler. Thus we intend to use multiprocessing methods to improve the efficiency. Basically, the data in the format of dataframe is divided into several parts and distributed to different pools and mapped with the same function. The task in each pool is executed at the same time so that the aggregation proceeds much faster by 42.4%. (Figure 14).



(Before)

```
%%timeit -n 1 -r 1
previous_counts = agg_categorical(previous, 'SK_ID_CURR', 'previous') # aggregate categorical features
```

```
34.8 s ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
```

```
import multiprocessing as mp
p = mp.Pool(processes=8)
df_pool_results = p.map(agg, split_categoricals)
p.close()
```

```
%%timeit -n 1 -r 1
opt_previous_counts = opt_agg_categorical(previous, 'SK_ID_CURR', 'previous')
```

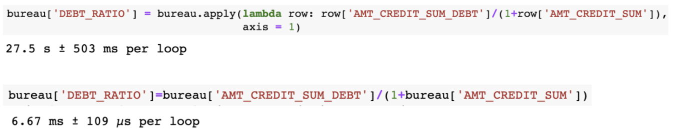
(After)

```
20.1 s ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
```

Fig. 14. Concurrecy Optimization

D. Vectorization in computation

When generating domain features, for example, debt ratio, there are many calculations on elements of two columns. Instead of iterating through each row to compute debt ration, we vectorized the columns and conducted vector computations. As a result, the speed is boosted by 99% (Figure 15).



```
bureau['DEBT_RATIO'] = bureau.apply(lambda row: row['AMT_CREDIT_SUM_DEBT']/(1+row['AMT_CREDIT_SUM']),
                                    axis = 1)
```

```
27.5 s ± 503 ms per loop
```

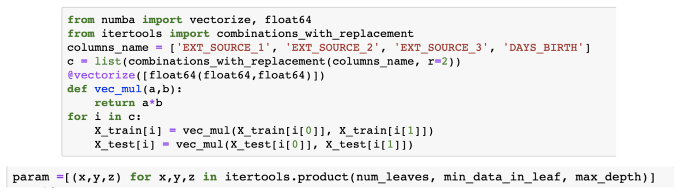
```
bureau['DEBT_RATIO']=bureau['AMT_CREDIT_SUM_DEBT']/(1+bureau['AMT_CREDIT_SUM'])
```

```
6.67 ms ± 109 µs per loop
```

Fig. 15. Vectorization

E. Itertools & Numba (vectorize)

Itertools module provides memory efficient tools without producing intermediate results to improve its performance. It is similar to the lazy evaluation in Spark that no computation is executed until we ask for it. To be concrete, in hyperparameter tuning part, we implement "itertool.product" to get a Cartesian product of three parameters, "num_leaves", "min_data_in_leaf", and "max_depth" instead of using nested loops. In the feature engineering part, we use "itertool.combinations_with_replacement" to get interaction terms and vectorize decorator to write our functions as operating over input scalars rather than arrays. Numba will further optimize the calculation by allowing efficient iterations over the actual inputs (Figure 16). After that, we successfully accelerated our functions with 28.22% improvement.



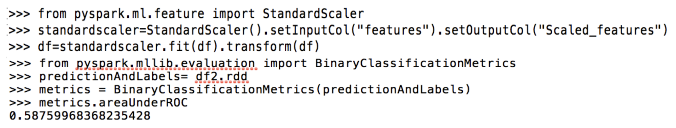
```
from numba import vectorize, float64
from itertools import combinations_with_replacement
columns_name = ['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'DAYS_BIRTH']
c = list(combinations_with_replacement(columns_name, r=2))
@vectorize([float64(float64,float64)])
def vec_mul(a,b):
    return a*b
for i in c:
    X_train[i] = vec_mul(X_train[i[0]], X_train[i[1]])
    X_test[i] = vec_mul(X_test[i[0]], X_test[i[1]])
```

```
param =[(x,y,z) for x,y,z in itertools.product(num_leaves, min_data_in_leaf, max_depth)]
```

Fig. 16. Itertools & Numba Vectorize Decorator

F. *PySpark*

Pyspark works as an alternative tool to implement the SVM model because of the extremely slow model fitting process in Python. There is MlLib embedded in PySpark and we would like to test how the SVM model with linear kernel(which is the only option in the Spark MlLib library). We transform the feature variables into "double" type and target variable into "integer" type. Then by using vector_assembler, all the feature variables are encapsulated into one vector. And we use the LinearSVC package for model fitting. We run the SVM model for more than 4 hours in Python but fail to retrieve the result. On PySpark platform, We successfully got the result after 50 minutes. (Figure 17)



```
>>> from pyspark.ml.feature import StandardScaler
>>> standardscaler=StandardScaler().setInputCol("features").setOutputCol("Scaled_features")
>>> df=standardscaler.fit(df).transform(df)
>>> from pyspark.mllib.evaluation import BinaryClassificationMetrics
>>> predictionAndLabels= df2.rdd
>>> metrics = BinaryClassificationMetrics(predictionAndLabels)
>>> metrics.areaUnderROC
0.58759968368235428
```

Fig. 17. PySpark LinearSVM

IX. CONCLUSIONS

Among the three models we've tried including Naive Bayes, SVM, random forest and LightGBM, LightGBM has the best performance regarding the AUC and recall score. When we rank the most important features according to our best model, features that are typically used in credit scoring, like debt ratio and credit utilization, are proved to be important in default prediction in the Home Credit case as well. In terms of performance optimization, methods we tried all achieved a significant speed up.

Future work of this project may include exploring more features and potential optimization methods. For the preprocessing, we may build models to predict those missing values as the missing pattern seems not to be random. Last but not least, the evaluation metrics can be combined with cost analysis if the related information is available.

REFERENCES

[1] Ke, Guolin and Meng, Qi and Finley, Thomas and Wang, Taifeng and Chen, Wei and Ma, Weidong and Ye, Qiwei and Liu, Tie-Yan, Lightgbm: A highly efficient gradient boosting decision tree, Advances in Neural Information Processing Systems, 2017, pp.3146–3154

[2] Meng, Qi and Ke, Guolin and Wang, Taifeng and Chen, Wei and Ye, Qiwei and Ma, Zhi-Ming and Liu, Tie-Yan, A communication-efficient parallel algorithm for decision tree, Advances in Neural Information Processing Systems, 2016, pp.1279–1287

[3] How to handle Imbalanced Classification Problems in machine learning? (https://www.analyticsvidhya.com/blog/2017/03/imbalanced-classification-problem/)

[4] SMOTE Documentation https://docs.microsoft.com/bs-latn-ba/azure/machine-learning/studio-module-reference/smote

[5] Precision vs Recall (https://towardsdatascience.com/precision-vs-recall-386cf9f89488)

[6] Lecture Slide from DSGA-1001 "Introduction to Data Science"