With you today :

Eng. Ahmed Ashraf Khalil.

Data scientist

home credit risk

MY GP IN EPSILON AI INSTITUTE ABOUT PREDICTING WHO WILL BE ABLE TO PAY THE LOAN IN TIME

Data input features :

CNT_CHILDREN , AMT_TOTAL_INCOME , AMT_GOODS_PRICE,….

SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RELATIVE	DAYS_BIRTH	DAYS_
100002	0	202500.0	406597.5	24700.5	351000.0	0.018801	-9461	
100003	0	270000.0	1293502.5	35698.5	1129500.0	0.003541	-16765	
100004	0	67500.0	135000.0	6750.0	135000.0	0.010032	-19046	
100006	0	135000.0	312682.5	29686.5	297000.0	0.008019	-19005	
100007	0	121500.0	513000.0	21865.5	513000.0	0.028663	-19932	

Which:

CNT_CHILDREN : count of children

TOTAL_INCOME : Total income

GOODS_PRICE : Goods price

Machine learning procedure :

1- EDA

2- Preprocessing

3- Machine learning model

EDA :

* Exploring data types

* Nan values count in data

* statistical information about the data
    * central tendency
    * quartiles statistics

* relation between features and correlations

Exploring data types

```
print("data types in the dataframe : ",data.dtypes.unique(),"\n\n",data.dtypes)
```

```
data types in the dataframe :  [dtype('int64') dtype('O') dtype('float64')]

 SK_ID_CURR                     int64
NAME_CONTRACT_TYPE            object
CODE_GENDER                   object
FLAG_OWN_CAR                  object
FLAG_OWN_REALTY               object
                               ...
AMT_REQ_CREDIT_BUREAU_WEEK   float64
AMT_REQ_CREDIT_BUREAU_MON    float64
AMT_REQ_CREDIT_BUREAU_QRT    float64
AMT_REQ_CREDIT_BUREAU_YEAR   float64
is_train                       int64
Length: 122, dtype: object
```

Nan values count percentage in data

```
total = df[df.columns[df.isnull().any()==True]]
total.isnull().mean().sort_values(ascending=False)
```

```
COMMONAREA_MEDI            0.698723
COMMONAREA_AVG             0.698723
COMMONAREA_MODE            0.698723
NONLIVINGAPARTMENTS_AVG    0.694330
NONLIVINGAPARTMENTS_MODE   0.694330
                             ...
EXT_SOURCE_2               0.002146
AMT_GOODS_PRICE            0.000904
AMT_ANNUITY                0.000039
CNT_FAM_MEMBERS            0.000007
DAYS_LAST_PHONE_CHANGE     0.000003
Length: 67, dtype: float64
```

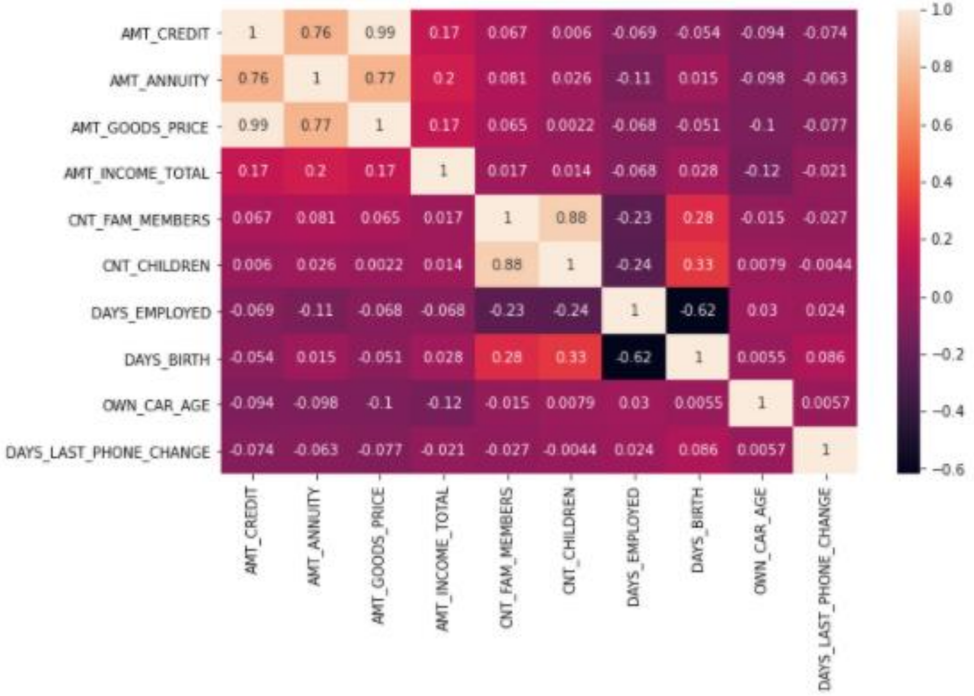
statistical information about the data

```
data.describe()
```

