Home Credit Project

April 19, 2022

Reference:

https://www.kaggle.com/code/willkoehrsen/introduction-to-manual-feature-engineering/notebook https://www.kaggle.com/code/willkoehrsen/introduction-to-manual-feature-engineering-p2/notebook https://www.kaggle.com/c/santander-customer-transaction-prediction/discussion/81289 https://www.kaggle.com/c/home-credit-default-risk/discussion/64821 https://analyticsindiamag.com/implementing-bayesian-optimization-on-xgboost-a-beginners-guide/

[ ]:
```
# numpy and pandas for data manipulation
import numpy as np
import pandas as pd

# sklearn preprocessing for dealing with categorical variables
from sklearn.preprocessing import LabelEncoder

# File system manangement
import os

# Suppress warnings
import warnings
warnings.filterwarnings('ignore')

# matplotlib and seaborn for plotting
import matplotlib.pyplot as plt
import seaborn as sns
import gc

import lightgbm as lgb
from sklearn.model_selection import KFold
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import LabelEncoder

import lightgbm as lgb
!pip install bayesian-optimization
from bayes_opt import BayesianOptimization
from sklearn.metrics import roc_auc_score
#Importing necessary libraries
```

```
! pip install bayesian-optimization
from bayes_opt import BayesianOptimization
import xgboost as xgb
from sklearn.metrics import roc_auc_score
import xgboost as xgb
```

1 Prepare Data

[ ]:
```
train = pd.read_csv('application_train.csv')
test = pd.read_csv('application_test.csv')
```

[ ]:
```
total = train.append(test)
```

[ ]:
```
total['TARGET'].isnull().sum()
```

[ ]:
```
48744
```

[ ]:
```
total=total.reset_index(drop=True)
```

[ ]:
```
train['TARGET'].value_counts()
```

[ ]:
```
0    282686
1     24825
Name: TARGET, dtype: int64
```

1.1 Missing values

[ ]:
```
def missing_values_table(df):
        # Total missing values
        mis_val = df.isnull().sum()

        # Percentage of missing values
        mis_val_percent = 100 * df.isnull().sum() / len(df)

        # Make a table with the results
        mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)

        # Rename the columns
        mis_val_table_ren_columns = mis_val_table.rename(
        columns = {0 : 'Missing Values', 1 : '% of Total Values'})

        # Sort the table by percentage of missing descending
        mis_val_table_ren_columns = mis_val_table_ren_columns[
            mis_val_table_ren_columns.iloc[:,1] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)
        # Print some summary information
        print ("Your selected dataframe has " + str(df.shape[1]) + " columns.\n"
```

```
            "There are " + str(mis_val_table_ren_columns.shape[0]) +
              " columns that have missing values.")

        # Return the dataframe with missing information
        return mis_val_table_ren_columns
```

[ ]:
```
missing_values = missing_values_table(train)
missing_values.head(20)
```

```
Your selected dataframe has 122 columns.
There are 67 columns that have missing values.
```

[ ]:

	Missing Values	% of Total Values
COMMONAREA_MEDI	214865	69.9
COMMONAREA_AVG	214865	69.9
COMMONAREA_MODE	214865	69.9
NONLIVINGAPARTMENTS_MEDI	213514	69.4
NONLIVINGAPARTMENTS_MODE	213514	69.4
NONLIVINGAPARTMENTS_AVG	213514	69.4
FONDKAPREMONT_MODE	210295	68.4
LIVINGAPARTMENTS_MODE	210199	68.4
LIVINGAPARTMENTS_MEDI	210199	68.4
LIVINGAPARTMENTS_AVG	210199	68.4
FLOORSMIN_MODE	208642	67.8
FLOORSMIN_MEDI	208642	67.8
FLOORSMIN_AVG	208642	67.8
YEARS_BUILD_MODE	204488	66.5
YEARS_BUILD_MEDI	204488	66.5
YEARS_BUILD_AVG	204488	66.5
OWN_CAR_AGE	202929	66.0
LANDAREA_AVG	182590	59.4
LANDAREA_MEDI	182590	59.4
LANDAREA_MODE	182590	59.4

1.2 Noisy data

[ ]:
```
(train['DAYS_BIRTH'] / -365).describe()
```

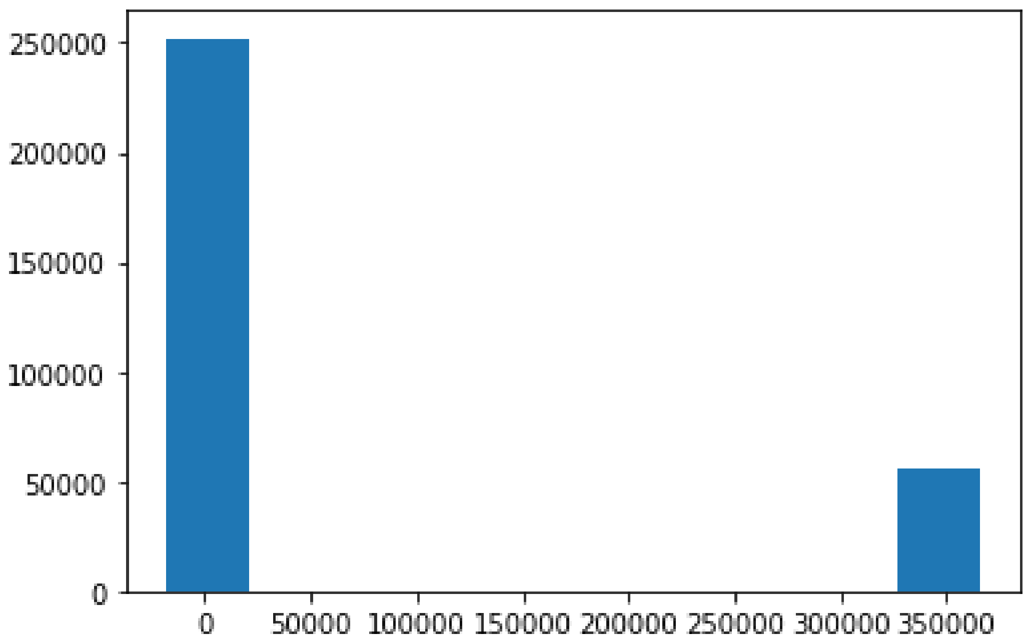
[ ]:
```
count    307511.000000
mean         43.936973
std          11.956133
min          20.517808
25%          34.008219
50%          43.150685
75%          53.923288
max          69.120548
Name: DAYS_BIRTH, dtype: float64
```

[ ]:
```
train['DAYS_EMPLOYED'].describe()
```

[ ]:
```
count    307511.000000
mean      63815.045904
std      141275.766519
min      -17912.000000
25%       -2760.000000
50%       -1213.000000
75%        -289.000000
max      365243.000000
Name: DAYS_EMPLOYED, dtype: float64
```

[ ]:
```
plt.hist(train['DAYS_EMPLOYED'])
```

[ ]:
```
(array([252137.,      0.,      0.,      0.,      0.,      0.,      0.,
             0.,      0.,  55374.]),
 array([-17912. ,  20403.5,  58719. ,  97034.5, 135350. , 173665.5,
        211981. , 250296.5, 288612. , 326927.5, 365243. ]),
 <BarContainer object of 10 artists>)
```



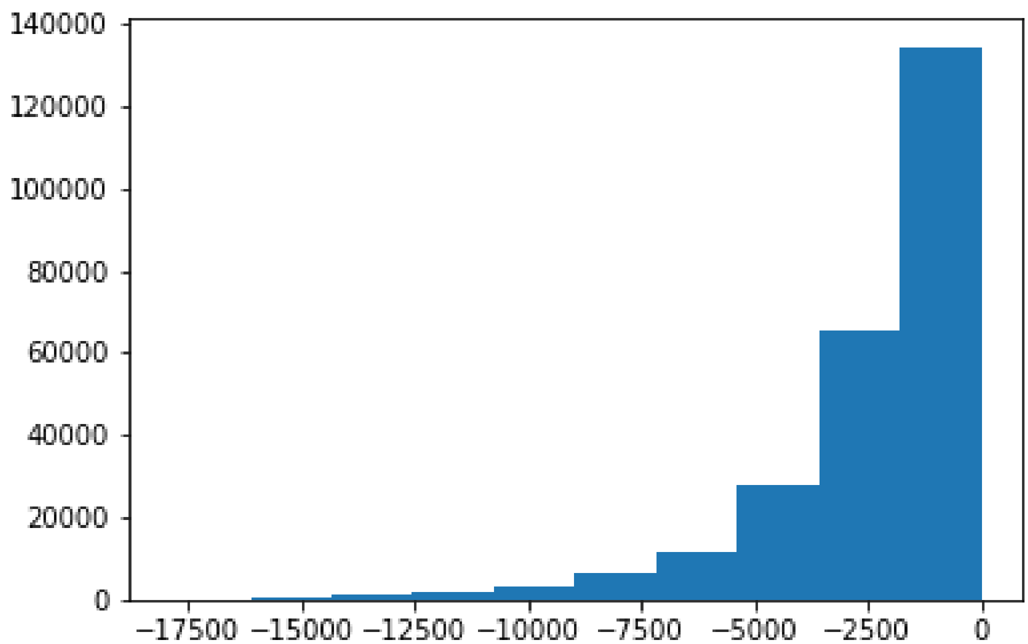
There are extremely large number: 365243 days, which is impossible.

