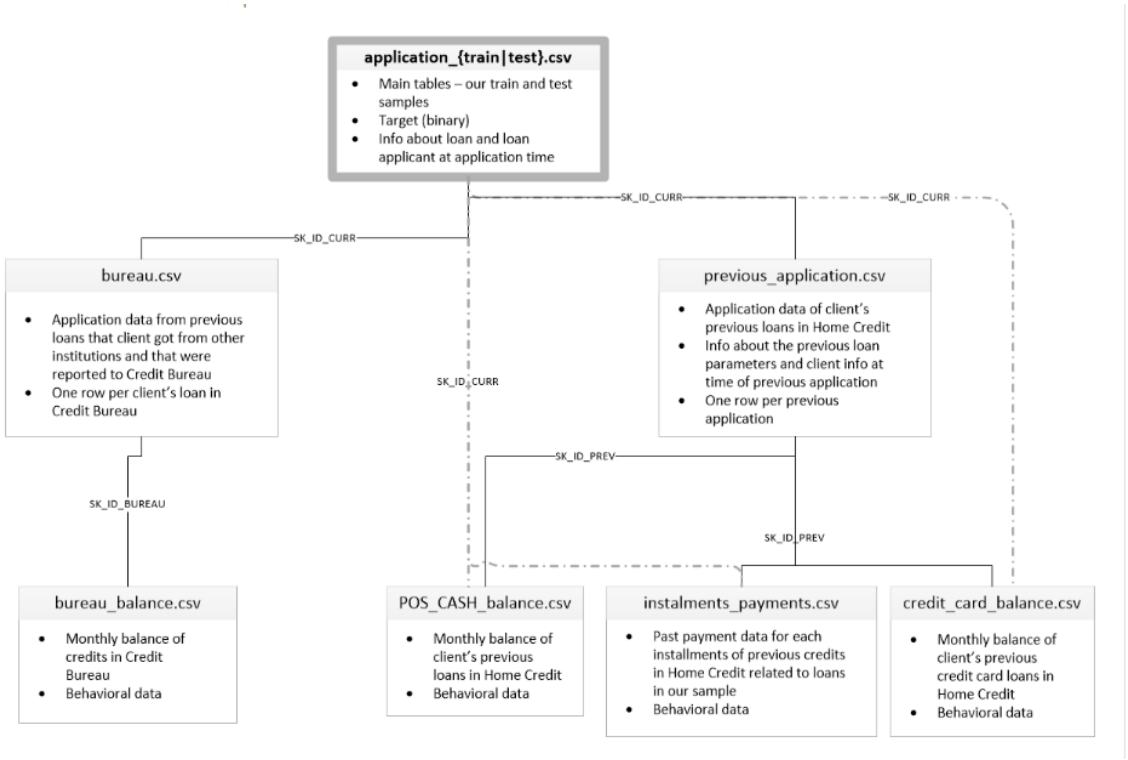
Home Credit Default Risk

By Jean-Philippe Jacques, Bhargav Parekh, David-Alain Jean-Baptiste and Vivudh Prashar

For the Home Credit project, we used various statistical and machine learning methods to predict clients' repayment abilities. Doing so will ensure that clients capable of repayment are not rejected and that loans with a principal, maturity, and repayment calendar that will empower their clients to be successful are given. The data is divided in 7 different files as it is shown in this flow chart. This is a challenging dataset because of the size and the division of the data that need many different aggregations by clients and merge by clients. The Feature engineering part is also very challenging because of the amount of data and the almost unlimited number of new features that we can create by combining two pieces of information or more. The training part was also very difficult mostly because of all the time it took to get the end result. The best ROC we reach was 0.7934 using LightGBM.



Data Processing

For processing and merging the 7 different files and turning them into more than 3000 feature we proceeded with the following step :

1-For the object columns, null values were replaced by unknow. We created dummies variables for features with less than 20 different unique values which were transformed to int with the help of LabelEncoder from the sklearn.preprocessing module.

2-For numerical features, null values were replaced by the median value of the column.

3-We aggregated every object by ['max', 'min' , ' nunique '] and every numerical column by ['max','sum','mean']

4-For the time series data frame we created different time windows for the aggregation: the last month, the last six months, the month one year ago and the difference between last month and one year ago for the trend.

5-We merged credit_card_balance, installments_payments and POS_CASH_balance with previous_application and we merged bureau_balance with df_bureau.

6-We aggregated df_bureau and previous_application by ['max','sum','nunique'] and finally merged it with application_train.

7-All the Data processing cleaning encoding aggregation and the merging were put in two loops to avoid code repetition.

With more computer power we would have tried to add more aggregations and more dummies variable to improve our model.

