High-Impact Lending Decisions

September 14, 2024

In the banking sector, the decision to grant loans is a critical process that hinges on a meticulous evaluation of various customer variables. Factors such as credit score, income stability, employment history, existing debt levels, and financial behavior are thoroughly analyzed to assess the creditworthiness of potential borrowers. This rigorous assessment is essential for minimizing default risks and ensuring the financial stability of the institution. Banks employ sophisticated statistical models and data analytics to make informed lending decisions, which helps in predicting the likelihood of repayment and identifying potential red flags.

The necessity of such a comprehensive evaluation process cannot be overstated. By effectively managing and mitigating risks, banks protect their capital reserves, thereby maintaining liquidity and operational viability. Furthermore, this approach enables banks to extend credit to reliable customers, fostering economic expansion and enhancing customer relationships. Through prudent lending practices, banks support both individual financial goals and broader economic growth. The systematic approach to loan approval, incorporating a balance of risk and opportunity, is indispensable for the sustainable growth and profitability of financial institutions. It ensures that banks can continue to play a pivotal role in the financial ecosystem by facilitating investment, consumption, and economic stability.

0.1 The Libraries Required

[2]:
```
!pip install xgboost
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: xgboost in
c:\users\sayak\appdata\roaming\python\python311\site-packages (2.1.0)
Requirement already satisfied: numpy in e:\anaconda\lib\site-packages (from
xgboost) (1.24.3)
Requirement already satisfied: scipy in e:\anaconda\lib\site-packages (from
xgboost) (1.11.1)
```

[94]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from xgboost import XGBClassifier as xgb
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score
from scipy.stats import chi2_contingency
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```python
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪precision_recall_fscore_support
import warnings
import os
```

0.2 The Dataset

In this project, I am going to work with two datasets, which contains information regarding 50k+ customers of some bank along with their personal information like, marital status, education, credit score and so on.

[4]:
```python
a1 = pd.read_excel("case_study1.xlsx")
a2 = pd.read_excel("case_study2.xlsx")
```

[5]:
```python
df1 = a1.copy()
df2 = a2.copy()
```

[6]:
```python
print(df1.shape)
print(df2.shape)
```

```
(51336, 26)
(51336, 62)
```

[7]:
```python
df1.head()
```

[7]:
```
   PROSPECTID  Total_TL  Tot_Closed_TL  Tot_Active_TL  Total_TL_opened_L6M  \
0           1         5              4              1                    0
1           2         1              0              1                    0
2           3         8              0              8                    1
3           4         1              0              1                    1
4           5         3              2              1                    0

   Tot_TL_closed_L6M  pct_tl_open_L6M  pct_tl_closed_L6M  pct_active_tl  \
0                  0            0.000                0.0          0.200
1                  0            0.000                0.0          1.000
2                  0            0.125                0.0          1.000
3                  0            1.000                0.0          1.000
4                  0            0.000                0.0          0.333

   pct_closed_tl  …  CC_TL  Consumer_TL  Gold_TL  Home_TL  PL_TL  \
0          0.800  …      0            0        1        0      4
1          0.000  …      0            1        0        0      0
2          0.000  …      0            6        1        0      0
3          0.000  …      0            0        0        0      0
4          0.667  …      0            0        0        0      0
```

```
   Secured_TL  Unsecured_TL  Other_TL  Age_Oldest_TL  Age_Newest_TL
0           1             4         0             72             18
1           0             1         0              7              7
2           2             6         0             47              2
3           0             1         1              5              5
4           3             0         2            131             32