[ ]:
```
# add tage to the anomaly

total['DAYS_EMPLOYED_ANOM'] = total["DAYS_EMPLOYED"] == 365243
# Replace the anomalous values with nan
```

```
total['DAYS_EMPLOYED'].replace({365243: np.nan}, inplace = True)
```

[ ]:
```
train['DAYS_EMPLOYED'].replace({365243: np.nan}, inplace = True)
plt.hist(train['DAYS_EMPLOYED'])
```

[ ]:
```
(array([5.00000e+01, 2.61000e+02, 9.55000e+02, 2.02700e+03, 3.38700e+03,
        6.56300e+03, 1.13820e+04, 2.79320e+04, 6.51880e+04, 1.34392e+05]),
 array([-17912. , -16120.8, -14329.6, -12538.4, -10747.2,  -8956. ,
         -7164.8,  -5373.6,  -3582.4,  -1791.2,      0. ]),
 <BarContainer object of 10 artists>)
```



[ ]:
```
train['DAYS_REGISTRATION'].describe()
```

[ ]:
```
count    307511.000000
mean      -4986.120328
std        3522.886321
min      -24672.000000
25%       -7479.500000
50%       -4504.000000
75%       -2010.000000
max           0.000000
Name: DAYS_REGISTRATION, dtype: float64
```

[ ]:
```
train['DAYS_ID_PUBLISH'].describe()
```

```
[ ]: count    307511.000000
     mean      -2994.202373
     std        1509.450419
     min       -7197.000000
     25%       -4299.000000
     50%       -3254.000000
     75%       -1720.000000
     max           0.000000
     Name: DAYS_ID_PUBLISH, dtype: float64
```

```
[ ]: train['OWN_CAR_AGE'].describe()
```

```
[ ]: count    104582.000000
     mean         12.061091
     std          11.944812
     min           0.000000
     25%           5.000000
     50%           9.000000
     75%          15.000000
     max          91.000000
     Name: OWN_CAR_AGE, dtype: float64
```

```
[ ]: train['CNT_FAM_MEMBERS'].describe()
```

```
[ ]: count    307509.000000
     mean          2.152665
     std           0.910682
     min           1.000000
     25%           2.000000
     50%           2.000000
     75%           3.000000
     max          20.000000
     Name: CNT_FAM_MEMBERS, dtype: float64
```

2 Transform Data

2.1 One-hot encoding of categorical variables

```
[ ]: total = pd.get_dummies(total)
```

```
[ ]: total.head()
```

```
[ ]:    SK_ID_CURR  TARGET  CNT_CHILDREN  AMT_INCOME_TOTAL  AMT_CREDIT  \
     0      100002     1.0             0          202500.0    406597.5
     1      100003     0.0             0          270000.0   1293502.5
     2      100004     0.0             0           67500.0    135000.0
     3      100006     0.0             0          135000.0    312682.5
```

```
4      100007     0.0             0          121500.0    513000.0

   AMT_ANNUITY  AMT_GOODS_PRICE  REGION_POPULATION_RELATIVE  DAYS_BIRTH  \
0      24700.5         351000.0                    0.018801       -9461
1      35698.5        1129500.0                    0.003541      -16765
2       6750.0         135000.0                    0.010032      -19046
3      29686.5         297000.0                    0.008019      -19005
4      21865.5         513000.0                    0.028663      -19932

   DAYS_EMPLOYED  …  HOUSETYPE_MODE_terraced house  \
0         -637.0  …                              0
1        -1188.0  …                              0
2         -225.0  …                              0
3        -3039.0  …                              0
4        -3038.0  …                              0

   WALLSMATERIAL_MODE_Block  WALLSMATERIAL_MODE_Mixed  \
0                         0                         0
1                         1                         0
2                         0                         0
3                         0                         0
4                         0                         0

   WALLSMATERIAL_MODE_Monolithic  WALLSMATERIAL_MODE_Others  \
0                              0                          0
1                              0                          0
2                              0                          0
3                              0                          0
4                              0                          0

   WALLSMATERIAL_MODE_Panel  WALLSMATERIAL_MODE_Stone, brick  \
0                         0                                1
1                         0                                0
2                         0                                0
3                         0                                0
4                         0                                0

   WALLSMATERIAL_MODE_Wooden  EMERGENCYSTATE_MODE_No  EMERGENCYSTATE_MODE_Yes
0                          0                       1                        0
1                          0                       1                        0
2                          0                       0                        0
3                          0                       0                        0
4                          0                       0                        0

[5 rows x 247 columns]
```

3 Exploratory Data Analysis

[ ]:
```
correlations = train.corr()['TARGET'].sort_values()
```

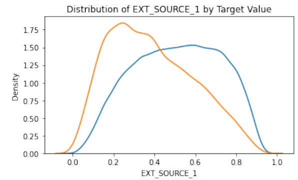
[ ]:
```
correlations.head(5)
```

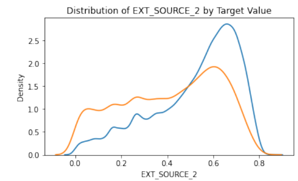
[ ]:
```
EXT_SOURCE_3     -0.178919
EXT_SOURCE_2     -0.160472
EXT_SOURCE_1     -0.155317
FLOORSMAX_AVG    -0.044003
FLOORSMAX_MEDI   -0.043768
Name: TARGET, dtype: float64
```

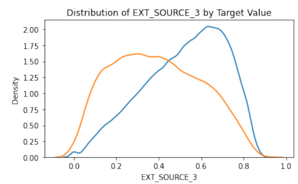
[ ]:
```
correlations.tail(6)
```

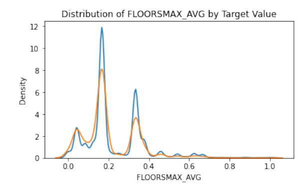
[ ]:
```
DAYS_LAST_PHONE_CHANGE          0.055218
REGION_RATING_CLIENT            0.058899
REGION_RATING_CLIENT_W_CITY     0.060893
DAYS_EMPLOYED                   0.074958
DAYS_BIRTH                      0.078239
TARGET                          1.000000
Name: TARGET, dtype: float64
```

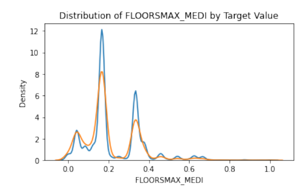
[ ]:
```
plt.figure(figsize = (6, 36))

# iterate through the sources
for i, source in enumerate(['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3',
                            'FLOORSMAX_AVG','FLOORSMAX_MEDI',
                            'DAYS_LAST_PHONE_CHANGE','REGION_RATING_CLIENT',
                           ␣
 ↪'REGION_RATING_CLIENT_W_CITY','DAYS_EMPLOYED','DAYS_BIRTH']):

    # create a new subplot for each source
    plt.subplot(10, 1, i + 1)
    # plot repaid loans
    sns.kdeplot(train.loc[train['TARGET'] == 0, source], label = 'target == 0')
    # plot loans that were not repaid
    sns.kdeplot(train.loc[train['TARGET'] == 1, source], label = 'target == 1')

    # Label the plots
    plt.title('Distribution of %s by Target Value' % source)
    plt.xlabel('%s' % source); plt.ylabel('Density');

plt.tight_layout(h_pad = 2.5)
```

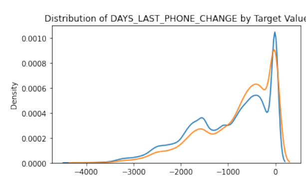

Distribution of EXT_SOURCE_1 by Target Value
Density
EXT_SOURCE_1

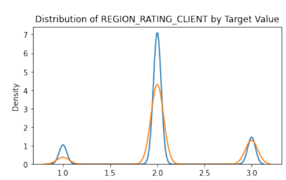

Distribution of EXT_SOURCE_2 by Target Value
Density
EXT_SOURCE_2

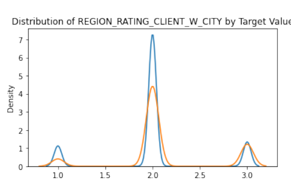

Distribution of EXT_SOURCE_3 by Target Value
Density
EXT_SOURCE_3

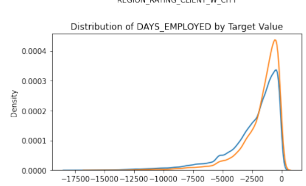

Distribution of FLOORSMAX_AVG by Target Value
Density
FLOORSMAX_AVG

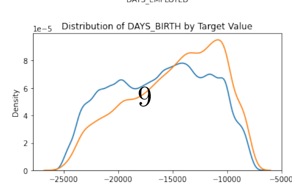

Distribution of FLOORSMAX_MEDI by Target Value
Density
FLOORSMAX_MEDI


Distribution of DAYS_LAST_PHONE_CHANGE by Target Value
Density
DAYS_LAST_PHONE_CHANGE


Distribution of REGION_RATING_CLIENT by Target Value
Density
REGION_RATING_CLIENT


Distribution of REGION_RATING_CLIENT_W_CITY by Target Value
Density
REGION_RATING_CLIENT_W_CITY


Distribution of DAYS_EMPLOYED by Target Value
Density
DAYS_EMPLOYED


Distribution of DAYS_BIRTH by Target Value
1e−5
Density
DAYS_BIRTH

4 Feature Engineering

Statistic feature

[ ]:
```python
def agg_numeric(df, parent_var, df_name):
    for col in df:
        if col != parent_var and 'SK_ID' in col:
            df = df.drop(columns = col)

    # Only want the numeric variables
    parent_ids = df[parent_var].copy()
    numeric_df = df.select_dtypes('number').copy()
    numeric_df[parent_var] = parent_ids

    # Group by the specified variable and calculate the statistics
    agg = numeric_df.groupby(parent_var).agg(['count', 'mean', 'max', 'min',␣
 ↪'sum'])

    # Need to create new column names
    columns = []

    # Iterate through the variables names
    for var in agg.columns.levels[0]:
        if var != parent_var:
            # Iterate through the stat names
            for stat in agg.columns.levels[1]:
                # Make a new column name for the variable and stat
                columns.append('%s_%s_%s' % (df_name, var, stat))
    agg.columns = columns

    # Remove the columns with all redundant values
    _, idx = np.unique(agg, axis = 1, return_index=True)
    agg = agg.iloc[:, idx]

    return agg
```

[ ]:
```python
def agg_categorical(df, parent_var, df_name):
    categorical = pd.get_dummies(df.select_dtypes('object'))

    # Make sure to put the identifying id on the column
    categorical[parent_var] = df[parent_var]

    # Groupby the group var and calculate the sum and mean
    categorical = categorical.groupby(parent_var).agg(['sum', 'count', 'mean'])
```

```
    column_names = []

    # Iterate through the columns in level 0
    for var in categorical.columns.levels[0]:
        # Iterate through the stats in level 1
        for stat in ['sum', 'count', 'mean']:
            # Make a new column name
            column_names.append('%s_%s_%s' % (df_name, var, stat))

    categorical.columns = column_names

    # Remove duplicate columns by values
    _, idx = np.unique(categorical, axis = 1, return_index = True)
    categorical = categorical.iloc[:, idx]

    return categorical
```

[ ]:
```
def aggregate_client(df, group_vars, df_names):
    """Aggregate a dataframe with data at the loan level
    at the client level

    Args:
        df (dataframe): data at the loan level
        group_vars (list of two strings): grouping variables for the loan
        and then the client (example ['SK_ID_PREV', 'SK_ID_CURR'])
        names (list of two strings): names to call the resulting columns
        (example ['cash', 'client'])

    Returns:
        df_client (dataframe): aggregated numeric stats at the client level.
        Each client will have a single row with all the numeric data aggregated
    """

    # Aggregate the numeric columns
    df_agg = agg_numeric(df, parent_var = group_vars[0], df_name = df_names[0])

    # If there are categorical variables
    if any(df.dtypes == 'category'):

        # Count the categorical columns
        df_counts = agg_categorical(df, parent_var = group_vars[0], df_name =␣
 ↪df_names[0])

        # Merge the numeric and categorical
        df_by_loan = df_counts.merge(df_agg, on = group_vars[0], how = 'outer')
```

```python
        gc.enable()

        del df_agg, df_counts
        gc.collect()

        # Merge to get the client id in dataframe
        df_by_loan = df_by_loan.merge(df[[group_vars[0], group_vars[1]]], on =␣
↪group_vars[0], how = 'left')

        # Remove the loan id
        df_by_loan = df_by_loan.drop(columns = [group_vars[0]])

        # Aggregate numeric stats by column
        df_by_client = agg_numeric(df_by_loan, parent_var = group_vars[1],␣
↪df_name = df_names[1])


    # No categorical variables
    else:
        # Merge to get the client id in dataframe
        df_by_loan = df_agg.merge(df[[group_vars[0], group_vars[1]]], on =␣
↪group_vars[0], how = 'left')

        gc.enable()
        del df_agg
        gc.collect()

        # Remove the loan id
        df_by_loan = df_by_loan.drop(columns = [group_vars[0]])

        # Aggregate numeric stats by column
        df_by_client = agg_numeric(df_by_loan, parent_var = group_vars[1],␣
↪df_name = df_names[1])

    # Memory management
    gc.enable()
    del df, df_by_loan
    gc.collect()

    return df_by_client
```