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RELATIVE	DAYS_BIR
count	356255.000000	356255.000000	3.562550e+05	3.562550e+05	356219.000000	3.559770e+05	356255.000000	356255.000
mean	278128.000000	0.414316	1.701161e+05	5.877674e+05	27425.560657	5.280200e+05	0.020917	-16041.248
std	102842.104413	0.720378	2.235068e+05	3.986237e+05	14732.808190	3.660650e+05	0.013915	4358.803
min	100001.000000	0.000000	2.565000e+04	4.500000e+04	1615.500000	4.050000e+04	0.000253	-25229.000
25%	189064.500000	0.000000	1.125000e+05	2.700000e+05	16731.000000	2.340000e+05	0.010006	-19676.000
50%	278128.000000	0.000000	1.530000e+05	5.002110e+05	25078.500000	4.500000e+05	0.018850	-15755.000
75%	367191.500000	1.000000	2.025000e+05	7.975575e+05	34960.500000	6.750000e+05	0.028663	-12425.000
max	456255.000000	20.000000	1.170000e+08	4.050000e+06	258025.500000	4.050000e+06	0.072508	-7338.000

8 rows × 106 columns

relation between features and correlations



Preprocessing :

* data cleaning
    * filling Nan values
    * deleting the Nan values

* removing outliers

* Feature Generation

* labeling categorical values

* Feature Scaling

Dealing with Nan values

```
df['DAYS_EMPLOYED'].fillna(df['DAYS_EMPLOYED'].mode()[0],inplace=True)
```

```
data['CNT_FAM_MEMBERS'].fillna(data['CNT_FAM_MEMBERS'].median(),inplace=True)
data['NEW_SCORES_STD'].fillna(data['NEW_SCORES_STD'].mean(),inplace=True)
```

```
def process_df(df, raw=False):
    df_nan = df[df.columns[df.isnull().any()]]
    df_na = df_nan[df_nan.columns[df_nan.isnull().mean()*100 < 50]]
    df_not_nan = df[df.columns[~df.isnull().any()]]
    df_na_cat = df_na.select_dtypes(include='O')
    for col in df_na_cat:
        df_na_cat[col].fillna(df_na_cat[col].mode()[0], inplace=True)
    df_na_num = df_na.select_dtypes(exclude='O')
    df_na_num = remove_outliers(df_na_num)
    df_na_num = df_na_num.fillna(df_na_num.mean())
#     if raw:
#         df = pd.concat([df_not_nan.drop('SK_ID_CURR', axis=1), df_na_cat, df_na_num], axis=1)
#     else:
    df = pd.concat([df_not_nan, df_na_cat, df_na_num], axis=1)
    df = pd.get_dummies(df, drop_first=True)
    return df
```

Removing outliers

```
def remove_outliers(df):
#     d1 = df.select_dtypes(include='O')
#     df = df.select_dtypes(exclude='O')
    Q1 = df.quantile(0.25)
    Q3 = df.quantile(0.75)
    IQR = Q3 - Q1
    df = df[~(df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))]
#     df = pd.concat([df, d1], axis=1)
    return df
```

Feature Generation

```
#feature generation and filling nana values
data['NEW_CREDIT_TO_ANNUITY_RATIO'] = data['AMT_CREDIT'] / data['AMT_ANNUITY']
data['NEW_CREDIT_TO_GOODS_RATIO'] = data['AMT_CREDIT'] / data['AMT_GOODS_PRICE']
data['NEW_ANNUITY_TO_INCOME_RATIO'] = data['AMT_ANNUITY'] / data['AMT_INCOME_TOTAL']
data['NEW_CREDIT_TO_INCOME_RATIO'] = data['AMT_CREDIT'] / data['AMT_INCOME_TOTAL']
data['NEW_INC_PER_MEMB'] = data['AMT_INCOME_TOTAL'] / data['CNT_FAM_MEMBERS']
data['NEW_INC_PER_CHLD'] = data['AMT_INCOME_TOTAL'] / (1 + data['CNT_CHILDREN'])
```

```
data['NEW_EMPLOY_TO_BIRTH_RATIO'] = data['DAYS_EMPLOYED'] / data['DAYS_BIRTH']
data['NEW_CAR_TO_BIRTH_RATIO'] = data['OWN_CAR_AGE'] / data['DAYS_BIRTH']
data['NEW_CAR_TO_EMPLOY_RATIO'] = data['OWN_CAR_AGE'] / data['DAYS_EMPLOYED']
data['NEW_PHONE_TO_BIRTH_RATIO'] = data['DAYS_LAST_PHONE_CHANGE'] / data['DAYS_BIRTH']

data['NEW_SOURCES_PROD'] = data['EXT_SOURCE_1'] * data['EXT_SOURCE_2'] * data['EXT_SOURCE_3']
data['NEW_EXT_SOURCES_MEAN'] = data[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3']].mean(axis=1)
data['NEW_SCORES_STD'] = data[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3']].std(axis=1)
```