[5 rows x 26 columns]
```

[77]:
```
df2.head()
```

[77]:
```
   PROSPECTID  time_since_recent_payment  num_times_delinquent  \
0           1                        549                    11
1           2                         47                     0
2           3                        302                     9
4           5                        583                     0
5           6                        245                    14

   max_recent_level_of_deliq  num_deliq_6mts  num_deliq_12mts  \
0                         29               0                0
1                          0               0                0
2                         25               1                9
4                          0               0                0
5                        270               0                0

   num_deliq_6_12mts  num_times_30p_dpd  num_times_60p_dpd  num_std  …  \
0                  0                  0                  0       21  …
1                  0                  0                  0        0  …
2                  8                  0                  0       10  …
4                  0                  0                  0       53  …
5                  0                 13                 11        5  …

   pct_PL_enq_L6m_of_L12m  pct_CC_enq_L6m_of_L12m  pct_PL_enq_L6m_of_ever  \
0                     0.0                     0.0                   0.000
1                     0.0                     0.0                   0.000
2                     0.0                     0.0                   0.000
4                     0.0                     0.0                   0.000
5                     1.0                     0.0                   0.429

   pct_CC_enq_L6m_of_ever  HL_Flag  GL_Flag  last_prod_enq2  first_prod_enq2  \
0                     0.0        1        0              PL               PL
1                     0.0        0        0    ConsumerLoan     ConsumerLoan
2                     0.0        1        0    ConsumerLoan           others
4                     0.0        0        0              AL               AL
5                     0.0        1        0    ConsumerLoan               PL

   Credit_Score  Approved_Flag
```

```
0            696            P2
1            685            P2
2            693            P2
4            753            P1
5            668            P3

[5 rows x 54 columns]
```

Clearly the datasets needed to be cleaned and pre-processed before it can be used for modelling. So, we shall start with possible "Exploratory Data Analysis"

0.3 Exploratory Data Analysis

Remove Nulls When companies gathers data, instead of keeping a cell blank or absurd, it tends to input a pre-decided value. In this scenario, the value "-99999" is one of such. So we shall try to remove or impute this value considering it as NULL/NA value.

[9]:
```python
#In case of first data, df1. Only one column has -99999 values that is
 ↪"Age_Oldest_TL" column
df = df1.loc[df1['Age_Oldest_TL'] == -99999]
df.shape[0]
```

[9]: 40

Only 40 rows has this value, so we can just remove those rows as our dataset is huge this small number wont affect.

[10]:
```python
df1 = df1.loc[df1['Age_Oldest_TL'] != -99999]
df1.shape
```

[10]: (51296, 26)

For the second dataset, df2 we shall adopt a scheme for handling this NA value:- If the number of "-99999" values are more than 10000 we shall drop the column, else we shall drop the number of rows. At the end if we can retain more than 80% of the data, we are good to go.

[11]:
```python
# Here we figure out the columns that are to be removed with this scheme

columns_to_be_removed = []
for i in df2.columns:
    if df2.loc[df2[i] == -99999].shape[0] > 10000:
        columns_to_be_removed .append(i)
```

[12]:
```python
print(columns_to_be_removed)
df2 = df2.drop(columns_to_be_removed, axis =1)
```

```
['time_since_first_deliquency', 'time_since_recent_deliquency',
'max_delinquency_level', 'max_deliq_6mts', 'max_deliq_12mts', 'CC_utilization',
'PL_utilization', 'max_unsec_exposure_inPct']
```

[13]:
```
df2.shape
```

[13]: (51336, 54)

[14]:
```
# To remove the corresponding rows from the dataset

for i in df2.columns:
    df2 = df2.loc[df2[i] != -99999]
```

[15]:
```
df2.shape
```

[15]: (42066, 54)

Ultimately, more than 80% of the data is retained after removal of NULL values. So we shall proceed

Now, our next step is to merge the datasets with respect to some columns

[16]:
```
# To figure out any common column in both
for i in list(df1.columns):
    if i in list(df2.columns):
        print(i)
```

```
PROSPECTID
```

[17]:
```
# We shall merge the two datasets based on this column
# We will use inner join as we need all the common data in these two datasets.
df = pd. merge ( df1, df2, how ='inner', left_on = ['PROSPECTID'], right_on =␣
↪['PROSPECTID'] )
df.shape
```

[17]: (42064, 79)

[18]:
```
# Number of categorical variables
for i in df.columns:
    if df[i].dtype=='object':
        print(i)
```