4.1 Main Table

[ ]:
```python
#add percentage
total['DAYS_EMPLOYED_PERC'] = total['DAYS_EMPLOYED'] / total['DAYS_BIRTH']
total['INCOME_CREDIT_PERC'] = total['AMT_INCOME_TOTAL'] / total['AMT_CREDIT']
```

```
total['INCOME_PER_PERSON'] = total['AMT_INCOME_TOTAL'] /␣
↪total['CNT_FAM_MEMBERS']
total['ANNUITY_INCOME_PERC'] = total['AMT_ANNUITY'] / total['AMT_INCOME_TOTAL']
total['PAYMENT_RATE'] = total['AMT_ANNUITY'] / total['AMT_CREDIT']
```

4.2 Bureau balance

[ ]:
```
bureau_balance = pd.read_csv('bureau_balance.csv')
```

[ ]:
```
bureau_balance_cat =agg_categorical(bureau_balance, parent_var =␣
↪'SK_ID_BUREAU', df_name = 'bureau_balance')
```

[ ]:
```
bureau_balance_num = agg_numeric(bureau_balance, parent_var = 'SK_ID_BUREAU',␣
↪df_name = 'bureau_balance')
```

[ ]:
```
bureau_balance_by_bureauid = bureau_balance_num.merge(bureau_balance_cat,␣
↪right_index = True, left_on = 'SK_ID_BUREAU', how = 'outer')
```

4.3 Bureau

[ ]:
```
bureau = pd.read_csv('bureau.csv')
```

[ ]:
```
bureau['debt_to_credit'] = bureau['AMT_CREDIT_SUM_DEBT'] /␣
↪bureau['AMT_CREDIT_SUM']
bureau['unused_credit'] = bureau['AMT_CREDIT_SUM_LIMIT'] /␣
↪bureau['AMT_CREDIT_SUM']
bureau['over_due_ratio'] = bureau['AMT_CREDIT_SUM_OVERDUE'] /␣
↪bureau['AMT_CREDIT_SUM']
bureau['annuity_to_credit'] = bureau['AMT_ANNUITY'] / bureau['AMT_CREDIT_SUM']
```

[ ]:
```
bureau['SK_ID_BUREAU'].nunique()
```

[ ]:
```
1716428
```

[ ]:
```
bureau = bureau.merge(bureau_balance_by_bureauid, on = 'SK_ID_BUREAU', how =␣
↪'left')
```

[ ]:
```
bureau_by_loan_cat =agg_categorical(bureau, parent_var = 'SK_ID_CURR', df_name␣
↪= 'bureau')
```

[ ]:
```
bureau_by_loan_num = agg_numeric(bureau, parent_var = 'SK_ID_CURR', df_name =␣
↪'bureau')
```

[ ]:
```
bureau_by_loan = bureau_by_loan_num.merge(bureau_by_loan_cat, right_index =␣
↪True, left_on = 'SK_ID_CURR', how = 'outer')
```

[ ]:
```
total = total.merge(bureau_by_loan, on = 'SK_ID_CURR', how = 'left')
```

4.4 previous application

[ ]:
```
previous = pd.read_csv('previous_application.csv')
```

[ ]:
```
previous['CREDIT_approved_ratio'] = previous['AMT_CREDIT'] /␣
↪previous['AMT_APPLICATION']
previous['application_to_annuity'] = previous['AMT_APPLICATION'] /␣
↪previous['AMT_ANNUITY']
previous['DOWN_PAYMENT_to_credit'] = previous['AMT_DOWN_PAYMENT'] /␣
↪previous['AMT_CREDIT']
previous['AMT_GOODS_PRICE_to_credit'] = previous['AMT_GOODS_PRICE'] /␣
↪previous['AMT_CREDIT']
```

[ ]:
```
previous_by_loan_cat =agg_categorical(previous, parent_var = 'SK_ID_CURR',␣
↪df_name = 'previous')
```

[ ]:
```
previous_by_loan_num = agg_numeric(previous, parent_var = 'SK_ID_CURR', df_name␣
↪= 'previous')
```

[ ]:
```
previous_by_loan = previous_by_loan_num.merge(previous_by_loan_cat, right_index␣
↪= True, left_on = 'SK_ID_CURR', how = 'outer')
```

[ ]:
```
total = total.merge(previous_by_loan, on ='SK_ID_CURR', how = 'left')
```

4.5 cash

[ ]:
```
cash = pd.read_csv('POS_CASH_balance.csv')
```

[ ]:
```
cash['INSTALMENT_to_balance'] = cash['CNT_INSTALMENT'] / cash['MONTHS_BALANCE']
cash['INSTALMENT_past_future'] = cash['CNT_INSTALMENT'] /␣
↪cash['CNT_INSTALMENT_FUTURE']
```

[ ]:
```
cash_by_loan = aggregate_client(cash, group_vars = ['SK_ID_PREV',␣
↪'SK_ID_CURR'], df_names = ['cash', 'client'])
```

[ ]:
```
total = total.merge(cash_by_loan, on = 'SK_ID_CURR', how = 'left')
```

4.6 credit

[ ]:
```
credit = pd.read_csv('credit_card_balance.csv')
```

[ ]:
```
credit['balance_credit_ratio'] = credit['AMT_BALANCE'] /␣
↪credit['AMT_CREDIT_LIMIT_ACTUAL']
credit['balance_reveivable_ratio'] = credit['AMT_TOTAL_RECEIVABLE']/␣
↪credit['AMT_BALANCE']
credit['payment_to_receivable'] = credit['AMT_PAYMENT_TOTAL_CURRENT'] /␣
↪credit['AMT_TOTAL_RECEIVABLE']
```

```
credit['drawing_to_credit'] = credit['AMT_DRAWINGS_ATM_CURRENT'] /␣
↪credit['AMT_CREDIT_LIMIT_ACTUAL']
```

[ ]:
```
credit_by_loan = aggregate_client(credit, group_vars = ['SK_ID_PREV',␣
↪'SK_ID_CURR'], df_names = ['credit', 'client'])
```

[ ]:
```
total = total.merge(credit_by_loan, on = 'SK_ID_CURR', how = 'left')
```

4.7 installments

[ ]:
```
installments = pd.read_csv('installments_payments.csv')
```

[ ]:
```
installments['AMT_DIFF'] = installments['AMT_INSTALMENT'] -␣
↪installments['AMT_PAYMENT']
installments['AMT_RATIO'] =  (installments['AMT_PAYMENT'] +1)/␣
↪(installments['AMT_INSTALMENT'] + 1)
installments['SK_DPD'] = installments['DAYS_ENTRY_PAYMENT'] -␣
↪installments['DAYS_INSTALMENT']
installments['INS_IS_DPD'] = installments['SK_DPD'].apply(lambda x: 1 if x > 0␣
↪else 0)
installments['INS_IS_DPD_UNDER_120'] = installments['SK_DPD'].apply(lambda x:1␣
↪if (x > 0) & (x <120) else 0 )
installments['INS_IS_DPD_OVER_120'] = installments['SK_DPD'].apply(lambda x:1␣
↪if x >= 120 else 0)
```

[ ]:
```
installments.head()
```

[ ]:

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	\
0	1054186	161674	1.0	6	
1	1330831	151639	0.0	34	
2	2085231	193053	2.0	1	
3	2452527	199697	1.0	3	
4	2714724	167756	1.0	2	

	DAYS_INSTALMENT	DAYS_ENTRY_PAYMENT	AMT_INSTALMENT	AMT_PAYMENT	AMT_DIFF	\
0	-1180.0	-1187.0	6948.360	6948.360	0.000	
1	-2156.0	-2156.0	1716.525	1716.525	0.000	
2	-63.0	-63.0	25425.000	25425.000	0.000	
3	-2418.0	-2426.0	24350.130	24350.130	0.000	
4	-1383.0	-1366.0	2165.040	2160.585	4.455	

	AMT_RATIO	SK_DPD	INS_IS_DPD	INS_IS_DPD_UNDER_120	INS_IS_DPD_OVER_120
0	1.000000	-7.0	0	0	0
1	1.000000	0.0	0	0	0
2	1.000000	0.0	0	0	0
3	1.000000	-8.0	0	0	0
4	0.997943	17.0	1	1	0

```
[ ]: installments_by_loan = aggregate_client(installments, group_vars =␣
     ↪['SK_ID_PREV', 'SK_ID_CURR'], df_names = ['installments', 'client'])
```

```
[ ]: total = total.merge(installments_by_loan, on = 'SK_ID_CURR', how = 'left')
```

```
[ ]: total.shape
```

```
[ ]: (356255, 1607)
```

5 Export

```
[ ]: bureau_by_loan.to_csv('bureau_by_loan.csv', index = False)
     previous_by_loan.to_csv('previous_by_loan.csv', index = False)
     cash_by_loan.to_csv('cash_by_loan.csv',index = False)
     installments_by_loan.to_csv('installments_by_loan.csv',index = False)
```

```
[ ]: total.to_csv('total_f.csv', index = False)
```

