Final Project Submission

Please fill out:

Student name: Yeonjae Zhang

Student pace: full time

Scheduled project review date/time: April 22nd, 2022 Friday

Instructor name: Praveen Gowtham

Blog post URL: https://msyeon.blogspot.com/2022/04/imbalance-data-treatment.html

Overview

Banks lose moneys from loan defaulters. I will build a prediction model of loan defaulter. This model will help banks to reduce loan default risk.

Import Modules

Import neccesary modules.

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from imblearn.over_sampling import SMOTE
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import accuracy_score, f1_score, plot_confusion_matrix, recall_score
from imblearn.under_sampling import RandomUnderSampler, NearMiss
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
pd.set_option('display.max_columns', None)
plt.rcParams.update({'font.size': 10})
```

Data Understanding

Look into the given data.

In [2]:
```
df = pd.read_csv('data/application_data.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	NAMI
0	100002	1	Cash loans	M	N	Y	0	202500.0	406597.5	24700.5	351000.0	U
1	100003	0	Cash loans	F	N	N	0	270000.0	1293502.5	35698.5	1129500.0	
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	135000.0	6750.0	135000.0	U
3	100006	0	Cash loans	F	N	Y	0	135000.0	312682.5	29686.5	297000.0	U
4	100007	0	Cash loans	M	N	Y	0	121500.0	513000.0	21865.5	513000.0	U

There are too many columns and missing values.