```
MARITALSTATUS
EDUCATION
GENDER
last_prod_enq2
first_prod_enq2
Approved_Flag
```

[19]:
```
for i in ['MARITALSTATUS', 'EDUCATION', 'GENDER', 'last_prod_enq2',␣
↪'first_prod_enq2']:
    chi2, pval, _, _ = chi2_contingency(pd.crosstab(df[i], df['Approved_Flag']))
    print(i, '---', pval)
```

```
MARITALSTATUS --- 3.578180861038862e-233
EDUCATION --- 2.6942265249737532e-30
GENDER --- 1.907936100186563e-05
last_prod_enq2 --- 0.0
first_prod_enq2 --- 7.84997610555419e-287
```

Since all the categorical variables has pvalue < 0.05, we will accept all

[20]:
```
# Now we shall check multicollinearity in the numerical columns
numeric_columns=[]
for i in df.columns:
    if df[i].dtype != 'object':
        numeric_columns.append(i)
```

[21]:
```
vif_data=df[numeric_columns]
total_columns=vif_data.shape[1]
columns_to_be_kept=[]
column_index=0
```

[22]:
```
for i in range (0,total_columns):

    vif_value = variance_inflation_factor(vif_data, column_index)


    if vif_value <= 6:
        columns_to_be_kept.append( numeric_columns[i] )
        column_index = column_index+1

    else:
        print([numeric_columns[i]])
        vif_data = vif_data.drop([ numeric_columns[i] ] , axis=1)
```

```
E:\Anaconda\Lib\site-packages\statsmodels\stats\outliers_influence.py:198:
RuntimeWarning: divide by zero encountered in scalar divide
  vif = 1. / (1. - r_squared_i)
```

```
['Total_TL']
```

```
E:\Anaconda\Lib\site-packages\statsmodels\stats\outliers_influence.py:198:
RuntimeWarning: divide by zero encountered in scalar divide
  vif = 1. / (1. - r_squared_i)
```

```
['Tot_Closed_TL']
['Tot_Active_TL']
['Total_TL_opened_L6M']
['Tot_TL_closed_L6M']
```

```
E:\Anaconda\Lib\site-packages\statsmodels\stats\outliers_influence.py:198:
RuntimeWarning: divide by zero encountered in scalar divide
  vif = 1. / (1. - r_squared_i)
```

```
['pct_active_tl']
['pct_closed_tl']
['Total_TL_opened_L12M']
['pct_tl_open_L12M']
```

```
E:\Anaconda\Lib\site-packages\statsmodels\stats\outliers_influence.py:198:
RuntimeWarning: divide by zero encountered in scalar divide
  vif = 1. / (1. - r_squared_i)
```

```
['Auto_TL']
['Consumer_TL']
['Gold_TL']
['num_times_delinquent']
```

```
E:\Anaconda\Lib\site-packages\statsmodels\stats\outliers_influence.py:198:
RuntimeWarning: divide by zero encountered in scalar divide
  vif = 1. / (1. - r_squared_i)
```

```
['num_deliq_6mts']
['num_deliq_12mts']
['num_times_30p_dpd']
['num_std']
['num_std_6mts']
['num_dbt_6mts']
['num_lss_6mts']
['tot_enq']
['CC_enq']
['CC_enq_L6m']
['PL_enq']
['PL_enq_L6m']
['enq_L12m']
['enq_L6m']
['AGE']
['pct_of_active_TLs_ever']
['pct_opened_TLs_L6m_of_L12m']
['pct_PL_enq_L6m_of_L12m']
['pct_CC_enq_L6m_of_L12m']
['Credit_Score']
```

[23]:
```
# The remaining columns are :-
print(columns_to_be_kept)
```