Feature scaling

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(trainX)
trainX, testX = scaler.transform(trainX), scaler.transform(testX)
```

Feature scaling

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(trainX)
trainX, testX = scaler.transform(trainX), scaler.transform(testX)
```

Labeling categorical values

FONDKAPREMONT_MODE_reg oper spec account	HOUSETYPE_MODE_specific housing	HOUSETYPE_MODE_terraced house	WALLSMATERIAL_MODE_Mixed	WALLSMATERIAL_MODE_Monolithic
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

The Machine learning model :

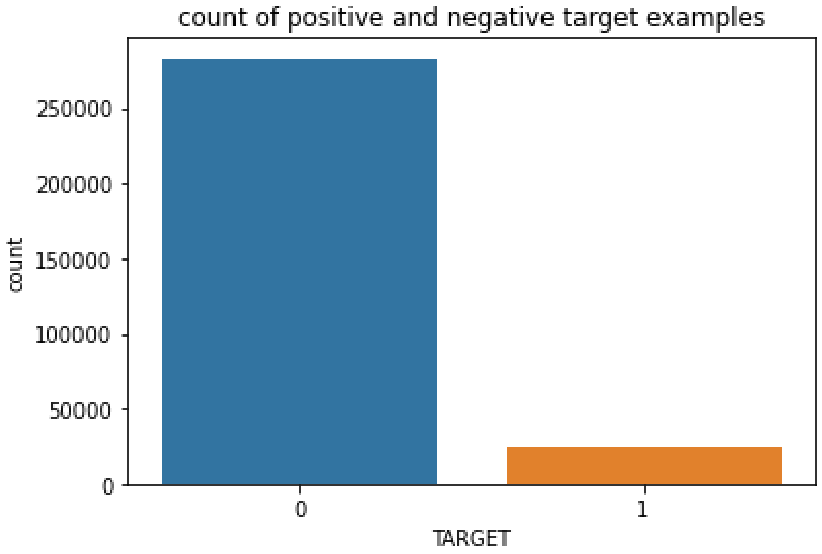
* Deal with imbalanced target label

* Choose the learning parameters

* Evaluate the module

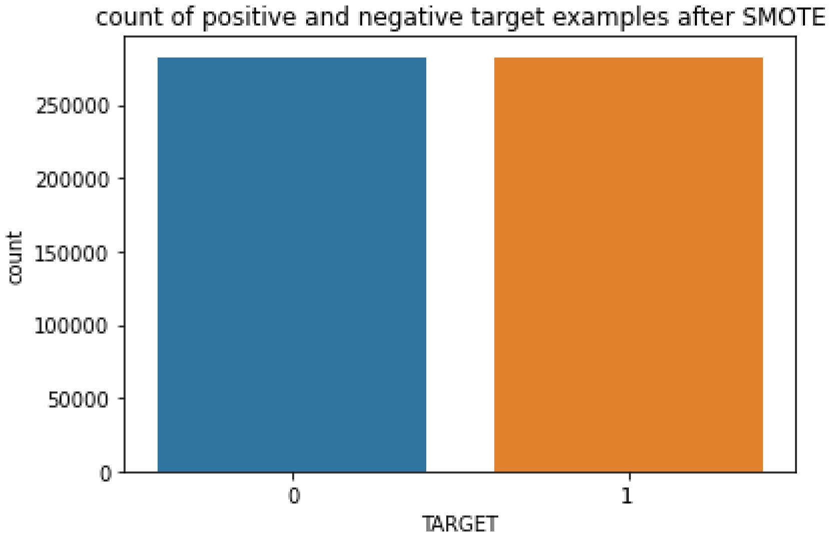
Deal with imbalanced target label

before:




```
0      282686
1       24825
```

after :




```
1      282686
0      282686
```

Choose the learning parameters

```
params = {'task': 'train', 'boosting_type': 'gbdt', 'objective': 'binary', 'metric': 'auc',
          'learning_rate': 0.01, 'num_leaves': 48, 'num_iteration': 5000, 'verbose': 0 ,
          'colsample_bytree':.8, 'subsample':.9, 'max_depth':7, 'reg_alpha':.1, 'reg_lambda':.1,
          'min_split_gain':.01, 'min_child_weight':1}
model = lgb.train(params, lgb_train, valid_sets=lgb_eval, early_stopping_rounds=100, verbose_eval=20)
```

Evaluate the module locally

```
[3320]  valid_0's auc: 0.981489
Early stopping, best iteration is:
[3228]  valid_0's auc: 0.981492
```

Evaluate the model by kaggle

Name	Submitted	Wait time	Execution time	Score
sub.csv	4 days ago	1 seconds	1 seconds	0.78259

Complete

Thank you