6 Modeling

6.1 Modelling Function

```
[ ]: ### define a basic lightgbm
     def lgb_basic(features, test_features, n_folds = 5):
         # Extract the ids
         train_ids = features['SK_ID_CURR']
         test_ids = test_features['SK_ID_CURR']

         # Extract the labels for training
         labels = features['TARGET']

         # Remove the ids and target
         features = features.drop(columns = ['SK_ID_CURR', 'TARGET'])
         test_features = test_features.drop(columns = ['SK_ID_CURR'])

         print('Training Data Shape: ', features.shape)
         print('Testing Data Shape: ', test_features.shape)

         # Extract feature names
         feature_names = list(features.columns)

         # Convert to np arrays
         features = np.array(features)
         test_features = np.array(test_features)

         # Create the kfold object
```

```
    k_fold = KFold(n_splits = n_folds, shuffle = True, random_state = 50)

    # Empty array for feature importances
    feature_importance_values = np.zeros(len(feature_names))

    # Empty array for test predictions
    test_predictions = np.zeros(test_features.shape[0])

    # Empty array for out of fold validation predictions
    out_of_fold = np.zeros(features.shape[0])

    # Lists for recording validation and training scores
    valid_scores = []
    train_scores = []

    # Iterate through each fold
    for train_indices, valid_indices in k_fold.split(features):

        # Training data for the fold
        train_features, train_labels = features[train_indices],␣
↪labels[train_indices]
        # Validation data for the fold
        valid_features, valid_labels = features[valid_indices],␣
↪labels[valid_indices]

        # 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)

        # Record the best iteration
        best_iteration = model.best_iteration_

        # Record the feature importances
```

```
        feature_importance_values += model.feature_importances_ / k_fold.
↪n_splits

        # Make predictions
        test_predictions += model.predict_proba(test_features, num_iteration =␣
↪best_iteration)[:, 1] / k_fold.n_splits

        # Record the out of fold predictions
        out_of_fold[valid_indices] = model.predict_proba(valid_features,␣
↪num_iteration = best_iteration)[:, 1]

        # Record the best score
        valid_score = model.best_score_['valid']['auc']
        train_score = model.best_score_['train']['auc']

        valid_scores.append(valid_score)
        train_scores.append(train_score)

        # Clean up memory
        gc.enable()
        del model, train_features, valid_features
        gc.collect()

    # Make the submission dataframe
    submission = pd.DataFrame({'SK_ID_CURR': test_ids, 'TARGET':␣
↪test_predictions})

    # Make the feature importance dataframe
    feature_importances = pd.DataFrame({'feature': feature_names, 'importance':␣
↪feature_importance_values})

    # Overall validation score
    valid_auc = roc_auc_score(labels, out_of_fold)

    # Add the overall scores to the metrics
    valid_scores.append(valid_auc)
    train_scores.append(np.mean(train_scores))

    # Needed for creating dataframe of validation scores
    fold_names = list(range(n_folds))
    fold_names.append('overall')

    # Dataframe of validation scores
    metrics = pd.DataFrame({'fold': fold_names,
                            'train': train_scores,
                            'valid': valid_scores})
```

```
    return submission, feature_importances, metrics
```

[ ]:
```
### define a basic xgboost
def xgb_basic(features, test_features, n_folds = 3):
    # Extract the ids
    train_ids = features['SK_ID_CURR']
    test_ids = test_features['SK_ID_CURR']

    # Extract the labels for training
    labels = features['TARGET']

    # Remove the ids and target
    features = features.drop(columns = ['SK_ID_CURR', 'TARGET'])
    test_features = test_features.drop(columns = ['SK_ID_CURR'])

    print('Training Data Shape: ', features.shape)
    print('Testing Data Shape: ', test_features.shape)

    # Extract feature names
    feature_names = list(features.columns)

    # Convert to np arrays
    features = np.array(features)
    test_features = np.array(test_features)

    # Create the kfold object
    k_fold = KFold(n_splits = n_folds, shuffle = True, random_state = 50)

    # Empty array for test predictions
    test_predictions = np.zeros(test_features.shape[0])

    # Lists for recording validation and training scores
    valid_scores = []
    train_scores = []

    # Iterate through each fold
    for train_indices, valid_indices in k_fold.split(features):

        # Training data for the fold
        train_features, train_labels = features[train_indices],␣
↪labels[train_indices]
        # Validation data for the fold
        valid_features, valid_labels = features[valid_indices],␣
↪labels[valid_indices]

        # 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)

        # Make predictions
        test_predictions += model.predict_proba(test_features)[:, 1] / k_fold.
↪n_splits

        # Clean up memory
        gc.enable()
        del model, train_features, valid_features
        gc.collect()

    # Make the submission dataframe
    submission = pd.DataFrame({'SK_ID_CURR': test_ids, 'TARGET':␣
↪test_predictions})


    return submission
```

6.2 Basic LightGBM

6.2.1 Main table

[ ]:
```
train_main = pd.read_csv('application_train.csv')
test_main = pd.read_csv('application_test.csv')
train_main = pd.get_dummies(train_main)
test_main = pd.get_dummies(test_main)
train_labels = train['TARGET']
```

```
# Align the training and testing data, keep only columns present in both␣
 ↪dataframes
train_main, test_main = train_main.align(test_main, join = 'inner', axis = 1)

# Add the target back in
train_main['TARGET'] = train_labels
```

[ ]:
```
submission_0, fi_0, metrics_0 = lgb_basic(train_main, test_main)
```

```
Training Data Shape:  (307511, 241)
Testing Data Shape:  (48744, 241)
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.7989     train's binary_logloss: 0.547642        valid's
auc: 0.755463   valid's binary_logloss: 0.563361
[400]   train's auc: 0.82864    train's binary_logloss: 0.518235        valid's
auc: 0.755594   valid's binary_logloss: 0.544951
Early stopping, best iteration is:
[309]   train's auc: 0.815791   train's binary_logloss: 0.531059        valid's
auc: 0.755755   valid's binary_logloss: 0.55289
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.798638   train's binary_logloss: 0.547974        valid's
auc: 0.758354   valid's binary_logloss: 0.56326
Early stopping, best iteration is:
[282]   train's auc: 0.811912   train's binary_logloss: 0.53493 valid's auc:
0.758533   valid's binary_logloss: 0.555531
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.7977     train's binary_logloss: 0.549358        valid's
auc: 0.763287   valid's binary_logloss: 0.564505
Early stopping, best iteration is:
[284]   train's auc: 0.811252   train's binary_logloss: 0.536199        valid's
auc: 0.763822   valid's binary_logloss: 0.556296
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.798947   train's binary_logloss: 0.547854        valid's
auc: 0.757823   valid's binary_logloss: 0.562315
Early stopping, best iteration is:
[240]   train's auc: 0.805899   train's binary_logloss: 0.540963        valid's
auc: 0.758345   valid's binary_logloss: 0.558134
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.798357   train's binary_logloss: 0.548311        valid's
auc: 0.758237   valid's binary_logloss: 0.564466
Early stopping, best iteration is:
[255]   train's auc: 0.807459   train's binary_logloss: 0.539362        valid's
auc: 0.758535   valid's binary_logloss: 0.559106
```

[ ]:
```
submission_0.to_csv('submission_0.csv', index = False)
```

6.2.2 Full Table

[ ]:
```
train = total[total['TARGET'].notnull()]
test = total[total['TARGET'].isnull()]
```

[ ]:
```
del test['TARGET']
print('Final Training Shape: ', train.shape)
print('Final Testing Shape: ', test.shape)
```

```
Final Training Shape:  (307511, 1607)
Final Testing Shape:  (48744, 1606)
```

[ ]:
```
submission_1, fi_1, metrics_1 = lgb_basic(train, test)
```

```
Training Data Shape:  (307511, 1605)
Testing Data Shape:  (48744, 1605)
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.837712   train's binary_logloss: 0.506515        valid's
auc: 0.787749   valid's binary_logloss: 0.527615
[400]   train's auc: 0.874551   train's binary_logloss: 0.463815        valid's
auc: 0.789231   valid's binary_logloss: 0.500202
Early stopping, best iteration is:
[401]   train's auc: 0.874714   train's binary_logloss: 0.463601        valid's
auc: 0.789258   valid's binary_logloss: 0.500069
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.837992   train's binary_logloss: 0.506175        valid's
auc: 0.786721   valid's binary_logloss: 0.526608
Early stopping, best iteration is:
[286]   train's auc: 0.855704   train's binary_logloss: 0.485884        valid's
auc: 0.787351   valid's binary_logloss: 0.514436
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.837538   train's binary_logloss: 0.506806        valid's
auc: 0.78945    valid's binary_logloss: 0.528401
[400]   train's auc: 0.874749   train's binary_logloss: 0.463849        valid's
auc: 0.791125   valid's binary_logloss: 0.500669
Early stopping, best iteration is:
[380]   train's auc: 0.871707   train's binary_logloss: 0.46743 valid's auc:
0.791304   valid's binary_logloss: 0.502976
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.837686   train's binary_logloss: 0.506779        valid's
auc: 0.789684   valid's binary_logloss: 0.52632
Early stopping, best iteration is:
[232]   train's auc: 0.844858   train's binary_logloss: 0.498691        valid's
auc: 0.789871   valid's binary_logloss: 0.521335
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.838362   train's binary_logloss: 0.505954        valid's
auc: 0.784618   valid's binary_logloss: 0.529084
[400]   train's auc: 0.875497   train's binary_logloss: 0.462884        valid's
auc: 0.785648   valid's binary_logloss: 0.502203
```

```
Early stopping, best iteration is:
[334]   train's auc: 0.864647   train's binary_logloss: 0.475738        valid's
auc: 0.785942   valid's binary_logloss: 0.510007
```

[ ]:
```
submission_1.to_csv('submission_1.csv', index = False)
```

6.2.3 Full Table after feature selection

[ ]:
```
#fi_1 = fi_1.sort_values('importance', ascending = False).reset_index()
#fi_1.tail(1000)
cols_to_remove1 = list(fi_1['feature'][fi_1['importance']<6])
train_imp_removed1 = train.drop(columns = cols_to_remove1)
test_imp_removed1 = test.drop(columns = cols_to_remove1)
```

[ ]:
```
fi_1.to_csv('fi_1_basiclgm.csv', index = False)
```

[ ]:
```
train_imp_removed1.shape
```

[ ]:
```
(307511, 427)
```

[ ]:
```
train_imp_removed1.to_csv('train_imp_removed1.csv',index=False)
test_imp_removed1.to_csv('test_imp_removed1.csv',index=False)
```

[ ]:
```
submission_2, fi_2, metrics_2 = lgb_basic(train_imp_removed1, test_imp_removed1)
```

```
Training Data Shape:  (307511, 425)
Testing Data Shape:  (48744, 425)
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.836635   train's binary_logloss: 0.507342        valid's
auc: 0.78828    valid's binary_logloss: 0.52733
[400]   train's auc: 0.873351   train's binary_logloss: 0.464666        valid's
auc: 0.789044   valid's binary_logloss: 0.500729
Early stopping, best iteration is:
[351]   train's auc: 0.865542   train's binary_logloss: 0.473958        valid's
auc: 0.789422   valid's binary_logloss: 0.506527
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.837175   train's binary_logloss: 0.506799        valid's
auc: 0.786306   valid's binary_logloss: 0.527255
[400]   train's auc: 0.873461   train's binary_logloss: 0.464601        valid's
auc: 0.787706   valid's binary_logloss: 0.50058
Early stopping, best iteration is:
[425]   train's auc: 0.877232   train's binary_logloss: 0.459992        valid's
auc: 0.787968   valid's binary_logloss: 0.497583
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.836697   train's binary_logloss: 0.507589        valid's
auc: 0.789514   valid's binary_logloss: 0.528931
[400]   train's auc: 0.873832   train's binary_logloss: 0.464375        valid's
auc: 0.79071    valid's binary_logloss: 0.501277
[600]   train's auc: 0.900753   train's binary_logloss: 0.43033 valid's auc:
```

```
0.790431   valid's binary_logloss: 0.479436
Early stopping, best iteration is:
[520]   train's auc: 0.890906   train's binary_logloss: 0.443205        valid's
auc: 0.790911   valid's binary_logloss: 0.48765
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.836562   train's binary_logloss: 0.507898        valid's
auc: 0.790517   valid's binary_logloss: 0.526938
Early stopping, best iteration is:
[274]   train's auc: 0.851904   train's binary_logloss: 0.490237        valid's
auc: 0.791105   valid's binary_logloss: 0.516091
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.83769    train's binary_logloss: 0.50676 valid's auc:
0.784483   valid's binary_logloss: 0.529775
[400]   train's auc: 0.873881   train's binary_logloss: 0.46398 valid's auc:
0.786605   valid's binary_logloss: 0.503078
Early stopping, best iteration is:
[424]   train's auc: 0.877406   train's binary_logloss: 0.459734        valid's
auc: 0.78685    valid's binary_logloss: 0.500329
```