Feature Engineering

To improve the model, ee add many new features to our dataset using brain and brute force feature engineering with a super loop that test 1 250 000 new feature to extract the 625 best. That cost us a lot of time. Many of the most important features in LGBM are features we created with operation + - * / between two columns,

```
1 feature_importance.sort_values('mean', ascending = False).head(20)
```

	0	1	2	3	4	mean
app_AMT_CREDIT_div_AMT_ANNUITY	2560	2566	2573	2753	2793	2649.0
df_application_train_DAYS_ID_PUBLISH	1554	1515	1418	1819	1720	1605.2
df_bureau_DAYS_CREDITsum_-_df_bureau_DAYS_ENDDATE_FACTsum	1317	1273	1299	1464	1471	1364.8
df_application_train_AMT_INCOME_TOTAL_/_12_-_df_application_train_AMT_ANNUITY	1295	1392	1225	1436	1459	1361.4
df_application_train_DAYS_REGISTRATION	1134	1346	974	1523	1474	1290.2
df_application_train_AMT_INCOME_TOTAL_*_df_application_train_AMT_ANNUITY	1042	1357	1167	1314	1368	1249.6
df_application_train_AMT_INCOME_TOTAL_/_df_application_train_AMT_ANNUITY	1104	1157	1122	1426	1292	1220.2
df_bureau_DAYS_CREDITmax	1271	1094	968	1342	1406	1216.2
app_EXT_SOURCE_1_div_DAYS_BIRTH	1195	1198	1048	1324	1278	1208.6
df_POS_CASH_balance_CNT_INSTALMENT_FUTUREmeanmax	1159	1138	1184	1226	1292	1199.8
df_bureau_AMT_CREDIT_SUMsum_-_df_bureau_AMT_CREDIT_SUM_DEBTsum	1056	1108	1083	1245	1203	1139.0
df_application_train_OWN_CAR_AGE_/_df_application_train_DAYS_EMPLOYED	980	1007	1110	1183	1221	1100.2
df_application_train_OWN_CAR_AGE_/_df_application_train_DAYS_BIRTH	1073	1135	959	1263	1066	1099.2
df_application_train_DAYS_EMPLOYED_-_df_application_train_DAYS_BIRTH	900	1104	986	1212	1210	1082.4
df_application_train_DAYS_LAST_PHONE_CHANGE	1025	1121	898	1124	1165	1066.6
df_application_train_AMT_CREDIT_*_df_application_train_AMT_ANNUITY	891	967	943	1176	1230	1041.4
df_application_train_DAYS_BIRTH	944	1052	937	1147	1068	1029.6
df_previous_application_AMT_APPLICATIONsum_/_df_previous_application_AMT_CREDITsum	892	1020	908	1029	1106	991.0
df_bureau_DAYS_CREDIT_ENDDATEmax	866	1065	881	1032	926	954.0
df_bureau_DAYS_ENDDATE_FACTmax	887	927	744	1082	1005	929.0

We build those new features with the help of our financial knowledge, some inspiration from Kaggle and with the help of a loop that test a lot of new features from operation between two columns. This loop uses SelectKbest from the sklearn.feature package. At each round of the loop, with the help of SelectKBest, the best features created were selected.

```
for opp in range(1,6):
    print(opp)
    i=0

    for c in var[:-1]:
        i=i+1
        Newfeat=pd.DataFrame()
        for cc in var[i:]:
            if opp ==1:
                Newfeat[c+"_plus_"+cc]=X_train[c]+X_train[cc]
            if opp ==2:
                Newfeat[c+"_minus_"+cc]=X_train[c]-X_train[cc]
            if opp ==3:
                Newfeat[c+"_multi_"+cc]=X_train[c]*X_train[cc]
            if opp ==4:
                Newfeat[c+"_div_"+cc]=X_train[c]/X_train[cc]
            if opp ==5:
                Newfeat[c+"_div_"+cc]=X_train[c]/X_train[cc]
        if i > 1:
            Newfeat=pd.concat([Newfeat,bestfeat],axis=1)

        Newfeat=Newfeat.replace(-np.Inf, 0)
        Newfeat=Newfeat.replace(np.Inf, 0)
        Newfeat=Newfeat.replace(np.nan, 0)
        fvalue_selector = SelectKBest(f_classif, k=min(Newfeat.shape[1],100))
        FX_kbest = fvalue_selector.fit_transform(Newfeat, y_train)
```