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
```

In [5]:
```
df.describe()
```

Out[5]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RELATIVE	DAYS_BIRTH	DAYS_EMPLOYED	DAY
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.000000	3.072330e+05	307511.000000	307511.000000	307511.000000	
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.573909	5.383962e+05	0.020868	-16036.995067	63815.045904	
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.737315	3.694465e+05	0.013831	4363.988632	141275.766519	
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.500000	4.050000e+04	0.000290	-25229.000000	-17912.000000	
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.000000	2.385000e+05	0.010006	-19682.000000	-2760.000000	
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.000000	4.500000e+05	0.018850	-15750.000000	-1213.000000	
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.000000	6.795000e+05	0.028663	-12413.000000	-289.000000	
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.500000	4.050000e+06	0.072508	-7489.000000	365243.000000	

Data Preparation

Select relevant coloumns and clean missing values.

Feature Engineering

In [6]:
```
dummy_df = pd.get_dummies(df).copy()
```

In [7]:
```
impr_columns = abs(dummy_df.corr()).loc['TARGET', :].sort_values(axis=0, ascending=False).index[:20]
```

In [8]:
```
impr_df = dummy_df[impr_columns]
impr_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 20 columns):
 #   Column                                       Non-Null Count   Dtype  
---  ------                                       --------------   -----  
 0   TARGET                                       307511 non-null  int64  
 1   EXT_SOURCE_3                                 246546 non-null  float64
 2   EXT_SOURCE_2                                 306851 non-null  float64
 3   EXT_SOURCE_1                                 134133 non-null  float64
 4   DAYS_BIRTH                                   307511 non-null  int64  
 5   REGION_RATING_CLIENT_W_CITY                  307511 non-null  int64  
 6   REGION_RATING_CLIENT                         307511 non-null  int64  
 7   NAME_INCOME_TYPE_Working                     307511 non-null  uint8  
 8   NAME_EDUCATION_TYPE_Higher education         307511 non-null  uint8  
 9   DAYS_LAST_PHONE_CHANGE                       307510 non-null  float64
 10  CODE_GENDER_M                                307511 non-null  uint8  
 11  CODE_GENDER_F                                307511 non-null  uint8  
 12  DAYS_ID_PUBLISH                              307511 non-null  int64  
 13  REG_CITY_NOT_WORK_CITY                       307511 non-null  int64  
 14  NAME_EDUCATION_TYPE_Secondary / secondary special  307511 non-null  uint8  
 15  NAME_INCOME_TYPE_Pensioner                   307511 non-null  uint8  
 16  ORGANIZATION_TYPE_XNA                        307511 non-null  uint8  
 17  FLAG_EMP_PHONE                               307511 non-null  int64  
 18  DAYS_EMPLOYED                                307511 non-null  int64  
 19  REG_CITY_NOT_LIVE_CITY                       307511 non-null  int64  
dtypes: float64(4), int64(9), uint8(7)
memory usage: 32.6 MB
```

Data Cleaning

Drop nulls.

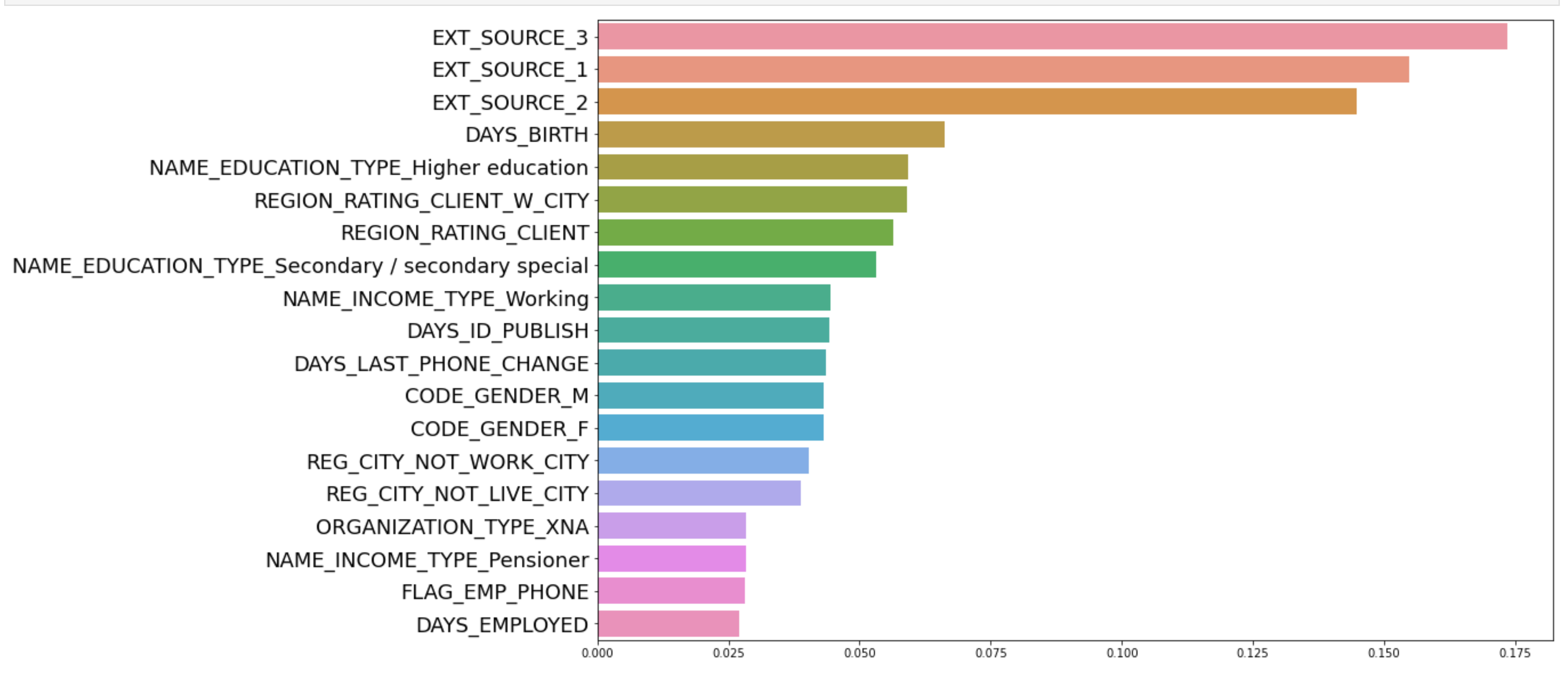
In [9]:
```
cleaned_df = impr_df.dropna()
```

Data Analysis

Visualize what columns are considered to the related column.

In [10]:
```
corr = abs(cleaned_df.corr().iloc[0, 1:]).sort_values(ascending=False)
```

In [11]:
```
plt.figure(figsize=(15,10))
sns.barplot(x=corr.values, y=corr.index);
plt.yticks(fontsize=18);
```



Modeling

Split data to train and test

In [12]:
```
y = cleaned_df['TARGET']
X = cleaned_df.drop('TARGET', axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, stratify=y)
X_train, y_train = RandomUnderSampler(random_state=7).fit_resample(X_train, y_train)
```

Model Selection

Find best performace model for the given data.

In [13]:
```
# Compare Scores
models = {'Logistic Regression': LogisticRegression(max_iter=5000, random_state=42),
          'DecisionTree': DecisionTreeClassifier(),
          'KNeighbor': KNeighborsClassifier(),
          'LinearSVC': LinearSVC(max_iter=5000, random_state=42),
          'RandomForest': RandomForestClassifier(random_state=42),
         }