[ ]:
```
submission_2.to_csv('submission_2.csv', index = False)
```

6.2.4 Full Table after feature selection and correlation remove

[ ]:
```
threshold = 0.8
corrs = train_imp_removed1.corr()

# Empty dictionary to hold correlated variables
above_threshold_vars = {}

# For each column, record the variables that are above the threshold
for col in corrs:
    above_threshold_vars[col] = list(corrs.index[corrs[col] > threshold])
```

[ ]:
```
cols_to_remove = []
cols_seen = []
cols_to_remove_pair = []

# Iterate through columns and correlated columns
for key, value in above_threshold_vars.items():
    # Keep track of columns already examined
    cols_seen.append(key)
    for x in value:
        if x == key:
            next
        else:
            # Only want to remove one in a pair
            if x not in cols_seen:
                cols_to_remove.append(x)
```

```
                cols_to_remove_pair.append(key)

cols_to_remove2 = list(set(cols_to_remove))
print('Number of columns to remove: ', len(cols_to_remove2))
```

```
Number of columns to remove:  171
```

[ ]:
```
train_corr_removed1 = train_imp_removed1.drop(columns = cols_to_remove2)
test_corr_removed1 = test_imp_removed1.drop(columns = cols_to_remove2)
```

[ ]:
```
submission_3, fi_3, metrics_3 = lgb_basic(train_corr_removed1,␣
↪test_corr_removed1)
```

```
Training Data Shape:  (307511, 254)
Testing Data Shape:  (48744, 254)
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.832601   train's binary_logloss: 0.511857        valid's
auc: 0.787806   valid's binary_logloss: 0.530662
[400]   train's auc: 0.868113   train's binary_logloss: 0.471259        valid's
auc: 0.789374   valid's binary_logloss: 0.504874
Early stopping, best iteration is:
[352]   train's auc: 0.860716   train's binary_logloss: 0.479799        valid's
auc: 0.789643   valid's binary_logloss: 0.510267
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.83357    train's binary_logloss: 0.51074 valid's auc:
0.784456   valid's binary_logloss: 0.530013
[400]   train's auc: 0.868886   train's binary_logloss: 0.470253        valid's
auc: 0.78495    valid's binary_logloss: 0.505018
Early stopping, best iteration is:
[310]   train's auc: 0.854605   train's binary_logloss: 0.48666 valid's auc:
0.785217   valid's binary_logloss: 0.515182
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.832809   train's binary_logloss: 0.511809        valid's
auc: 0.788671   valid's binary_logloss: 0.532114
[400]   train's auc: 0.868043   train's binary_logloss: 0.47152 valid's auc:
0.790203   valid's binary_logloss: 0.505913
Early stopping, best iteration is:
[448]   train's auc: 0.875168   train's binary_logloss: 0.462989        valid's
auc: 0.790414   valid's binary_logloss: 0.500244
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.832823   train's binary_logloss: 0.511786        valid's
auc: 0.787835   valid's binary_logloss: 0.530687
Early stopping, best iteration is:
[268]   train's auc: 0.846464   train's binary_logloss: 0.496275        valid's
auc: 0.788757   valid's binary_logloss: 0.521151
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.833566   train's binary_logloss: 0.510965        valid's
auc: 0.783934   valid's binary_logloss: 0.532668
```

```
[400]	train's auc: 0.868949	train's binary_logloss: 0.470401	valid's
auc: 0.785282	valid's binary_logloss: 0.507192
Early stopping, best iteration is:
[363]	train's auc: 0.86331	train's binary_logloss: 0.477062	valid's
auc: 0.785504	valid's binary_logloss: 0.511281
```

[ ]:
```
submission_3.to_csv('submission_3.csv', index = False)
```

[ ]:
```
train_corr_removed1.to_csv('train_corr_removed1.csv', index = False)
test_corr_removed1.to_csv('test_corr_removed1.csv', index = False)
```

6.3 Basic XGBoost

6.3.1 Main Table

[ ]:
```
total_main=total.copy()
```

[ ]:
```
train_main = total_main[total_main['TARGET'].notnull()]
test_main = total_main[total_main['TARGET'].isnull()]
```

[ ]:
```
del test_main['TARGET']
```

[ ]:
```
submission_2 = xgb_basic(train_main, test_main)
```

[ ]:
```
submission_2.to_csv('submission_2.csv', index = False)
# without imputation: 0.75083
# with imputation:  0.75014
```

6.3.2 Baseline(main with imputation)

[ ]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='mean')
imputer.fit(train_main)
train_main_transfer = imputer.transform(train_main)
imputer.fit(test_main)
test_main_transfer = imputer.transform(test_main)
```

[ ]:
```
train_main_transfer = pd.DataFrame(train_main_transfer, columns = train_main.
 ↪columns)
test_main_transfer = pd.DataFrame(test_main_transfer, columns = test_main.
 ↪columns)
```

[ ]:
```
train_main_transfer['SK_ID_CURR']= train_main_transfer['SK_ID_CURR'].
 ↪astype('int32')
test_main_transfer['SK_ID_CURR']=test_main_transfer['SK_ID_CURR'].
 ↪astype('int32')
```

[ ]:
```
missing_values_table(train_main_transfer)
```

[ ]:
```
missing_values_table(test_main_transfer)
```

6.3.3 Full Table

[ ]:
```
train.replace([np.inf, -np.inf], np.nan, inplace=True)
test.replace([np.inf, -np.inf], np.nan, inplace=True)
```

[ ]:
```
submission_4 = xgb_basic(train, test)
```

```
Training Data Shape:  (307511, 1605)
Testing Data Shape:  (48744, 1605)
[21:33:09] WARNING: ../src/learner.cc:576:
Parameters: { "nrounds" } might not be used.

  This could be a false alarm, with some parameters getting used by language
bindings but
  then being mistakenly passed down to XGBoost core, or some parameter actually
being used
  but getting flagged wrongly here. Please open an issue if you find any such
cases.


[0]     validation_0-auc:0.69412        validation_1-auc:0.71054
[200]   validation_0-auc:0.78360        validation_1-auc:0.83638
[400]   validation_0-auc:0.78769        validation_1-auc:0.86328
[600]   validation_0-auc:0.78873        validation_1-auc:0.88372
[800]   validation_0-auc:0.78879        validation_1-auc:0.90011
[999]   validation_0-auc:0.78809        validation_1-auc:0.91394
[22:41:30] WARNING: ../src/learner.cc:576:
Parameters: { "nrounds" } might not be used.

  This could be a false alarm, with some parameters getting used by language
bindings but
  then being mistakenly passed down to XGBoost core, or some parameter actually
being used
  but getting flagged wrongly here. Please open an issue if you find any such
cases.


[0]     validation_0-auc:0.69375        validation_1-auc:0.70765
[200]   validation_0-auc:0.78767        validation_1-auc:0.83633
[400]   validation_0-auc:0.79190        validation_1-auc:0.86413
[600]   validation_0-auc:0.79232        validation_1-auc:0.88409
[800]   validation_0-auc:0.79224        validation_1-auc:0.90027
[999]   validation_0-auc:0.79156        validation_1-auc:0.91446
[23:56:48] WARNING: ../src/learner.cc:576:
Parameters: { "nrounds" } might not be used.
```

```
  This could be a false alarm, with some parameters getting used by language
bindings but
  then being mistakenly passed down to XGBoost core, or some parameter actually
being used
  but getting flagged wrongly here. Please open an issue if you find any such
cases.


[0]     validation_0-auc:0.69386        validation_1-auc:0.70714
[200]   validation_0-auc:0.78442        validation_1-auc:0.83679
[400]   validation_0-auc:0.78881        validation_1-auc:0.86401
[600]   validation_0-auc:0.78942        validation_1-auc:0.88358
[800]   validation_0-auc:0.78895        validation_1-auc:0.90065
[999]   validation_0-auc:0.78802        validation_1-auc:0.91469
```

[ ]:
```
submission_4.to_csv('submission_4.csv', index = False)
```

6.3.4 Full Table after feature selection

[ ]:
```
train_imp=pd.read_csv('/content/train_imp_removed1.csv')
test_imp=pd.read_csv('/content/test_imp_removed1.csv')
```

[ ]:
```
submission_3 = xgb_basic(train_imp, test_imp)
```

```
Training Data Shape:  (307511, 425)
Testing Data Shape:  (48744, 425)
[0]     validation_0-auc:0.683017       validation_1-auc:0.697894
Multiple eval metrics have been passed: 'validation_1-auc' will be used for
early stopping.

Will train until validation_1-auc hasn't improved in 100 rounds.
[200]   validation_0-auc:0.787262       validation_1-auc:0.862596
[400]   validation_0-auc:0.786496       validation_1-auc:0.900466
[600]   validation_0-auc:0.783857       validation_1-auc:0.928599
[800]   validation_0-auc:0.781523       validation_1-auc:0.948851
[999]   validation_0-auc:0.778698       validation_1-auc:0.96347
[0]     validation_0-auc:0.687408       validation_1-auc:0.698289
Multiple eval metrics have been passed: 'validation_1-auc' will be used for
early stopping.

Will train until validation_1-auc hasn't improved in 100 rounds.
[200]   validation_0-auc:0.788898       validation_1-auc:0.862812
[400]   validation_0-auc:0.787796       validation_1-auc:0.899894
[600]   validation_0-auc:0.786556       validation_1-auc:0.92675
```

[ ]:
```
submission_3.to_csv('submission_3.csv', index = False)
```

6.3.5 Full Table after feature selection and correlation remove

[ ]:
```
threshold = 0.8
corrs = train.corr()
# Empty dictionary to hold correlated variables
above_threshold_vars = {}

# For each column, record the variables that are above the threshold
for col in corrs:
    above_threshold_vars[col] = list(corrs.index[corrs[col] > threshold])

cols_to_remove1 = []
cols_seen = []
cols_to_remove_pair = []

# Iterate through columns and correlated columns
for key, value in above_threshold_vars.items():
    # Keep track of columns already examined
    cols_seen.append(key)
    for x in value:
        if x == key:
            next
        else:
            # Only want to remove one in a pair
            if x not in cols_seen:
                cols_to_remove.append(x)
                cols_to_remove_pair.append(key)

cols_to_remove1 = list(set(cols_to_remove))
print('Number of columns to remove: ', len(cols_to_remove))
```

[ ]:
```
train_corrs_removed = train.drop(columns = cols_to_remove)
test_corrs_removed = test.drop(columns = cols_to_remove)
```

[ ]:
```
submission_4 = xgb_basic(train_corrs_removed, test_corrs_removed)
```

[ ]:
```
submission_4.to_csv('submission_4.csv', index = False)
#
```

7 BayesianOptimization

[ ]:
```
### define a BayesianOptimization function for lightgbm
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)
```

[ ]:

```
### define a BayesianOptimization function for XGBoost
#Converting the dataframe into XGBoost's Dmatrix object
dtrain = xgb.DMatrix(train, label=train_imp['TARGET'])

#Bayesian Optimization function for xgboost
#specify the parameters you want to tune as keyword arguments
def bo_tune_xgb(max_depth, gamma, n_estimators ,learning_rate):
     params = {'max_depth': int(max_depth),
               'gamma': gamma,
               'n_estimators': int(n_estimators),
               'learning_rate':learning_rate,
               'subsample': 0.8,
               'eta': 0.1,
               'eval_metric': 'auc'}
    #Cross validating with the specified parameters in 5 folds and 70 iterations
     cv_result = xgb.cv(params, dtrain, num_boost_round=70, nfold=5)
    #Return the auc
     return max(cv_result['test-auc-mean'])

#Invoking the Bayesian Optimizer with the specified parameters to tune
xgb_bo = BayesianOptimization(bo_tune_xgb, {'max_depth': (3, 10),
                                             'gamma': (0, 1),
                                             'learning_rate':(0,1),
```

```
                                                'n_estimators':(1000,1200)
                                               })

#performing Bayesian optimization for 5 iterations with 8 steps of random␣
↪exploration with an #acquisition function of expected improvement
xgb_bo.maximize(n_iter=5, init_points=8, acq='ei')
```

```
Requirement already satisfied: bayesian-optimization in
/usr/local/lib/python3.7/dist-packages (1.2.0)
Requirement already satisfied: numpy>=1.9.0 in /usr/local/lib/python3.7/dist-
packages (from bayesian-optimization) (1.21.5)
Requirement already satisfied: scipy>=0.14.0 in /usr/local/lib/python3.7/dist-
packages (from bayesian-optimization) (1.4.1)
Requirement already satisfied: scikit-learn>=0.18.0 in
/usr/local/lib/python3.7/dist-packages (from bayesian-optimization) (1.0.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.18.0->bayesian-
optimization) (3.1.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-
packages (from scikit-learn>=0.18.0->bayesian-optimization) (1.1.0)
|   iter    |  target   |   gamma   | learni… | max_depth | n_esti… |
-------------------------------------------------------------------------
|  1        |  0.7517   |  0.08604  |  0.6105
|  4.889    |  108.5    |
|  2        |  0.7636   |  0.09757  |  0.09752

 |  3.461    |  116.3    |
|  3        |  0.7673   |  0.9101   |  0.1474

 |  5.709    |  112.4    |
|  4        |  0.7423   |  0.359    |  0.6182
|  5.9      |  111.4    |
|  5        |  0.7441   |  0.5929   |  0.8481
|  3.394    |  103.4    |
|  6        |  0.7514   |  0.4904   |  0.439
|  5.021    |  110.3    |
|  7        |  0.7267   |  0.9486   |  0.00948
|  4.781    |  113.7    |
|  8        |  0.76     |  0.9147   |  0.1052
|  8.932    |  108.5    |
|  9        |  0.7293   |  0.9211   |  0.7306
|  7.652    |  101.6    |
|  10       |  0.7632   |  0.8523   |  0.2098
|  5.772    |  112.3    |
|  11       |  0.5      |  1.0      |  0.0
|  5.552    |  112.4    |
|  12       |  0.7499   |  0.877    |  0.4863
|  5.746    |  101.5    |
```

```
|  13        |  0.7427    |  0.5085    |  0.4864
|  6.091     |  113.2     |
==========================================================================
```

8 Tuned LightGBM

```
[ ]: train = total[total['TARGET'].notnull()]
     test = total[total['TARGET'].isnull()]
```

```
[ ]: del test['TARGET']
     print('Final Training Shape: ', train.shape)
     print('Final Testing Shape: ', test.shape)
```

```
Final Training Shape:  (307511, 1685)
Final Testing Shape:  (48744, 1684)
```

8.1 Full table

```
[ ]: def lightgbm_tuned(features, test_features, encoding = 'ohe', n_folds = 5):
         # Extract the ids
         train_ids = features['SK_ID_CURR']
         test_ids = test_features['SK_ID_CURR']

         # Extract the labels for training
         labels = features['TARGET']

         # Remove the ids and target
         features = features.drop(columns = ['SK_ID_CURR', 'TARGET'])
         test_features = test_features.drop(columns = ['SK_ID_CURR'])

         print('Training Data Shape: ', features.shape)
         print('Testing Data Shape: ', test_features.shape)

         # Extract feature names
         feature_names = list(features.columns)

         # Convert to np arrays
         features = np.array(features)
         test_features = np.array(test_features)

         # Create the kfold object
         k_fold = KFold(n_splits = n_folds, shuffle = True, random_state = 50)

         # Empty array for feature importances
         feature_importance_values = np.zeros(len(feature_names))

         # Empty array for test predictions
```

```
    test_predictions = np.zeros(test_features.shape[0])

    # Empty array for out of fold validation predictions
    out_of_fold = np.zeros(features.shape[0])

    # Lists for recording validation and training scores
    valid_scores = []
    train_scores = []

    # Iterate through each fold
    for train_indices, valid_indices in k_fold.split(features):

        # Training data for the fold
        train_features, train_labels = features[train_indices],␣
↪labels[train_indices]
        # Validation data for the fold
        valid_features, valid_labels = features[valid_indices],␣
↪labels[valid_indices]

        # 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)

        # Record the best iteration
        best_iteration = model.best_iteration_

        # Record the feature importances
        feature_importance_values += model.feature_importances_ / k_fold.
↪n_splits
```

```
        # Make predictions
        test_predictions += model.predict_proba(test_features, num_iteration =␣
↪best_iteration)[:, 1] / k_fold.n_splits

        # Record the out of fold predictions
        out_of_fold[valid_indices] = model.predict_proba(valid_features,␣
↪num_iteration = best_iteration)[:, 1]

        # Record the best score
        valid_score = model.best_score_['valid']['auc']
        train_score = model.best_score_['train']['auc']

        valid_scores.append(valid_score)
        train_scores.append(train_score)

        # Clean up memory
        gc.enable()
        del model, train_features, valid_features
        gc.collect()

    # Make the submission dataframe
    submission = pd.DataFrame({'SK_ID_CURR': test_ids, 'TARGET':␣
↪test_predictions})

    # Make the feature importance dataframe
    feature_importances = pd.DataFrame({'feature': feature_names, 'importance':␣
↪feature_importance_values})

    # Overall validation score
    valid_auc = roc_auc_score(labels, out_of_fold)

    # Add the overall scores to the metrics
    valid_scores.append(valid_auc)
    train_scores.append(np.mean(train_scores))

    # Needed for creating dataframe of validation scores
    fold_names = list(range(n_folds))
    fold_names.append('overall')

    # Dataframe of validation scores
    metrics = pd.DataFrame({'fold': fold_names,
                            'train': train_scores,
                            'valid': valid_scores})

    return submission, feature_importances, metrics
```

[ ]:
```
submission_f, fi_f, metrics_f = lightgbm_tuned(train, test)
```

```
Training Data Shape:  (307511, 1605)
Testing Data Shape:  (48744, 1605)
[LightGBM] [Warning] feature_fraction is set=0.1926, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.1926
[LightGBM] [Warning] bagging_fraction is set=0.8898, subsample=0.8 will be
ignored. Current value: bagging_fraction=0.8898
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.828608   train's binary_logloss: 0.519552        valid's
auc: 0.788115   valid's binary_logloss: 0.534029
[400]   train's auc: 0.860668   train's binary_logloss: 0.480837        valid's
auc: 0.791601   valid's binary_logloss: 0.508776
Early stopping, best iteration is:
[456]   train's auc: 0.867999   train's binary_logloss: 0.472189        valid's
auc: 0.791875   valid's binary_logloss: 0.503341
[LightGBM] [Warning] feature_fraction is set=0.1926, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.1926
[LightGBM] [Warning] bagging_fraction is set=0.8898, subsample=0.8 will be
ignored. Current value: bagging_fraction=0.8898
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.82843    train's binary_logloss: 0.519003        valid's
auc: 0.785621   valid's binary_logloss: 0.53421
[400]   train's auc: 0.860895   train's binary_logloss: 0.4803  valid's auc:
0.789403   valid's binary_logloss: 0.509083
[600]   train's auc: 0.884785   train's binary_logloss: 0.451449        valid's
auc: 0.789894   valid's binary_logloss: 0.49078
Early stopping, best iteration is:
[521]   train's auc: 0.876128   train's binary_logloss: 0.462088        valid's
auc: 0.790096   valid's binary_logloss: 0.49748
[LightGBM] [Warning] feature_fraction is set=0.1926, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.1926
[LightGBM] [Warning] bagging_fraction is set=0.8898, subsample=0.8 will be
ignored. Current value: bagging_fraction=0.8898
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.828293   train's binary_logloss: 0.51999 valid's auc:
0.78888    valid's binary_logloss: 0.535769
[400]   train's auc: 0.860756   train's binary_logloss: 0.4809  valid's auc:
0.793654   valid's binary_logloss: 0.509391
[600]   train's auc: 0.885174   train's binary_logloss: 0.451479        valid's
auc: 0.794953   valid's binary_logloss: 0.489981
Early stopping, best iteration is:
[660]   train's auc: 0.89128    train's binary_logloss: 0.443874        valid's
auc: 0.795304   valid's binary_logloss: 0.484939
[LightGBM] [Warning] feature_fraction is set=0.1926, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.1926
[LightGBM] [Warning] bagging_fraction is set=0.8898, subsample=0.8 will be
ignored. Current value: bagging_fraction=0.8898
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.828061   train's binary_logloss: 0.520327        valid's
```

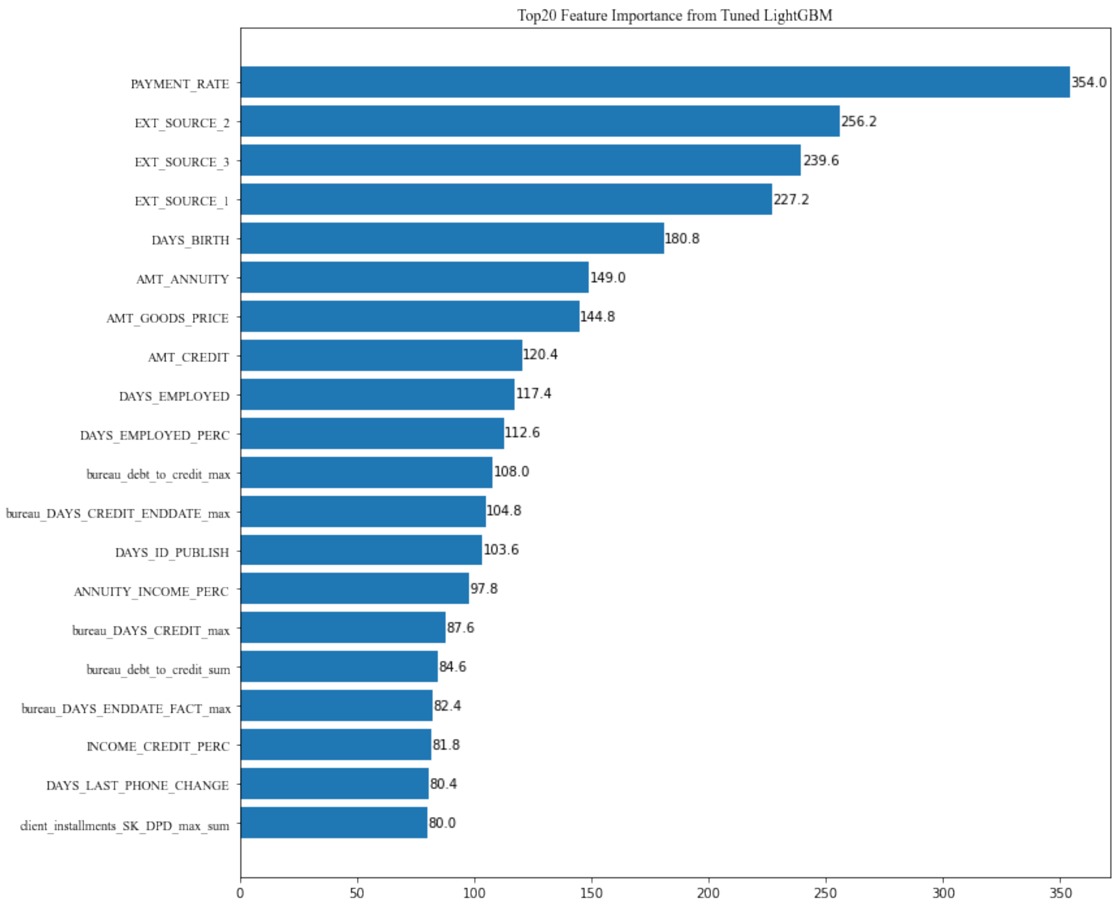
```
auc: 0.78965    valid's binary_logloss: 0.534057
[400]   train's auc: 0.860449   train's binary_logloss: 0.481368        valid's
auc: 0.792956   valid's binary_logloss: 0.509041
[600]   train's auc: 0.884881   train's binary_logloss: 0.451953        valid's
auc: 0.793199   valid's binary_logloss: 0.490351
Early stopping, best iteration is:
[690]   train's auc: 0.893737   train's binary_logloss: 0.440661        valid's
auc: 0.793418   valid's binary_logloss: 0.483099
[LightGBM] [Warning] feature_fraction is set=0.1926, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.1926
[LightGBM] [Warning] bagging_fraction is set=0.8898, subsample=0.8 will be
ignored. Current value: bagging_fraction=0.8898
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.829253   train's binary_logloss: 0.51883 valid's auc:
0.784444   valid's binary_logloss: 0.536922
[400]   train's auc: 0.86138    train's binary_logloss: 0.480032        valid's
auc: 0.788726   valid's binary_logloss: 0.511919
[600]   train's auc: 0.885526   train's binary_logloss: 0.450919        valid's
auc: 0.789516   valid's binary_logloss: 0.493375
Early stopping, best iteration is:
[634]   train's auc: 0.88906    train's binary_logloss: 0.446462        valid's
auc: 0.789591   valid's binary_logloss: 0.490585
```

[ ]:
```python
submission_f.to_csv('submission_f.csv', index = False)
```

[ ]:
```python
fi_f.to_csv('fi_f.csv', index = False)
```

[ ]:
```python
fi_f = fi_f.sort_values('importance', ascending = False).reset_index()
```

[ ]:
```python
fi_f_head= fi_f.head(20)
from matplotlib.pyplot import figure

fig, ax = plt.subplots(figsize =(12, 12))
fi_f_head.sort_values('importance',inplace=True)
plt.barh(fi_f_head['feature'],fi_f_head['importance'])
plt.title('Top20 Feature Importance from Tuned LightGBM',fontname = "Times New␣
↪Roman")
plt.yticks(fontname = "Times New Roman")
for i in ax.patches:
    plt.text(i.get_width()+0.2, i.get_y()+0.3,
             str(round((i.get_width()), 2)),
             fontsize = 10)
plt.show()
```



8.2 Main table

[ ]:
```
submission_f_0, fi_f_0, metrics_f_0 = lightgbm_tuned(train_main, test_main)
```

```
Training Data Shape:  (307511, 241)
Testing Data Shape:  (48744, 241)
[LightGBM] [Warning] feature_fraction is set=0.1926, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.1926
[LightGBM] [Warning] bagging_fraction is set=0.8898, subsample=0.8 will be
ignored. Current value: bagging_fraction=0.8898
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.787588   train's binary_logloss: 0.561587        valid's
auc: 0.752542   valid's binary_logloss: 0.572127
[400]   train's auc: 0.81002    train's binary_logloss: 0.537701        valid's
auc: 0.755419   valid's binary_logloss: 0.556298
[600]   train's auc: 0.827867   train's binary_logloss: 0.519643        valid's
auc: 0.755376   valid's binary_logloss: 0.545117
Early stopping, best iteration is:
[549]   train's auc: 0.823629   train's binary_logloss: 0.523924        valid's
```

```
auc: 0.755785   valid's binary_logloss: 0.547742
[LightGBM] [Warning] feature_fraction is set=0.1926, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.1926
[LightGBM] [Warning] bagging_fraction is set=0.8898, subsample=0.8 will be
ignored. Current value: bagging_fraction=0.8898
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.786989   train's binary_logloss: 0.562427        valid's
auc: 0.756629   valid's binary_logloss: 0.572099
[400]   train's auc: 0.809942   train's binary_logloss: 0.538134        valid's
auc: 0.758747   valid's binary_logloss: 0.556798
Early stopping, best iteration is:
[425]   train's auc: 0.812399   train's binary_logloss: 0.535665        valid's
auc: 0.758832   valid's binary_logloss: 0.555311
[LightGBM] [Warning] feature_fraction is set=0.1926, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.1926
[LightGBM] [Warning] bagging_fraction is set=0.8898, subsample=0.8 will be
ignored. Current value: bagging_fraction=0.8898
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.786418   train's binary_logloss: 0.562949        valid's
auc: 0.760513   valid's binary_logloss: 0.572927
[400]   train's auc: 0.80935    train's binary_logloss: 0.538848        valid's
auc: 0.76275    valid's binary_logloss: 0.556755
Early stopping, best iteration is:
[364]   train's auc: 0.805615   train's binary_logloss: 0.542514        valid's
auc: 0.762877   valid's binary_logloss: 0.558993
[LightGBM] [Warning] feature_fraction is set=0.1926, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.1926
[LightGBM] [Warning] bagging_fraction is set=0.8898, subsample=0.8 will be
ignored. Current value: bagging_fraction=0.8898
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.787174   train's binary_logloss: 0.562297        valid's
auc: 0.757339   valid's binary_logloss: 0.571029
[400]   train's auc: 0.810125   train's binary_logloss: 0.538107        valid's
auc: 0.759487   valid's binary_logloss: 0.555363
[600]   train's auc: 0.82836    train's binary_logloss: 0.5197  valid's auc:
0.759651   valid's binary_logloss: 0.544103
Early stopping, best iteration is:
[642]   train's auc: 0.831818   train's binary_logloss: 0.516061        valid's
auc: 0.759847   valid's binary_logloss: 0.541771
[LightGBM] [Warning] feature_fraction is set=0.1926, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.1926
[LightGBM] [Warning] bagging_fraction is set=0.8898, subsample=0.8 will be
ignored. Current value: bagging_fraction=0.8898
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.786902   train's binary_logloss: 0.562444        valid's
auc: 0.757282   valid's binary_logloss: 0.5726
[400]   train's auc: 0.809435   train's binary_logloss: 0.538345        valid's
auc: 0.759381   valid's binary_logloss: 0.557242
```

```
[600]	train's auc: 0.827349	train's binary_logloss: 0.520385	valid's
auc: 0.759415	valid's binary_logloss: 0.54636
Early stopping, best iteration is:
[515]	train's auc: 0.820254	train's binary_logloss: 0.52756 valid's auc:
0.759605	valid's binary_logloss: 0.550622
```

[ ]:
```
submission_f_0.to_csv('submission_f_0.csv', index = False)
```

8.3 Full Table after feature selection

[ ]:
```
submission_f_1, fi_f_1, metrics_f_1 = lightgbm_tuned(train_imp_removed1,␣
↪test_imp_removed1)
```

```
Training Data Shape:  (307511, 425)
Testing Data Shape:  (48744, 425)
[LightGBM] [Warning] feature_fraction is set=0.1926, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.1926
[LightGBM] [Warning] bagging_fraction is set=0.8898, subsample=0.8 will be
ignored. Current value: bagging_fraction=0.8898
Training until validation scores don't improve for 100 rounds
[200]	train's auc: 0.827262	train's binary_logloss: 0.520998	valid's
auc: 0.788258	valid's binary_logloss: 0.535091
[400]	train's auc: 0.858214	train's binary_logloss: 0.483241	valid's
auc: 0.792727	valid's binary_logloss: 0.510114
Early stopping, best iteration is:
[476]	train's auc: 0.868003	train's binary_logloss: 0.471672	valid's
auc: 0.793057	valid's binary_logloss: 0.502901
[LightGBM] [Warning] feature_fraction is set=0.1926, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.1926
[LightGBM] [Warning] bagging_fraction is set=0.8898, subsample=0.8 will be
ignored. Current value: bagging_fraction=0.8898
Training until validation scores don't improve for 100 rounds
[200]	train's auc: 0.827863	train's binary_logloss: 0.519967	valid's
auc: 0.785563	valid's binary_logloss: 0.535138
[400]	train's auc: 0.85884	train's binary_logloss: 0.482252	valid's
auc: 0.789494	valid's binary_logloss: 0.510813
Early stopping, best iteration is:
[469]	train's auc: 0.867432	train's binary_logloss: 0.472033	valid's
auc: 0.790117	valid's binary_logloss: 0.504326
[LightGBM] [Warning] feature_fraction is set=0.1926, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.1926
[LightGBM] [Warning] bagging_fraction is set=0.8898, subsample=0.8 will be
ignored. Current value: bagging_fraction=0.8898
Training until validation scores don't improve for 100 rounds
[200]	train's auc: 0.827154	train's binary_logloss: 0.521041	valid's
auc: 0.788089	valid's binary_logloss: 0.537032
[400]	train's auc: 0.858669	train's binary_logloss: 0.482873	valid's
auc: 0.7939	valid's binary_logloss: 0.510896
```