LGBM training

To train our model we used LightLGBM. The best ROC we reach was 0.7934 using LightGBM with the data split into 5 different folds.

```
Starting LightGBM. Train shape: (246008, 1814), test shape: (61503, 1814)
Fold 0 started at Tue Mar 24 07:05:17 2020
Training until validation scores don't improve for 500 rounds
Early stopping, best iteration is:
[1733]  training's auc: 0.913409        training's binary_logloss: 0.185214     valid_1's auc: 0.783141 valid_1's binary_loglos
s: 0.238398
Fold  0 AUC : 0.783141
Fold 1 started at Tue Mar 24 07:37:23 2020
Training until validation scores don't improve for 500 rounds
Early stopping, best iteration is:
[1925]  training's auc: 0.92016 training's binary_logloss: 0.181904     valid_1's auc: 0.785013 valid_1's binary_logloss: 0.236
699
Fold  1 AUC : 0.785013
Fold 2 started at Tue Mar 24 08:11:09 2020
Training until validation scores don't improve for 500 rounds
Early stopping, best iteration is:
[1735]  training's auc: 0.91287 training's binary_logloss: 0.185393     valid_1's auc: 0.789949 valid_1's binary_logloss: 0.237
751
Fold  2 AUC : 0.789949
Fold 3 started at Tue Mar 24 08:42:35 2020
Training until validation scores don't improve for 500 rounds
Early stopping, best iteration is:
[2402]  training's auc: 0.933526        training's binary_logloss: 0.174807     valid_1's auc: 0.793448 valid_1's binary_loglos
s: 0.233996
Fold  3 AUC : 0.793448
Fold 4 started at Tue Mar 24 09:22:09 2020
Training until validation scores don't improve for 500 rounds
Early stopping, best iteration is:
[2226]  training's auc: 0.930398        training's binary_logloss: 0.176667     valid_1's auc: 0.789931 valid_1's binary_loglos
s: 0.235874
Fold  4 AUC : 0.789931
Full AUC score 0.788298
```

To improve the LGBM model, we tried to optimize LGBM parameters with two different tools. We tried Optuna from optuna.org and BayesianOptimization from https://github.com/fmfn/BayesianOptimization. Optuna was harder to adjust and slower and lead us to lower ROC than BayesianOptimization.

```
params = {'colsample_bytree': (0.75, 1),
          'learning_rate': (.005, .025),
          'num_leaves': (25, 200),
          'subsample': (0.75, 1),
          'max_depth': (10, 25),
          'reg_alpha': (.025, .05),
          'reg_lambda': (.055, .08),
          'min_split_gain': (.01, .05),
          'min_child_weight': (20, 60),
          'max_bin': (100,2000),
          'n_estimators': (5000,50000),
          'feature_fraction':( 0.5,1),
          'bagging_fraction':(0.5,1),
          'bagging_freq': (3,10),
         }
```

```
gc.collect()
bo = BayesianOptimization(lgbm_evaluate, params)
bo.maximize(init_points = 5 , n_iter =5)
```

```
   iter    |  target   | baggin... | baggin... | colsam... | featur... | learni... |  max_bin  | max_depth | min_ch... | min_
sp... | n_esti... | num_le... | reg_alpha | reg_la... | subsample |
-------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------
   1       |  0.7836   |  0.6664   |  4.461    |  0.7529   |  0.6578   |  0.01903  |  1.554e+0 |  19.96    |  57.57    |  0.0
1933  |  2.594e+0 |  99.76    |  0.0404   |  0.07247  |  0.7851   |
   2       |  0.7838   |  0.8329   |  9.577    |  0.9349   |  0.8015   |  0.005356 |  1.362e+0 |  20.75    |  29.99    |  0.0
195   |  2.391e+0 |  157.5    |  0.02868  |  0.05651  |  0.7699   |
   3       |  0.785    |  0.9848   |  4.629    |  0.8495   |  0.6351   |  0.02136  |  1.226e+0 |  13.63    |  29.47    |  0.0
1705  |  6.253e+0 |  28.41    |  0.04625  |  0.06432  |  0.9251   |
   4       |  0.7841   |  0.5578   |  8.079    |  0.8047   |  0.7612   |  0.01336  |  1.148e+0 |  23.28    |  54.14    |  0.0
1357  |  1.074e+0 |  149.9    |  0.04757  |  0.06873  |  0.9163   |
```

Most of our training code came from https://www.kaggle.com/aantonova/797-lgbm-and-bayesian-optimization we made some adjustment to the parameters and the code to adapt it to the bayesian-optimization libraries update, but most of it is very similar and can be used to many other cases. It's a great LGBM pipeline.

Conclusion

To conclude, the main challenge that we faced during the project wasn't a coding one, there was no logical reasoning needed to be solved, it was more about our technology resources. That made us realize that the resources that are available to do a machine learning project are almost as important as the technical skills required to realize it. No matter how high is the level of your skills, the available resources will determine how far you will be able to go. Our results are good, but we think that they could have been much better with more computer power. we were very creative in our feature engineering, but memory space limited us to expend above 2000 features. The optimization of LGBM was also limited by the speed of the process and the memory. We recommend using a new generation CPU of 8 core at least and 32 Go of memory to run our code smoothly.