print('Model Comparison')
for key in models:
    pipe = Pipeline([('impute', SimpleImputer()),('scaler', MinMaxScaler()), ('model', models[key])])
    score = cross_val_score(pipe, X_train, y_train, scoring='f1', cv=5).mean()
    print('------------------')
    print(f'{key} CrossValidation:', score)
```

```
Model Comparison
------------------
Logistic Regression CrossValidation: 0.6839157536189202
------------------
DecisionTree CrossValidation: 0.5885555852109012
------------------
KNeighbor CrossValidation: 0.6141939000467416
------------------
LinearSVC CrossValidation: 0.684085751084041
------------------
RandomForest CrossValidation: 0.6694483519795893
```

Feature Selection

Find best performace feature numbers.

9 is the number of features for the highest f1 score

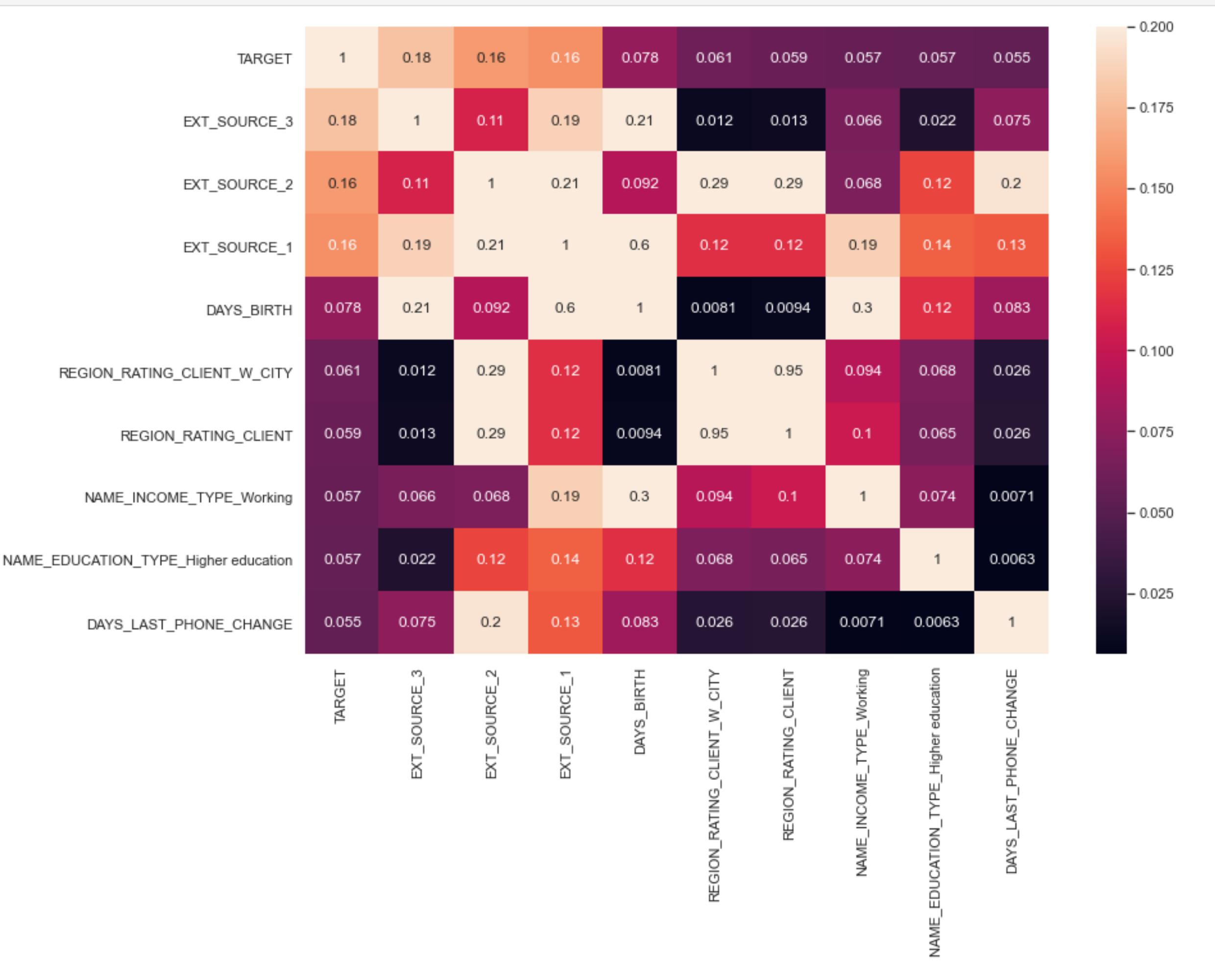
In [14]:
```
print('Model Comparison')
for n_features in range(3, 21, 1):
    rtree_pipe = Pipeline([('impute', SimpleImputer()),('scaler', MinMaxScaler()), ('model', LinearSVC(random_state=42))])
    score = cross_val_score(rtree_pipe, X_train.iloc[:, :n_features], y_train, scoring='f1', cv=5).mean()
    print('------------------')
    print(f'{n_features} Features CrossValidation:', score)
```