```
['PROSPECTID', 'pct_tl_open_L6M', 'pct_tl_closed_L6M', 'Tot_TL_closed_L12M',
'pct_tl_closed_L12M', 'Tot_Missed_Pmnt', 'CC_TL', 'Home_TL', 'PL_TL',
'Secured_TL', 'Unsecured_TL', 'Other_TL', 'Age_Oldest_TL', 'Age_Newest_TL',
'time_since_recent_payment', 'max_recent_level_of_deliq', 'num_deliq_6_12mts',
'num_times_60p_dpd', 'num_std_12mts', 'num_sub', 'num_sub_6mts',
'num_sub_12mts', 'num_dbt', 'num_dbt_12mts', 'num_lss', 'num_lss_12mts',
'recent_level_of_deliq', 'CC_enq_L12m', 'PL_enq_L12m', 'time_since_recent_enq',
'enq_L3m', 'NETMONTHLYINCOME', 'Time_With_Curr_Empr', 'pct_currentBal_all_TL',
```

'CC_Flag', 'PL_Flag', 'pct_PL_enq_L6m_of_ever', 'pct_CC_enq_L6m_of_ever', 'HL_Flag', 'GL_Flag']

Finally we shall test for difference in means, by performing ANOVA test

[24]:
```python
# check Anova for columns_to_be_kept

from scipy.stats import f_oneway

columns_to_be_kept_numerical = []

for i in columns_to_be_kept:
    a = list(df[i])
    b = list(df['Approved_Flag'])

    group_P1 = [value for value, group in zip(a, b) if group == 'P1']
    group_P2 = [value for value, group in zip(a, b) if group == 'P2']
    group_P3 = [value for value, group in zip(a, b) if group == 'P3']
    group_P4 = [value for value, group in zip(a, b) if group == 'P4']


    f_statistic, p_value = f_oneway(group_P1, group_P2, group_P3, group_P4)

    if p_value <= 0.05:
        columns_to_be_kept_numerical.append(i)
```

So the final set of features are :-

[25]:
```python
features = columns_to_be_kept_numerical + ['MARITALSTATUS', 'EDUCATION',␣
↪'GENDER', 'last_prod_enq2', 'first_prod_enq2']
df = df[features + ['Approved_Flag']]
df.describe()
```

[25]:

	PROSPECTID	pct_tl_open_L6M	pct_tl_closed_L6M	Tot_TL_closed_L12M	\
count	42064.000000	42064.000000	42064.000000	42064.000000	
mean	25649.827477	0.179032	0.097783	0.825504	
std	14844.173396	0.278043	0.210957	1.537208	
min	1.000000	0.000000	0.000000	0.000000	
25%	12776.750000	0.000000	0.000000	0.000000	
50%	25706.500000	0.000000	0.000000	0.000000	
75%	38518.250000	0.333000	0.100000	1.000000	
max	51336.000000	1.000000	1.000000	33.000000	

	pct_tl_closed_L12M	Tot_Missed_Pmnt	CC_TL	Home_TL	\
count	42064.000000	42064.000000	42064.000000	42064.000000	
mean	0.160365	0.525746	0.145921	0.076241	
std	0.258831	1.106442	0.549314	0.358582	
min	0.000000	0.000000	0.000000	0.000000	

```
25%              0.000000          0.000000        0.000000        0.000000
50%              0.000000          0.000000        0.000000        0.000000
75%              0.250000          1.000000        0.000000        0.000000
max              1.000000         34.000000       27.000000       10.000000

              PL_TL    Secured_TL  …  time_since_recent_enq        enq_L3m  \
count  42064.000000  42064.000000  …           42064.000000   42064.000000
mean       0.328000      2.921334  …             264.854507       1.230458
std        0.916368      6.379764  …             466.585002       2.069461
min        0.000000      0.000000  …               0.000000       0.000000
25%        0.000000      0.000000  …               9.000000       0.000000
50%        0.000000      1.000000  …              79.000000       1.000000
75%        0.000000      3.000000  …             302.000000       2.000000
max       29.000000    235.000000  …            4768.000000      42.000000