```
[600]	train's auc: 0.882672	train's binary_logloss: 0.454066	valid's
auc: 0.794509	valid's binary_logloss: 0.492449
Early stopping, best iteration is:
[507]	train's auc: 0.872072	train's binary_logloss: 0.466991	valid's
auc: 0.794618	valid's binary_logloss: 0.500709
[LightGBM] [Warning] feature_fraction is set=0.1926, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.1926
[LightGBM] [Warning] bagging_fraction is set=0.8898, subsample=0.8 will be
ignored. Current value: bagging_fraction=0.8898
Training until validation scores don't improve for 100 rounds
[200]	train's auc: 0.826939	train's binary_logloss: 0.521393	valid's
auc: 0.791002	valid's binary_logloss: 0.534447
[400]	train's auc: 0.858717	train's binary_logloss: 0.482772	valid's
auc: 0.795343	valid's binary_logloss: 0.509584
Early stopping, best iteration is:
[472]	train's auc: 0.867655	train's binary_logloss: 0.472032	valid's
auc: 0.795941	valid's binary_logloss: 0.502741
[LightGBM] [Warning] feature_fraction is set=0.1926, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.1926
[LightGBM] [Warning] bagging_fraction is set=0.8898, subsample=0.8 will be
ignored. Current value: bagging_fraction=0.8898
Training until validation scores don't improve for 100 rounds
[200]	train's auc: 0.828403	train's binary_logloss: 0.519883	valid's
auc: 0.78524	valid's binary_logloss: 0.537536
[400]	train's auc: 0.860046	train's binary_logloss: 0.481263	valid's
auc: 0.789558	valid's binary_logloss: 0.512588
[600]	train's auc: 0.883472	train's binary_logloss: 0.452784	valid's
auc: 0.790321	valid's binary_logloss: 0.494531
Early stopping, best iteration is:
[654]	train's auc: 0.889054	train's binary_logloss: 0.445723	valid's
auc: 0.790709	valid's binary_logloss: 0.489836
```

[ ]:
```python
submission_f_1.to_csv('submission_f_1.csv', index = False)
```

8.4 Full Table after feature selection and correlation remove

[ ]:
```python
submission_f_2, fi_f_2, metrics_f_2 = lightgbm_tuned(train_corr_removed1,␣
↪test_corr_removed1)
```

```
Training Data Shape:  (307511, 254)
Testing Data Shape:  (48744, 254)
[LightGBM] [Warning] feature_fraction is set=0.1926, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.1926
[LightGBM] [Warning] bagging_fraction is set=0.8898, subsample=0.8 will be
ignored. Current value: bagging_fraction=0.8898
Training until validation scores don't improve for 100 rounds
[200]	train's auc: 0.821835	train's binary_logloss: 0.52658 valid's auc:
0.785992	valid's binary_logloss: 0.539738
```

```
[400]   train's auc: 0.851666   train's binary_logloss: 0.491255        valid's
auc: 0.790204   valid's binary_logloss: 0.516224
[600]   train's auc: 0.87433    train's binary_logloss: 0.464493        valid's
auc: 0.790988   valid's binary_logloss: 0.499303
Early stopping, best iteration is:
[519]   train's auc: 0.865662   train's binary_logloss: 0.474747        valid's
auc: 0.791271   valid's binary_logloss: 0.505678
[LightGBM] [Warning] feature_fraction is set=0.1926, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.1926
[LightGBM] [Warning] bagging_fraction is set=0.8898, subsample=0.8 will be
ignored. Current value: bagging_fraction=0.8898
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.822889   train's binary_logloss: 0.525309        valid's
auc: 0.781689   valid's binary_logloss: 0.539326
[400]   train's auc: 0.853002   train's binary_logloss: 0.489552        valid's
auc: 0.786295   valid's binary_logloss: 0.516224
[600]   train's auc: 0.875002   train's binary_logloss: 0.463294        valid's
auc: 0.786924   valid's binary_logloss: 0.499953
Early stopping, best iteration is:
[580]   train's auc: 0.872964   train's binary_logloss: 0.46571 valid's auc:
0.787061   valid's binary_logloss: 0.501401
[LightGBM] [Warning] feature_fraction is set=0.1926, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.1926
[LightGBM] [Warning] bagging_fraction is set=0.8898, subsample=0.8 will be
ignored. Current value: bagging_fraction=0.8898
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.822198   train's binary_logloss: 0.52616 valid's auc:
0.78606    valid's binary_logloss: 0.541351
[400]   train's auc: 0.852392   train's binary_logloss: 0.49053 valid's auc:
0.790932   valid's binary_logloss: 0.517225
[600]   train's auc: 0.875115   train's binary_logloss: 0.46375 valid's auc:
0.791599   valid's binary_logloss: 0.499953
Early stopping, best iteration is:
[603]   train's auc: 0.875378   train's binary_logloss: 0.463426        valid's
auc: 0.791621   valid's binary_logloss: 0.499729
[LightGBM] [Warning] feature_fraction is set=0.1926, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.1926
[LightGBM] [Warning] bagging_fraction is set=0.8898, subsample=0.8 will be
ignored. Current value: bagging_fraction=0.8898
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.821658   train's binary_logloss: 0.52673 valid's auc:
0.787217   valid's binary_logloss: 0.538755
[400]   train's auc: 0.8523     train's binary_logloss: 0.490813        valid's
auc: 0.791488   valid's binary_logloss: 0.515205
[600]   train's auc: 0.87459    train's binary_logloss: 0.464387        valid's
auc: 0.792149   valid's binary_logloss: 0.498592
Early stopping, best iteration is:
[542]   train's auc: 0.868653   train's binary_logloss: 0.471499        valid's
```

```
auc: 0.792288   valid's binary_logloss: 0.503009
[LightGBM] [Warning] feature_fraction is set=0.1926, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.1926
[LightGBM] [Warning] bagging_fraction is set=0.8898, subsample=0.8 will be
ignored. Current value: bagging_fraction=0.8898
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.822781   train's binary_logloss: 0.525578        valid's
auc: 0.782118   valid's binary_logloss: 0.541515
[400]   train's auc: 0.853315   train's binary_logloss: 0.48951 valid's auc:
0.787428   valid's binary_logloss: 0.517948
[600]   train's auc: 0.875759   train's binary_logloss: 0.462888        valid's
auc: 0.788355   valid's binary_logloss: 0.501286
Early stopping, best iteration is:
[604]   train's auc: 0.876122   train's binary_logloss: 0.46244 valid's auc:
0.788425   valid's binary_logloss: 0.500983
```

[ ]:
```
submission_f_2.to_csv('submission_f_2.csv', index = False)
```

9 Stacking

[ ]:
```
X = train.copy()
```

[ ]:
```
train=pd.read_csv('/Users/yufan/Downloads/train_imp_removed1.csv')
```

[ ]:
```
train.columns = [''.join (c if c.isalnum() else '_' for c in str(x)) for x in␣
 ↪train.columns]
```

[ ]:
```
test=pd.read_csv('/Users/yufan/Downloads/test_imp_removed1.csv')
```

[ ]:
```
test.columns = [''.join (c if c.isalnum() else '_' for c in str(x)) for x in␣
 ↪test.columns]
```

[ ]:
```
test1=test.drop(columns=train.columns[0],
        axis=1)
```

[ ]:
```
testid=test['SK_ID_CURR']
```

[ ]:
```
Y=train['TARGET']
```

[ ]:
```
X=train.drop(columns=train.columns[[0,1]],
        axis=1)
```

[ ]:
```
from sklearn.model_selection import train_test_split
```

[ ]:
```
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import StackingClassifier
```

```python
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(class_weight =␣
↪'balanced')
)
```

[ ]:
```python
X_train,X_test,Y_train,Y_test = train_test_split(X,Y, random_state=42)
```

[ ]:
```python
clf.fit(X_train, Y_train).score(X_test, Y_test)
```

```
[22:01:29] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-4
3e9a6c0910f/volume/xgboost-split_1619728204606/work/src/learner.cc:541:
Parameters: { nrounds } might not be used.

  This may not be accurate due to some parameters are only used in language
bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through
this
  verification. Please open an issue if you find above cases.


[LightGBM] [Warning] feature_fraction is set=0.1926, colsample_bytree=1.0 will
```

```
be ignored. Current value: feature_fraction=0.1926
[LightGBM] [Warning] bagging_fraction is set=0.8898, subsample=0.8 will be
ignored. Current value: bagging_fraction=0.8898
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set
num_leaves OR 2^max_depth > num_leaves. (num_leaves=31).
[LightGBM] [Warning] feature_fraction is set=0.1926, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.1926
[LightGBM] [Warning] bagging_fraction is set=0.8898, subsample=0.8 will be
ignored. Current value: bagging_fraction=0.8898
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set
num_leaves OR 2^max_depth > num_leaves. (num_leaves=31).
[LightGBM] [Warning] feature_fraction is set=0.1926, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.1926
[LightGBM] [Warning] bagging_fraction is set=0.8898, subsample=0.8 will be
ignored. Current value: bagging_fraction=0.8898
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set
num_leaves OR 2^max_depth > num_leaves. (num_leaves=31).
[LightGBM] [Warning] feature_fraction is set=0.1926, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.1926
[LightGBM] [Warning] bagging_fraction is set=0.8898, subsample=0.8 will be
ignored. Current value: bagging_fraction=0.8898
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set
num_leaves OR 2^max_depth > num_leaves. (num_leaves=31).
[23:18:24] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-4
3e9a6c0910f/volume/xgboost-split_1619728204606/work/src/learner.cc:541:
Parameters: { nrounds } might not be used.

  This may not be accurate due to some parameters are only used in language
bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through
this
  verification. Please open an issue if you find above cases.


[23:41:17] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-4
3e9a6c0910f/volume/xgboost-split_1619728204606/work/src/learner.cc:541:
Parameters: { nrounds } might not be used.

  This may not be accurate due to some parameters are only used in language
bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through
this
  verification. Please open an issue if you find above cases.


[00:04:12] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-4
3e9a6c0910f/volume/xgboost-split_1619728204606/work/src/learner.cc:541:
Parameters: { nrounds } might not be used.
```

```
  This may not be accurate due to some parameters are only used in language
bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through
this
  verification. Please open an issue if you find above cases.


[00:27:14] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-4
3e9a6c0910f/volume/xgboost-split_1619728204606/work/src/learner.cc:541:
Parameters: { nrounds } might not be used.

  This may not be accurate due to some parameters are only used in language
bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through
this
  verification. Please open an issue if you find above cases.


[00:50:12] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-4
3e9a6c0910f/volume/xgboost-split_1619728204606/work/src/learner.cc:541:
Parameters: { nrounds } might not be used.

  This may not be accurate due to some parameters are only used in language
bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through
this
  verification. Please open an issue if you find above cases.
```

[ ]: 0.919170633991519

[ ]:
```python
# predict the results
y_pred=clf.predict_proba(test1)[:, 1]
```

[ ]:
```python
y_pred
```

[ ]:
```python
submission = pd.DataFrame({'SK_ID_CURR': testid, 'TARGET': y_pred})
```

[ ]:
```python
submission.to_csv('submission.csv_f', index = False)
```

[ ]:
```python
```