```
Model Comparison
------------------
3 Features CrossValidation: 0.6781234967695634
------------------
4 Features CrossValidation: 0.6787258055732138
------------------
5 Features CrossValidation: 0.6802656983490047
------------------
6 Features CrossValidation: 0.6807174395051399
------------------
7 Features CrossValidation: 0.6808175942893369
------------------
8 Features CrossValidation: 0.682652886294896
------------------
9 Features CrossValidation: 0.684203622839396
------------------
10 Features CrossValidation: 0.6838493268591591
------------------
11 Features CrossValidation: 0.6839074126959315
------------------
12 Features CrossValidation: 0.6831956854226368
------------------
13 Features CrossValidation: 0.6834754962640073
------------------
14 Features CrossValidation: 0.6835380422368255
------------------
15 Features CrossValidation: 0.6829301908727866
------------------
16 Features CrossValidation: 0.6829301908727866
------------------
17 Features CrossValidation: 0.6829863382957638
------------------
18 Features CrossValidation: 0.6836334693294848
------------------
19 Features CrossValidation: 0.684085751084041
------------------
20 Features CrossValidation: 0.684085751084041
```

In [15]:
```
X_train, X_test = X_train.iloc[:, :9], X_test.iloc[:, :9]
```

Visualize the selected features.

In [16]:
```
sns.set(rc = {'figure.figsize':(13,9)})
sns.heatmap(abs(dummy_df[impr_columns[:10]].corr()), vmax=0.2, annot=True);
```



Model Tuning

Set the baseline for comparison

In [17]:
```
print('Baseline before Tuning')
print('LinearSVC F1 CrossValidation: 0.684203622839396')
```