       NETMONTHLYINCOME  Time_With_Curr_Empr       CC_Flag       PL_Flag  \
count      4.206400e+04         42064.000000  42064.000000  42064.000000
mean       2.692990e+04           110.345783      0.102962      0.193063
std        2.084300e+04            75.629967      0.303913      0.394707
min        0.000000e+00             0.000000      0.000000      0.000000
25%        1.800000e+04            61.000000      0.000000      0.000000
50%        2.400000e+04            92.000000      0.000000      0.000000
75%        3.100000e+04           131.000000      0.000000      0.000000
max        2.500000e+06          1020.000000      1.000000      1.000000

       pct_PL_enq_L6m_of_ever  pct_CC_enq_L6m_of_ever       HL_Flag  \
count            42064.000000            42064.000000  42064.000000
mean                 0.195497                0.064186      0.252235
std                  0.367414                0.225989      0.434300
min                  0.000000                0.000000      0.000000
25%                  0.000000                0.000000      0.000000
50%                  0.000000                0.000000      0.000000
75%                  0.000000                0.000000      1.000000
max                  1.000000                1.000000      1.000000

            GL_Flag
count  42064.000000
mean       0.056580
std        0.231042
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        1.000000

[8 rows x 38 columns]
```

0.3.1 Label Encoding

Now we shall encode the categorical variables so that it will be easier to work with during modelling.

[26]:
```
for i in df.columns:
    if df[i].dtype=='object':
        print(i)
```

```
MARITALSTATUS
EDUCATION
GENDER
last_prod_enq2
first_prod_enq2
Approved_Flag
```

In the Education column, the variables are ordinal, but the other columns doesnt have such ordinal characteristics

[27]:
```
# We manually encoded the levels in EDUCATION column

df.loc[df['EDUCATION'] == 'SSC',['EDUCATION']]              = 1
df.loc[df['EDUCATION'] == '12TH',['EDUCATION']]             = 2
df.loc[df['EDUCATION'] == 'GRADUATE',['EDUCATION']]         = 3
df.loc[df['EDUCATION'] == 'UNDER GRADUATE',['EDUCATION']]   = 3
df.loc[df['EDUCATION'] == 'POST-GRADUATE',['EDUCATION']]    = 4
df.loc[df['EDUCATION'] == 'OTHERS',['EDUCATION']]           = 1
df.loc[df['EDUCATION'] == 'PROFESSIONAL',['EDUCATION']]     = 5
df['EDUCATION'].value_counts()
df['EDUCATION'] = df['EDUCATION'].astype(int)
```

[28]:
```
df_encoded = pd.get_dummies(df, columns=['MARITALSTATUS','GENDER',␣
 ↪'last_prod_enq2' ,'first_prod_enq2'])
```

[29]:
```
df_encoded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42064 entries, 0 to 42063
Data columns (total 56 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   PROSPECTID                    42064 non-null  int64
 1   pct_tl_open_L6M               42064 non-null  float64
 2   pct_tl_closed_L6M             42064 non-null  float64
 3   Tot_TL_closed_L12M            42064 non-null  int64
 4   pct_tl_closed_L12M            42064 non-null  float64
 5   Tot_Missed_Pmnt               42064 non-null  int64
 6   CC_TL                         42064 non-null  int64
 7   Home_TL                       42064 non-null  int64
 8   PL_TL                         42064 non-null  int64
 9   Secured_TL                    42064 non-null  int64
```

```
 10  Unsecured_TL                 42064 non-null  int64
 11  Other_TL                     42064 non-null  int64
 12  Age_Oldest_TL                42064 non-null  int64
 13  Age_Newest_TL                42064 non-null  int64
 14  time_since_recent_payment    42064 non-null  int64
 15  max_recent_level_of_deliq    42064 non-null  int64
 16  num_deliq_6_12mts            42064 non-null  int64
 17  num_times_60p_dpd            42064 non-null  int64
 18  num_std_12mts                42064 non-null  int64
 19  num_sub                      42064 non-null  int64
 20  num_sub_6mts                 42064 non-null  int64
 21  num_sub_12mts                42064 non-null  int64
 22  num_dbt                      42064 non-null  int64
 23  num_dbt_12mts                42064 non-null  int64
 24  num_lss                      42064 non-null  int64
 25  recent_level_of_deliq        42064 non-null  int64
 26  CC_enq_L12m                  42064 non-null  int64
 27  PL_enq_L12m                  42064 non-null  int64
 28  time_since_recent_enq        42064 non-null  int64
 29  enq_L3m                      42064 non-null  int64
 30  NETMONTHLYINCOME             42064 non-null  int64
 31  Time_With_Curr_Empr          42064 non-null  int64
 32  CC_Flag                      42064 non-null  int64
 33  PL_Flag                      42064 non-null  int64
 34  pct_PL_enq_L6m_of_ever       42064 non-null  float64
 35  pct_CC_enq_L6m_of_ever       42064 non-null  float64
 36  HL_Flag                      42064 non-null  int64
 37  GL_Flag                      42064 non-null  int64
 38  EDUCATION                    42064 non-null  int32
 39  Approved_Flag                42064 non-null  object
 40  MARITALSTATUS_Married        42064 non-null  bool
 41  MARITALSTATUS_Single         42064 non-null  bool
 42  GENDER_F                     42064 non-null  bool
 43  GENDER_M                     42064 non-null  bool
 44  last_prod_enq2_AL            42064 non-null  bool
 45  last_prod_enq2_CC            42064 non-null  bool
 46  last_prod_enq2_ConsumerLoan  42064 non-null  bool
 47  last_prod_enq2_HL            42064 non-null  bool
 48  last_prod_enq2_PL            42064 non-null  bool
 49  last_prod_enq2_others        42064 non-null  bool
 50  first_prod_enq2_AL           42064 non-null  bool
 51  first_prod_enq2_CC           42064 non-null  bool
 52  first_prod_enq2_ConsumerLoan  42064 non-null  bool
 53  first_prod_enq2_HL           42064 non-null  bool
 54  first_prod_enq2_PL           42064 non-null  bool
 55  first_prod_enq2_others       42064 non-null  bool
dtypes: bool(16), float64(5), int32(1), int64(33), object(1)
memory usage: 13.3+ MB
```

0.3.2 Model Fitting

0.3.3 We shall consider, 3 classification algorithms, namely :-

Decision Tree

[30]:
```
# Train-Test Split
from sklearn.tree import DecisionTreeClassifier
y = df_encoded['Approved_Flag']
x = df_encoded. drop ( ['Approved_Flag'], axis = 1 )
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
↪random_state=42)
```

[31]:
```
dt_model = DecisionTreeClassifier(max_depth=20, min_samples_split=10)
dt_model.fit(x_train, y_train)
y_pred = dt_model.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)
print ()
print(f"Accuracy: {accuracy:.2f}")
print ()
```

```
Accuracy: 0.71
```

[32]:
```
precision, recall, f1_score, _ = precision_recall_fscore_support(y_test, y_pred)

for i, v in enumerate(['p1', 'p2', 'p3', 'p4']):
    print(f"Class {v}:")
    print(f"Precision: {precision[i]}")
    print(f"Recall: {recall[i]}")
    print(f"F1 Score: {f1_score[i]}")
    print()
```

```
Class p1:
Precision: 0.7386934673366834
Recall: 0.7248520710059172
F1 Score: 0.7317073170731708

Class p2:
Precision: 0.8098918083462133
Recall: 0.8309217046580774
F1 Score: 0.820271989042168

Class p3:
Precision: 0.3396694214876033
Recall: 0.31018867924528304
F1 Score: 0.3242603550295859

Class p4:
```

```
Precision: 0.6366279069767442
Recall: 0.6384839650145773
F1 Score: 0.6375545851528384
```

Random Forest

[33]:
```
rf_classifier = RandomForestClassifier(n_estimators = 200, random_state=42)
rf_classifier.fit(x_train, y_train)
y_pred = rf_classifier.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print ()
print(f'Accuracy: {accuracy}')
print ()
precision, recall, f1_score, _ = precision_recall_fscore_support(y_test, y_pred)
for i, v in enumerate(['p1', 'p2', 'p3', 'p4']):
    print(f"Class {v}:")
    print(f"Precision: {precision[i]}")
    print(f"Recall: {recall[i]}")
    print(f"F1 Score: {f1_score[i]}")
    print()
```

```
Accuracy: 0.7654819921549982

Class p1:
Precision: 0.846517119244392
Recall: 0.7071005917159763
F1 Score: 0.7705534658785599

Class p2:
Precision: 0.7953968522592655
Recall: 0.931615460852329
F1 Score: 0.8581340149716997

Class p3:
Precision: 0.4423076923076923
Recall: 0.20830188679245282
F1 Score: 0.2832221652129297

Class p4:
Precision: 0.723136495643756
Recall: 0.7259475218658892
F1 Score: 0.7245392822502424
```

XGBoost

[34]:
```
from sklearn.preprocessing import LabelEncoder
xgb_classifier = xgb(objective='multi:softmax',  num_class=4)
y = df_encoded['Approved_Flag']
x = df_encoded. drop ( ['Approved_Flag'], axis = 1 )
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
x_train, x_test, y_train, y_test = train_test_split(x, y_encoded, test_size=0.
 ↪2, random_state=42)
xgb_classifier.fit(x_train, y_train)
y_pred = xgb_classifier.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print ()
print(f'Accuracy: {accuracy:.2f}')
print ()
precision, recall, f1_score, _ = precision_recall_fscore_support(y_test, y_pred)
for i, v in enumerate(['p1', 'p2', 'p3', 'p4']):
    print(f"Class {v}:")
    print(f"Precision: {precision[i]}")
    print(f"Recall: {recall[i]}")
    print(f"F1 Score: {f1_score[i]}")
    print()
```

```
Accuracy: 0.78

Class p1:
Precision: 0.8238993710691824
Recall: 0.7751479289940828
F1 Score: 0.798780487804878

Class p2:
Precision: 0.826157158234661
Recall: 0.9127849355797819
F1 Score: 0.8673133063376967

Class p3:
Precision: 0.4634433962264151
Recall: 0.29660377358490564
F1 Score: 0.3617119190059825

Class p4:
Precision: 0.7270973963355835
Recall: 0.7327502429543246
F1 Score: 0.7299128751210068
```

Here, we observe that the precision for p3 in all cases is very low. To fix this problem we shall try to do some Hyperparameter tuning.

[36]:
```
param_grid={
    'colsample_bytree':[0.1,0.3,0.5,0.7,0.9],
    'learning_rate':[0.001,0.01,0.1,1],
    'max_depth':[3,5,8,10],
    'alpha':[1,10,100],
    'n_estimators':[10,50,100]
}

index = 0

answers_grid = {
     'combination'       :[],
     'train_Accuracy'    :[],
     'test_Accuracy'     :[],
     'colsample_bytree'  :[],
     'learning_rate'     :[],
     'max_depth'         :[],
     'alpha'             :[],
     'n_estimators'      :[]

}
```

[64]:
```
y = df_encoded['Approved_Flag']
x = df_encoded. drop ( ['Approved_Flag'], axis = 1 )
```

[57]:
```
rs=GridSearchCV(xgb_classifier,param_grid=param_grid,cv=5,n_jobs=-1,verbose=4)
```

[58]:
```
model=rs.fit(x,y)
```

```
Fitting 5 folds for each of 720 candidates, totalling 3600 fits
```

[43]:
```
from sklearn import preprocessing
enc=preprocessing.LabelEncoder()
```

[44]:
```
y=enc.fit_transform(y)
```

[73]:
```
model.best_score_
```

[73]: 0.7757700293892245

[76]:
```
xgb_classifier = xgb(objective='multi:softmax',
                     num_class=4,
                     alpha=10,
                     colsample_bytree=0.7,
                     learning_rate=0.1,
                     max_depth=8,
                     n_estimators=50
                    )
```

```python
y = df_encoded['Approved_Flag']
x = df_encoded. drop ( ['Approved_Flag'], axis = 1 )
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
x_train, x_test, y_train, y_test = train_test_split(x, y_encoded, test_size=0.
 ↪2, random_state=42)
xgb_classifier.fit(x_train, y_train)
y_pred = xgb_classifier.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print ()
print(f'Accuracy: {accuracy:.2f}')
print ()
precision, recall, f1_score, _ = precision_recall_fscore_support(y_test, y_pred)
for i, v in enumerate(['p1', 'p2', 'p3', 'p4']):
    print(f"Class {v}:")
    print(f"Precision: {precision[i]}")
    print(f"Recall: {recall[i]}")
    print(f"F1 Score: {f1_score[i]}")
    print()
```

```
Accuracy: 0.78

Class p1:
Precision: 0.8448660714285714
Recall: 0.7465483234714004
F1 Score: 0.7926701570680629

Class p2:
Precision: 0.8080234159779615
Recall: 0.9302279484638256
F1 Score: 0.8648300009214044

Class p3:
Precision: 0.4658753709198813
Recall: 0.2369811320754717
F1 Score: 0.3141570785392696

Class p4:
Precision: 0.7352657004830918
Recall: 0.7395529640427599
F1 Score: 0.7374031007751939
```

For P3 we shall bootstrap samples from the data as this consistent low precision from all model might be a symptom of less representation.

[65]:

```python

```

[65]:
```
Approved_Flag
P2    25452
P3     6440
P4     5264
P1     4908
Name: count, dtype: int64
```

[89]:
```python
from sklearn.multiclass import OneVsRestClassifier

ovr = OneVsRestClassifier(xgb_classifier)
y = df_encoded['Approved_Flag']
x = df_encoded. drop ( ['Approved_Flag'], axis = 1 )
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
x_train, x_test, y_train, y_test = train_test_split(x, y_encoded, test_size=0.
 ↪2, random_state=42)
ovr.fit(x_train, y_train)
y_pred=ovr.predict(x_test)
```

[91]:
```python
accuracy = accuracy_score(y_test, y_pred)
print ()
print(f'Accuracy: {accuracy:.2f}')
print ()
precision, recall, f1_score, _ = precision_recall_fscore_support(y_test, y_pred)
for i, v in enumerate(['p1', 'p2', 'p3', 'p4']):
    print(f"Class {v}:")
    print(f"Precision: {precision[i]}")
    print(f"Recall: {recall[i]}")
    print(f"F1 Score: {f1_score[i]}")
    print()
```

```
Accuracy: 0.78

Class p1:
Precision: 0.844789356984479
Recall: 0.7514792899408284
F1 Score: 0.7954070981210856

Class p2:
Precision: 0.8059548254620124
Recall: 0.933597621407334
F1 Score: 0.8650932133345579

Class p3:
Precision: 0.47572815533980584
Recall: 0.2218867924528302
F1 Score: 0.30262480699948535
```

```
Class p4:
Precision: 0.7330791229742613
Recall: 0.7473275024295433
F1 Score: 0.7401347449470644
```

0.4 Conclusion

We are getting an average accuracy of 78% on test data which seem pretty good but, the classification accuracy for p3 is poor. To tackle this issue, we performed hyperparameter tuning and even bootstrapped data to balanced the imbalanced situation but the problem prevails. In the upcoming semester we might continue this project with the help of clustering algorithms and update the file.

[ ]:

```

```