```
Baseline before Tuning
LinearSVC F1 CrossValidation: 0.684203622839396
```

GridSearch Crossvalidation

0.0005 increased from tuning

In [18]:
```
svc_pipe = Pipeline([('impute', SimpleImputer()),('scaler', MinMaxScaler()), ('model', LinearSVC(random_state=42, max_iter=10000))])
parameters = {'model__C': [3, 4, 5, 6]}
result = GridSearchCV(svc_pipe, parameters, cv=5, scoring='f1').fit(X_train, y_train)
print('Grid Search')
print(result.best_score_, result.best_params_)
```

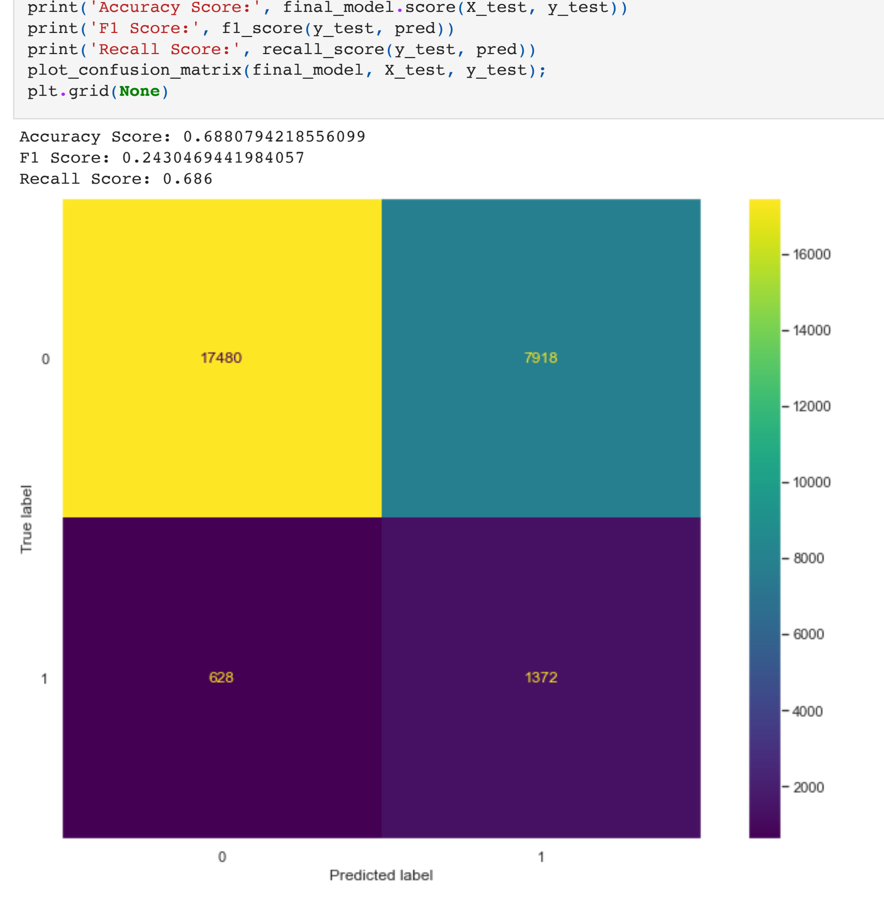
```
Grid Search
0.6847605446316545 {'model__C': 4}
```

In [19]:
```
final_model = result.best_estimator_
```

Final Score - Test Dataset

In [20]:
```
# TEST Score
final_model.fit(X_train, y_train)
pred = final_model.predict(X_test)
print('Accuracy Score:', final_model.score(X_test, y_test))
print('F1 Score:', f1_score(y_test, pred))
print('Recall Score:', recall_score(y_test, pred))
plot_confusion_matrix(final_model, X_test, y_test);
plt.grid(None)
```

```
Accuracy Score: 0.6880794218556099
F1 Score: 0.2430469441984057
Recall Score: 0.686
```



Apply to business

Bank interest income rate: 3%

Bank loan default recovery: 70%

In [21]:
```
print('From 27,398 future client')
print('----------------------------------')
print('If approve all loans without model')
borrower = 25398
default = 2000
loan = 1000000

result1 = borrower*loan*0.03 - default*loan*0.3
print('Bank earned ${}'.format(result1))
print('----------------------------------')
print('If approve loans with model')
borrower = 17480
default = 628
loan = 1000000

result2 = borrower*loan*0.03 - default*loan*0.3
print('Bank earned ${}'.format(result2))
print('----------------------------------')
print('Earning Differecnce:', result2-result1)
```

```
From 27,398 future client
----------------------------------
If approve all loans without model
Bank earned $161940000.0
----------------------------------
If approve loans with model
Bank earned $336000000.0
----------------------------------
Earning Differecnce: 174060000.0
```

Conclusion

If bank use our model to predict loan defaulter, bank will save 174,060,000 dollars from 27,398 future clients.

In [ ]: