Final_CC_Default

February 21, 2021

1 Prediction of Credit Card Default for Taiwanese Customers

2 Contents

- Introduction
- Importing Packages

- Uploading Data
- Creating Train, Validation, and Testing Sets

- Data Cleaning

- Exploratory Data Analysis
- Feature Engineering

3 Introduction

This dataset contains information on default payments, demographic factors, credit data, history of payment, and bill statements of credit card customers in Taiwan from April 2005 to September 2005.

There are 25 variables:

- ID: ID of each client
- LIMIT_BAL: Amount of given credit in NT dollars (includes individual and family/supplementary credit
- SEX: Gender
  - 1=male
  - 2=female
- EDUCATION:
  - 1=graduate school
  - 2=university
  - 3=high school
  - 4=others
  - 5=unknown
  - 6=unknown
- MARRIAGE: Marital status
  - 1=married

  - 2=single
  - 3=others)
- AGE: Age in years
- PAY_0: Repayment status in September, 2005
  - -1=pay duly
  - 1=payment delay for one month
  - 2=payment delay for two months ….
  - 8=payment delay for eight months
  - 9=payment delay for nine months and above
- PAY_2: Repayment status in August, 2005 (scale same as above)
- PAY_3: Repayment status in July, 2005 (scale same as above)
- PAY_4: Repayment status in June, 2005 (scale same as above)
- PAY_5: Repayment status in May, 2005 (scale same as above)
- PAY_6: Repayment status in April, 2005 (scale same as above)
- BILL_AMT1: Amount of bill statement in September,2005 (NT dollar)
- BILL_AMT2: Amount of bill statement in August, 2005 (NT dollar)
- BILL_AMT3: Amount of bill statement in July, 2005 (NT dollar)
- BILL_AMT4: Amount of bill statement in June, 2005 (NT dollar)
- BILL_AMT5: Amount of bill statement in May, 2005 (NT dollar)
- BILL_AMT6: Amount of bill statement in April, 2005 (NT dollar)
- PAY_AMT1: Amount of previous payment in September, 2005 (NT dollar)
- PAY_AMT2: Amount of previous payment in August, 2005 (NT dollar)
- PAY_AMT3: Amount of previous payment in July, 2005 (NT dollar)
- PAY_AMT4: Amount of previous payment in June, 2005 (NT dollar)
- PAY_AMT5: Amount of previous payment in May, 2005 (NT dollar)
- PAY_AMT6: Amount of previous payment in April, 2005 (NT dollar)
- default.payment.next.month: Default payment
  - 1=yes
  - 0=no

4 Importing Packages

[2]:
```
# Importing Packages
import numpy as np
import pandas as pd
import re
import json
import requests
import warnings
warnings.filterwarnings('ignore')
pd.set_option("display.max_rows", 999)
pd.set_option("display.max_columns", 999)
from collections import Counter
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns
%matplotlib inline
```

```
plt.style.use("fivethirtyeight")
import pickle

from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier,␣
↪RandomForestClassifier, BaggingClassifier
from sklearn.model_selection import train_test_split, KFold, cross_val_score,␣
↪cross_val_predict, GridSearchCV
from sklearn.feature_selection import RFECV
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score,␣
↪recall_score, f1_score, roc_auc_score,␣
↪classification_report,balanced_accuracy_score
from sklearn.preprocessing import StandardScaler
import xgboost as xgb
from xgboost.sklearn import XGBClassifier

%reload_ext autoreload
%autoreload 2
from utils import *
```

[3]:
```
df = pd.read_excel("data/default of credit card clients.xls")
new_header = df.iloc[0]
df = df[1:]
df.columns = new_header
df = df.rename(columns={"default payment next month": "default"})
```

5 Create Dataset Splits

[4]:
```
X = df.drop(["default"], axis=1)
y = df["default"]
X_train, X_val, y_train, y_val = train_test_split(X, y, train_size=0.8,␣
↪random_state=42)
X_tr, X_tt, y_tr, y_tt = train_test_split(X_train, y_train, train_size=0.875,␣
↪random_state=42)
train = pd.concat([X_tr, y_tr], axis=1)
val = pd.concat([X_val, y_val], axis=1)
tr = train.drop(["ID"], axis=1)
val = val.drop(["ID"], axis=1)
```

6 Data Cleaning

- The data cleaning process involved cooverting Taiwanese dollars to American dollars to facilitate understanding of the numbers involved and then converting them into integers.
- There were anomalous values in the marriage category, so observations with the value of 0 were converted into 3, which represented other.
- There were anomalous values in the education category, so observations with the value of 0, 5, or 6 were all lumped into the 4 or other category.
- In the behind1 - behind6 categories, I was originally going to convert all the observations with the negative values into 0, but since there were so many observations with -1 and -2, it couldn't have been an anomaly or mistake.

[5]:
```
url = 'https://openexchangerates.org/api/latest.json?
 ↪app_id=c51b1508fb4145259b1c2fade72a2c04'
response = requests.get(url)
data = response.json()
rate = data['rates']['TWD']
```

[6]:
```
data = [tr, val]
for d in data:
    d.rename(columns={"PAY_0": "behind1", "PAY_2": "behind2", "PAY_3":␣
 ↪"behind3", "PAY_4": "behind4", "PAY_5": "behind5", "PAY_6": "behind6",␣
 ↪"BILL_AMT1": "billed1", "BILL_AMT2": "billed2", "BILL_AMT3": "billed3",␣
 ↪"BILL_AMT4": "billed4", "BILL_AMT5": "billed5", "BILL_AMT6": "billed6",␣
 ↪"PAY_AMT1": "paid1", "PAY_AMT2": "paid2", "PAY_AMT3": "paid3", "PAY_AMT4":␣
 ↪"paid4", "PAY_AMT5": "paid5", "PAY_AMT6": "paid6", "SEX": "gender",␣
 ↪"EDUCATION": "education", "MARRIAGE": "marriage", "AGE": "age", "LIMIT_BAL":␣
 ↪"limit"}, inplace=True)
    d[['limit']] = d[['limit']]/rate
    d[['billed1', 'billed2', 'billed3', 'billed4', 'billed5', 'billed6']] =␣
 ↪d[['billed1', 'billed2', 'billed3', 'billed4', 'billed5', 'billed6']].
 ↪divide(rate, axis=1).astype(int)
    d[['paid1', 'paid2', 'paid3', 'paid4', 'paid5', 'paid6']] = d[['paid1',␣
 ↪'paid2', 'paid3', 'paid4', 'paid5', 'paid6']].divide(rate, axis=1).
 ↪astype(int)
    d['limit'] = d['limit'].apply(lambda x: round(x, 2))
    d.replace({'marriage': {0:3}}, inplace=True)
    d.replace({'education': {5:4, 0:4, 6:4}}, inplace=True)
```

[7]:
```
tr.head()
```

[7]:

	limit	gender	education	marriage	age	behind1	behind2	behind3	\
6191	1788.52	2	2	1	44	0	0	0	
16054	5723.26	2	3	1	46	-1	-1	-1	
19706	3577.04	2	2	1	47	-1	-1	-1	
23128	6080.96	2	2	1	29	0	0	0	
28516	5365.56	2	1	2	33	-2	-2	-2	

```
      behind4  behind5  behind6  billed1  billed2  billed3  billed4  billed5  \
6191        0        0        0     1630     1498     1277      799      846
16054       0       -1       -1      890       83      173      147      142
19706      -1       -1       -2      238      238        0      224      -14
23128       0        0        0     2829     2238     2264     2285     1556
28516      -2       -2       -2      872      960     1169     1196      994

       billed6  paid1  paid2  paid3  paid4  paid5  paid6  default
6191       980    107    178    107    107    178     33        0
16054       30     83    173     35    142     30    941        0
19706      -14    238      0    224      0      0      0        1
23128     1573     79     89     92     60     67     75        0
28516       80    966   1170   1197    994     80   6061        0
```

[8]:

```python
tr.describe()
```

[8]:

```
              limit     education      marriage       billed1      billed2  \
count  21000.000000  21000.000000  21000.000000  21000.000000  21000.00000
mean    5981.340686      1.842810      1.555333   1827.979905   1758.19519
std     4634.450783      0.746378      0.522538   2628.013848   2545.94903
min      357.700000      1.000000      1.000000  -5922.000000  -2495.00000
25%     1788.520000      1.000000      1.000000    127.000000    108.00000
50%     5007.850000      2.000000      2.000000    803.000000    766.00000
75%     8584.890000      2.000000      2.000000   2387.250000   2263.50000
max    35770.370000      4.000000      3.000000  34500.000000  35195.00000

            billed3       billed4       billed5       billed6         paid1  \
count  21000.000000  21000.000000  21000.000000  21000.000000  21000.000000
mean    1679.613952   1543.949857   1443.855667   1393.781810    204.618429
std     2489.800493   2299.447009   2182.100061   2134.592138    626.349241
min    -5625.000000  -6080.000000  -2909.000000  -7477.000000      0.000000
25%       98.000000     84.000000     63.000000     46.000000     35.000000
50%      717.500000    680.500000    647.000000    612.000000     75.000000
75%     2135.000000   1943.250000   1798.000000   1767.000000    179.000000
max    59525.000000  31892.000000  33165.000000  34399.000000  31247.000000

              paid2         paid3         paid4         paid5         paid6
count  21000.000000  21000.000000  21000.000000  21000.000000  21000.000000
mean     214.418048    188.432952    175.843714    172.090667    184.061095
std      897.321096    667.001026    601.041739    558.702990    631.634456
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%       29.000000     13.000000     10.000000      8.000000      4.000000
50%       71.000000     64.000000     53.000000     53.000000     53.000000
75%      178.000000    160.000000    143.000000    144.000000    143.000000
max    60246.000000  32051.000000  22213.000000  14951.000000  18856.000000
```

** Observations: **

- No missing data
- There were anomalous values for education and marriage, and the anomalous values were reassigned under other.
- Did not reassign -2 and -1 to 0 for 'behind' features despite being anomalous because they were so many -2 and -1. There must be so significance to those values.

7 Exploratory Data Anlaysis

[9]:
```
# organize features into categorical and continuous
categorical = tr[['gender', 'marriage', 'education', 'behind1', 'behind2', 
 ↪'behind3', 'behind4', 'behind5', 'behind6']]
continuous = tr[['limit', 'age', 'billed1', 'billed2', 'billed3', 'billed4', 
 ↪'billed5', 'billed6', 'paid1', 'paid2', 'paid3', 'paid4', 'paid5', 'paid6']]
cat_col = categorical.columns
cont_col = continuous.columns
```

[10]:
```
# display distributions of all the continuous variables

# con_1 = pd.melt(tr, value_vars = cont_col)
# sns.set_theme(style="darkgrid", font='serif', context='talk')
# g = sns.FacetGrid(con_1, col='variable', col_wrap=3, sharex=False, 
 ↪sharey=False, height=4)
# g = g.map(sns.distplot, 'value', color='r')
# g.set_xticklabels(rotation=45)
# g.fig.subplots_adjust(top=0.9)
# g.fig.suptitle("Distributions of Continuous Features")
# g.fig.tight_layout()
# plt.savefig("../images/distplot.png")
```

** Observations: **

- It is hard to observe any trends with the paid features.

[11]:
```
# Use bar graphs of the distribution of data for categorical variables

# cat_1 = pd.melt(tr, value_vars=cat_col)
# sns.set_theme(style="darkgrid", font='serif', context='talk')
# g = sns.FacetGrid(cat_1, col='variable', col_wrap=3, sharex=False, 
 ↪sharey=False, height=4)
# g = g.map(sns.countplot, 'value', color='dodgerblue')
# g.set_xticklabels()
# g.fig.subplots_adjust(top=0.9)
# g.fig.suptitle("Distributions of Categorical Features")
# g.fig.tight_layout()
# plt.savefig("../images/countplot.png")
```

[12]:
```
yes = tr.default.sum()
no = len(tr)-yes
perc_y = round(yes/len(tr)*100, 1)
perc_n = round(no/len(tr)*100, 1)

# plt.figure(figsize=(8,6))
# sns.set_theme(style="darkgrid", font='serif', context='talk')
# sns.countplot('default', data=tr)
# plt.title('Credit Card Baseline Default', size=16)
# plt.box(False);
# plt.savefig("../images/baseline.png")
```

- There is class inbalance in the dataset. Our baseline indicates

[14]:
```
print("Number of Total Non-Defaulters: ", yes)
print("Number of Defaulters: ", no)
print("Percentage of Non-Defaulters: ", perc_y)
print("Percentage of Defaulters: ", perc_n)

pd.DataFrame
default = pd.DataFrame(data = {"Training Dataset": [yes, no, perc_y, perc_n]},
                       index = ["Number of Total Non-Defaulters: ", "Number of␣
 ↪Defaulters: ", "Percentage of Non-Defaulters: ", "Percentage of Defaulters:␣
 ↪"])
default
```

```
Number of Total Non-Defaulters:  4656
Number of Defaulters:  16344
Percentage of Non-Defaulters:  22.2
Percentage of Defaulters:  77.8
```

[14]:

	Training Dataset
Number of Total Non-Defaulters:	4656.0
Number of Defaulters:	16344.0
Percentage of Non-Defaulters:	22.2
Percentage of Defaulters:	77.8

[15]:
```
# subset = tr[['gender', 'education', 'marriage', 'behind1', 'behind2',␣
 ↪'behind3', 'behind4', 'behind5', 'behind6', 'default']]
# f, axes = plt.subplots(3, 3, figsize=(15, 12), facecolor='white')
# sns.set_theme(style="darkgrid", font='serif', context='paper')
# f.suptitle('Frequency of Categorical Variables', size=16)
# ax1 = sns.countplot(x="gender", hue="default", data=subset, ax=axes[0,0])
# ax2 = sns.countplot(x="education", hue="default", data=subset, ax=axes[0,1])
# ax3 = sns.countplot(x="marriage", hue="default", data=subset, ax=axes[0,2])
# ax4 = sns.countplot(x="behind1", hue="default", data=subset, ax=axes[1,0])
# ax5 = sns.countplot(x="behind2", hue="default", data=subset, ax=axes[1,1])
# ax6 = sns.countplot(x="behind3", hue="default", data=subset, ax=axes[1,2])
```

```python
# ax7 = sns.countplot(x="behind4", hue="default", data=subset, ax=axes[2,0])
# ax8 = sns.countplot(x="behind5", hue="default", data=subset, ax=axes[2,1])
# ax9 = sns.countplot(x="behind6", hue="default", data=subset, ax=axes[2,2])
# plt.savefig("../images/default_freq_by_cat.png")
```

** Observations: **

- `gender`, `education`, and `marriage` doesn't seem to change with each group in terms of proportions. Behind seems to have some correlation with default. That would make sense since being behind in payments would make it more likely that you would default next month.
- There isn't a very clear distinction between the distribution of `default` on any of the demographic data. However, you do see quite some distribution differences of target classes for monthly repayment status (`behind1-behind6`).

[16]:
```python
education = tr.groupby(['education', 'default']).size().unstack(1)
education
# education.plot(kind="bar", stacked=True)
# plt.title("Distribution Count of Educational Level and Default Status",␣
 ↪size=14)
# plt.savefig("../data/stacked_bar2.png")
```

[16]:
```
default       0     1
education
1          6013  1424
2          7408  2341
3          2626   866
4           297    25
```

** No clear relationship with education. The proportion doesn't seem to change with each group. **

[17]:
```python
marriage = tr.groupby(['marriage', 'default']).size().unstack(1)
marriage
# marriage.plot(kind="bar", stacked=True)
# plt.title("Distribution of Default Status for Marital Status", size=14)
# plt.savefig("../images/stacked_bar3.png")
```

[17]:
```
default      0     1
marriage
1         7354  2258
2         8778  2336
3          212    62
```

8 Vanilla Model

[ ]:
```
logreg = LogisticRegression(solver="liblinear", random_state=42).fit(X_tr_dum,␣
↪y_tr)
y_pred_log_tr = logreg.predict(X_tr_dum)
y_pred_log_val = logreg.predict(X_val_dum)

rfc = RandomForestClassifier().fit(X_tr, y_tr)
y_pred_rfc_tr = rfc.predict(X_tr)
y_pred_rfc_val = rfc.predict(X_val)

dtc = DecisionTreeClassifier().fit(X_tr, y_tr)
y_pred_dtc_tr = dtc.predict(X_tr)
y_pred_dtc_val = dtc.predict(X_val)

knn = KNeighborsClassifier().fit(X_tr, y_tr)
y_pred_knn_tr = knn.predict(X_tr)
y_pred_knn_val = knn.predict(X_val)

gnb = GaussianNB().fit(X_tr, y_tr)
y_pred_gnb_tr = gnb.predict(X_tr)
y_pred_gnb_val = gnb.predict(X_val)

lda = LinearDiscriminantAnalysis().fit(X_tr, y_tr)
y_pred_lda_tr = lda.predict(X_tr)
y_pred_lda_val = lda.predict(X_val)

abc = AdaBoostClassifier().fit(X_tr, y_tr)
y_pred_abc_tr = abc.predict(X_tr)
y_pred_abc_val = abc.predict(X_val)

gbc = GradientBoostingClassifier().fit(X_tr, y_tr)
y_pred_gbc_tr = gbc.predict(X_tr)
y_pred_gbc_val = gbc.predict(X_val)

xgb = XGBClassifier().fit(X_tr, y_tr)
y_pred_xgb_tr = xgb.predict(X_tr)
y_pred_xgb_val = xgb.predict(X_val)
```

[18]:
```
baseline = pd.read_csv("data/charts/baseline.csv")
baseline
```

[18]:
```
                 Unnamed: 0  Accuracy  F1 Score    Recall  Precision  \
0       Logistic Regression  0.811500  0.360656  0.242955   0.699561
1  Random Forest Classifier  0.814833  0.461464  0.362529   0.634667
2  Decision Tree Classifier  0.730667  0.399257  0.408987   0.389978
3       K-Nearest Neighbors  0.798000  0.447080  0.373191   0.557452
```

```
4          Gaussian Naive Bayes  0.724000  0.498486  0.626809   0.413776
5  Linear Discriminant Analysis  0.810333  0.367778  0.252094   0.679671
6           AdaBoost Classifier  0.815667  0.425753  0.312262   0.668842
7  Gradient Boosting Classifier  0.820833  0.468085  0.360244   0.668079
8            XGBoost Classifier  0.816833  0.469338  0.370145   0.641161

     PR AUC
0  0.486829
1  0.513445
2  0.289090
3  0.416605
4  0.480981
5  0.480476
6  0.523430
7  0.545925
8  0.518488
```

9 Feature Engineering

- `age_bin`: 1 = young adult, 2 = middle age, 3 = senior
- `gen-mar`: interaction between gender and marriage status
- `gen-age`: interaction between age and gender
- 'avail…': fraction of estimated available balance based on what is billed per month

[19]:
```
data = [tr, val]

# create features for demographic variables
for d in data:
    d['age_bin'] = 0
    d.loc[((d['age'] > 20) & (d['age'] < 30)) , 'age_bin'] = 1
    d.loc[((d['age'] >= 30) & (d['age'] < 60)) , 'age_bin'] = 2
    d.loc[((d['age'] >= 60) & (d['age'] < 81)) , 'age_bin'] = 3
    # create categories for single, married, divorced males and females
    d['gen-mar'] = d['gender'] + d['marriage']
    # create categories for young, middle age and senior males and females
    d['gen-age'] = d['gender'] + d['age_bin']

# feature for credit use percentage: fraction of estimated available balance␣
 ↪based on what is billed per month
# (credit limit - monthly billed amount) / credit limit
for d in data:
    d['avail6'] = (d.limit - d.billed6) / d.limit
    d['avail5'] = (d.limit - d.billed5) / d.limit
    d['avail4'] = (d.limit - d.billed4) / d.limit
    d['avail3'] = (d.limit - d.billed3) / d.limit
    d['avail2'] = (d.limit - d.billed2) / d.limit
```

```python
    d['avail1'] = (d.limit - d.billed1) / d.limit
    d['avg_av'] = (d.avail1 + d.avail2 + d.avail3 + d.avail4 + d.avail5 + d.avail6) / 6

# create a feature that indicates whether a client has had a delayed payment or not
def delayed_payment(d):
    if (d.behind1 > 0) or (d.behind2 > 0) or (d.behind3 > 0) or (d.behind4 > 0) or (d.behind5 > 0) or (d.behind6 > 0):
        return 1
    else:
        return 0
for d in data:
    d['delayed'] = d.apply(delayed_payment, axis=1)

# create feature for the total number of months with delayed payment status for a particular client
def total_months_with_delayed_payments(d):
    count = 0
    if (d.behind1 > 0):
        count += 1
    if (d.behind2 > 0):
        count += 1
    if (d.behind3 > 0):
        count += 1
    if (d.behind4 > 0):
        count += 1
    if (d.behind5 > 0):
        count += 1
    if (d.behind6 > 0):
        count += 1
    return count
for d in data:
    d['latemths'] = d.apply(total_months_with_delayed_payments, axis=1)

# the ratio of amount paid and amount billed
for d in data:
    d['pperb1'] = d.paid1 / d.billed2
    d['pperb2'] = d.paid2 / d.billed3
    d['pperb3'] = d.paid3 / d.billed4
    d['pperb4'] = d.paid4 / d.billed5
    d['pperb5'] = d.paid5 / d.billed6

# remove any infinity and NaN values
datasets = ['pperb1', 'pperb2', 'pperb3', 'pperb4', 'pperb5']
for data in datasets:
    tr.replace({data: {np.inf: 0, np.nan: 0}}, inplace=True)
```

```
    val.replace({data: {np.inf: 0, np.nan: 0}}, inplace=True)
```

[20]:
```
# plt.style.use("fivethirtyeight")
# sns.set_theme(style="darkgrid", font='serif', context='paper')
# plt.figure(figsize = (20,16))
# plt.title('Pearson Correlation of Features', y = 1.05, size = 20)
# g = sns.heatmap(tr.corr(), cmap='RdBu', square=True, linecolor='white',␣
↪linewidths=0.2)
# plt.savefig("../images/correlation_matrix_2.png")
```

** This includes my engineered features. Default seems to be correlated with two of my engineered features, delayed and latemnths. Delayed is whether you have had a delayed payment durig the 6 month history or not. latemnths is the total nunber of months you were given a status of behind in payments. Seems to be correlated with behind1 and limit.**

[22]:
```
pickle_in = open("data/pickles/training_features.pickle","rb")
train2 = pickle.load(pickle_in)
pickle_in = open("data/pickles/validate_features.pickle","rb")
validate2 = pickle.load(pickle_in)
```

[23]:
```
X_train2 = train2.drop(["default"], axis=1)
y_tr = train2["default"]
X_validate2 = validate2.drop(["default"], axis=1)
y_val = validate2["default"]
```

[24]:
```
# # Grab indices of columns for creating dummy variables and create dataframe␣
↪with dummy variables
dum_feat = X_train2[['gender', 'education', 'marriage', 'age_bin', 'gen-mar',␣
↪'gen-age']]
dum_index = dum_feat.columns
tr_dum = pd.get_dummies(data=dum_feat, columns=dum_index, drop_first=True,␣
↪prefix=['sex', 'edu', 'mar', 'agebin', 'sexmar', 'sexage'])
cont_feat = X_train2.drop(['gender', 'education', 'marriage', 'age_bin',␣
↪'gen-mar', 'gen-age'], axis=1)
X_train2_dum = cont_feat.join(tr_dum)
X_train2_dum.head()
```

[24]:
```
     limit  age  behind1  behind2  behind3  behind4  behind5  behind6  \
0  1790.26   44        0        0        0        0        0        0
1  5728.83   46       -1       -1       -1        0       -1       -1
2  3580.52   47       -1       -1       -1       -1       -1       -2
3  6086.88   29        0        0        0        0        0        0
4  5370.78   33       -2       -2       -2       -2       -2       -2

   billed1  billed2  billed3  billed4  billed5  billed6   paid1   paid2  \
0  1631.93  1500.45  1278.35   800.60   847.12   981.81  107.99  179.13
1   891.69    83.71   173.87   147.77   143.04    30.15   83.89  173.87
```

```
2   238.68   238.68     0.00   224.50   -14.18   -14.18  238.68     0.00
3  2831.87  2240.51  2267.08  2288.06  1557.71  1575.25   80.02    89.26
4   873.40   961.26  1170.90  1198.01   995.38    80.96  966.99  1171.37

     paid3   paid4   paid5    paid6    avail6    avail5    avail4    avail3  \
0   107.42  107.42  179.03    33.08  0.451582  0.526817  0.552802  0.285942
1    35.81  143.04   30.15   942.14  0.994737  0.975032  0.974206  0.969650
2   224.50    0.00    0.00     0.00  1.003960  1.003960  0.937300  1.000000
3    92.56   60.26   68.07    75.58  0.741206  0.744087  0.624100  0.627546
4  1198.58  995.67   80.96  6067.73  0.984926  0.814668  0.776939  0.781987

     avail2    avail1    avg_av  delayed  latemths    pperb1    pperb2  \
0  0.161882  0.088440  0.344578        0         0  0.071972  0.140126
1  0.985388  0.844350  0.957227        0         0  1.002150  1.000000
2  0.933339  0.933339  0.968650        0         0  1.000000  0.000000
3  0.631912  0.534758  0.650602        0         0  0.035715  0.039372
4  0.821020  0.837379  0.836153        0         0  1.005961  1.000401

     pperb3    pperb4    pperb5  sex_2  edu_2  edu_3  edu_4  mar_2  mar_3  \
0  0.134174  0.126806  0.182347      1      1      0      0      0      0
1  0.242336  1.000000  1.000000      1      0      1      0      0      0
2  1.000000 -0.000000 -0.000000      1      1      0      0      0      0
3  0.040453  0.038685  0.043212      1      1      0      0      0      0
4  1.000476  1.000291  1.000000      1      0      0      0      1      0

   agebin_2  agebin_3  sexmar_3  sexmar_4  sexmar_5  sexage_3  sexage_4  \
0         1         0         1         0         0         0         1
1         1         0         1         0         0         0         1
2         1         0         1         0         0         0         1
3         0         0         1         0         0         1         0
4         1         0         0         1         0         0         1

   sexage_5
0         0
1         0
2         0
3         0
4         0
```

[25]:

```
dum_feat2 = X_validate2[['gender', 'education', 'marriage', 'age_bin',␣
↪'gen-mar', 'gen-age']]
dum_index2 = dum_feat2.columns
val_dum = pd.get_dummies(data=dum_feat2, columns=dum_index2, drop_first=True,␣
↪prefix=['sex', 'edu', 'mar', 'agebin', 'sexmar', 'sexage'])
cont_feat2 = X_validate2.drop(['gender', 'education', 'marriage', 'age_bin',␣
↪'gen-mar', 'gen-age'], axis=1)
X_validate2_dum = cont_feat2.join(val_dum)
```

```
X_validate2_dum.head()
```

[25]:

```
     limit  age  behind1  behind2  behind3  behind4  behind5  behind6  \
0  1074.16   25        0        0        0        0        0        0
1  5370.78   26        0        0        0        0        0        0
2  2506.36   32        0        0        0        0        0        0
3  4654.68   49        0        0        0        0        0       -1
4  1790.26   36        0        0        0        0        0        2

   billed1  billed2  billed3  billed4  billed5  billed6   paid1   paid2  \
0   317.38   360.27   414.66   450.43   491.10   530.92   53.71   71.61
1  4895.86  4498.96  4177.89  3637.13  2783.53  2766.45  160.62  151.64
2  2510.73  2473.42  2453.73  2497.52  2510.34  2513.96   87.04  111.43
3   740.38   678.72   579.04   605.04   402.31   248.63   57.65   64.74
4  3373.85  1705.58  1516.74   700.85   726.67   696.02   71.61   53.71

    paid3  paid4   paid5   paid6    avail6    avail5    avail4    avail3  \
0   53.71  53.71   53.71   71.61  0.505735  0.542806  0.580668  0.613968
1  113.18  94.78   95.56   95.56  0.484907  0.481727  0.322793  0.222107
2  107.42  87.29   89.51   91.45 -0.003032 -0.001588  0.003527  0.020999
3  251.14   0.97  251.03  157.83  0.946585  0.913569  0.870015  0.875600
4   35.81  64.45    0.00   35.81  0.611218  0.594098  0.608521  0.152782

     avail2    avail1    avg_av  delayed  latemths    pperb1    pperb2  \
0  0.664603  0.704532  0.602052        0         0  0.149083  0.172696
1  0.162327  0.088427  0.293715        0         0  0.035702  0.036296
2  0.013143 -0.001744  0.005217        0         0  0.035190  0.045412
3  0.854185  0.840939  0.883482        0         0  0.084939  0.111806
4  0.047300 -0.884559  0.188227        1         1  0.041986  0.035411

     pperb3    pperb4    pperb5  sex_2  edu_2  edu_3  edu_4  mar_2  mar_3  \
0  0.119242  0.109367  0.101164      0      1      0      0      1      0
1  0.031118  0.034050  0.034542      1      0      0      0      1      0
2  0.043011  0.034772  0.035605      1      0      1      0      0      0
3  0.415080  0.002411  1.009653      0      0      1      0      1      0
4  0.051095  0.088692  0.000000      1      1      0      0      1      0

   agebin_2  agebin_3  sexmar_3  sexmar_4  sexmar_5  sexage_3  sexage_4  \
0         0         0         1         0         0         0         0
1         0         0         0         1         0         1         0
2         1         0         1         0         0         0         1
3         1         0         1         0         0         1         0
4         1         0         0         1         0         0         1

   sexage_5
0         0
1         0
```

```
2          0
3          0
4          0
```

[22]:
```
scaler = StandardScaler().fit(X_train2_dum)
X_tr2_dum = scaler.transform(X_train2_dum)
X_val2_dum = scaler.transform(X_validate2_dum)
```

[23]:
```
scaler2 = StandardScaler().fit(X_train2)
X_tr2 = scaler2.transform(X_train2)
X_val2 = scaler2.transform(X_validate2)
```

[ ]:
```
logreg2 = LogisticRegression(solver="liblinear", random_state=42).
 ↪fit(X_tr2_dum, y_tr)
y_pred_log_tr2 = logreg2.predict(X_tr2_dum)
y_pred_log_val2 = logreg2.predict(X_val2_dum)

rfc2 = RandomForestClassifier().fit(X_tr2, y_tr)
y_pred_rfc_tr2 = rfc2.predict(X_tr2)
y_pred_rfc_val2 = rfc2.predict(X_val2)

dtc2 = DecisionTreeClassifier().fit(X_tr2, y_tr)
y_pred_dtc_tr2 = dtc2.predict(X_tr2)
y_pred_dtc_val2 = dtc2.predict(X_val2)

knn2 = KNeighborsClassifier().fit(X_tr2, y_tr)
y_pred_knn_tr2 = knn2.predict(X_tr2)
y_pred_knn_val2 = knn2.predict(X_val2)

gnb2 = GaussianNB().fit(X_tr2, y_tr)
y_pred_gnb_tr2 = gnb2.predict(X_tr2)
y_pred_gnb_val2 = gnb2.predict(X_val2)

lda2 = LinearDiscriminantAnalysis().fit(X_tr2, y_tr)
y_pred_lda_tr2 = lda2.predict(X_tr2)
y_pred_lda_val2 = lda2.predict(X_val2)

abc2 = AdaBoostClassifier().fit(X_tr2, y_tr)
y_pred_abc_tr2 = abc2.predict(X_tr2)
y_pred_abc_val2 = abc2.predict(X_val2)

gbc2 = GradientBoostingClassifier().fit(X_tr2, y_tr)
y_pred_gbc_tr2 = gbc2.predict(X_tr2)
y_pred_gbc_val2 = gbc2.predict(X_val2)

xgb2 = XGBClassifier().fit(X_tr2, y_tr)
y_pred_xgb_tr2 = xgb2.predict(X_tr2)
```

```
y_pred_xgb_val2 = xgb2.predict(X_val2)
```

[27]:
```
features_model = pd.read_csv("data/charts/features_model.csv")
features_model
```

[27]:

	Unnamed: 0	Accuracy	F1 Score	Recall	Precision	PR AUC
0	Logistic Regression 2	0.809000	0.399371	0.290175	0.640336	0.500013
1	Random Forest Classifier 2	0.816500	0.467344	0.367860	0.640584	0.517269
2	Decision Tree Classifier 2	0.725167	0.393527	0.407464	0.380512	0.284835
3	K-Nearest Neighbors 2	0.789500	0.425125	0.355674	0.528281	0.405541
4	Gaussian Naive Bayes 2	0.278167	0.374042	0.985529	0.230824	0.476403
5	Linear Discriminant Analysis 2	0.809000	0.437684	0.339680	0.615172	0.500475
6	AdaBoost Classifier 2	0.818500	0.451385	0.341203	0.666667	0.525636
7	Gradient Boosting Classifier 2	0.820500	0.463378	0.354151	0.670029	0.544927
8	XGBoost Classifier 2	0.813333	0.458937	0.361767	0.627477	0.529126

** Observations: **

In imbalanced datasets, we don't want to use accuracy as our gold standard metric since it is easy to get a high accuracy score by simply classifying all observations as the majority class.

ROC AUC score is equivalent to calculating the rank correlation between predictions and targets, i.e. how good at ranking predictions the model is. It tells us what is the probability that a positive instance randomly chosen is ranked higher than a negative instance randomly chosen. Generally, the ROC AUC is not used for imbalanced datasets because the FPR for highly imbalanced datasets is pulled down to a large number of true negatives.

With the F1 score, it combines both recall and prevision into one metric by calculating the harmonic mean. It is worth noting that the F1 score is a special case of the F-beta score, where 1 indicates that we care about recall and precision equally. For a F2 score, we care about more recall more than precision, in fact, twice as much.Both F1 score and acuracy is calculated on the predicted classes not the prediction scores. We can adjust the threshold to finetune the F1 score, but accuracy also depends on the threshold.

PR AUC score shows the tradeoff between precision and recall at every threshold, where a high score or area represents both high recall and precision. High precision relates to a low FPR, and high recall relates to a low FNR. A high score means that the classifier is returning accurate results (high precision), as well as returning a majority of all positive results (high recall).

- To be expected, Random Forest and Decision Tree is overfit with high training accuracy and much lower validation accuracy.
- Tree-based and ensemble classifiers have the most potential with the highest PR AUC scores: Random Forest, AdaBoost, Gradient Boosting, XGBoost
- Decision Tree once again has an exceptionally low PR AUC score
- Gaussian Naive Bayes has an exceptionally low accuracy score in the second model
- A change in the F1 score between first and second model is accompanied by changes in recall and precision, which is to be expected since there is a tradeoff between recall and precision.
- Gaussian Bayes Classifier metrics changes significantly between the two models
- For both models, Gradient Boosting, AdaBoost, XGBoost, Random Forest have the highest PR AUC scores

10 Feature Selection

** Observations: **

Top Features:

- Decision Tree Top 10: `behind1`, `age`, `latemnths`, `avail1`, `avail2`, `billed`, `avail5`, `limit`, `paid1`, `avg_av`….
- Random Forest Top 10: `behind1`, `age`, `latemnths`, `limit`, `avg_av`, `avail1`, `billed1`, `delayed`, `avail2`, `behind2`….
- XGBoost FI: `delayed`, `behind1`. (also `latemths`, `behind2`)
- RFECV: `limit`, `behind1`, `paid2`, `delayed`, `latemths`

Features to Remove:

- `age_bin`, `gender`, `marriage`, `gen-age`, `gen-mar`, `behind6`, `behind5`, `behind4`, `education`

11 New Baseline Model

[29]:
```
pickle_in = open("data/pickles/training_model.pickle","rb")
train3 = pickle.load(pickle_in)
pickle_in = open("data/pickles/validate_model.pickle","rb")
validate3 = pickle.load(pickle_in)
```

[30]:
```
X_train3 = train3.drop(["default"], axis=1)
y_tr = train3["default"]
X_validate3 = validate3.drop(["default"], axis=1)
y_val = validate3["default"]

scaler3 = StandardScaler().fit(X_train3)
X_tr3 = scaler3.transform(X_train3)
X_val3 = scaler3.transform(X_validate3)
```

[ ]:
```
logreg3 = LogisticRegression(solver="liblinear", random_state=42).fit(X_tr3,␣
↪y_tr)
y_pred_log_tr3 = logreg3.predict(X_tr3)
```

```
y_pred_log_val3 = logreg3.predict(X_val3)

rfc3 = RandomForestClassifier().fit(X_tr3, y_tr)
y_pred_rfc_tr3 = rfc3.predict(X_tr3)
y_pred_rfc_val3 = rfc3.predict(X_val3)

abc3 = AdaBoostClassifier().fit(X_tr3, y_tr)
y_pred_abc_tr3 = abc3.predict(X_tr3)
y_pred_abc_val3 = abc3.predict(X_val3)

gbc3 = GradientBoostingClassifier().fit(X_tr3, y_tr)
y_pred_gbc_tr3 = gbc3.predict(X_tr3)
y_pred_gbc_val3 = gbc3.predict(X_val3)

xgb3 = XGBClassifier().fit(X_tr3, y_tr)
y_pred_xgb_tr3 = xgb3.predict(X_tr3)
y_pred_xgb_val3 = xgb3.predict(X_val3)
```

[32]:
```
new_baseline = pd.read_csv("data/charts/new_baseline.csv")
new_baseline
```

[32]:
```
                           Unnamed: 0  Accuracy  F1 Score    Recall  Precision  \
0  Logistic Regression New Baseline  0.807167  0.380952  0.271135   0.640288
1        Random Forest New Baseline  0.811500  0.461685  0.369383   0.615482
2             AdaBoost New Baseline  0.820167  0.448646  0.334349   0.681677
3    Gradient Boosting New Baseline  0.820500  0.466039  0.357959   0.667614
4              XGBoost New Baseline  0.814167  0.455300  0.354912   0.634877

     PR AUC
0  0.498666
1  0.492459
2  0.531694
3  0.542846
4  0.508630
```

** Observations: **

- Brought it down to top 10 features
- Gradient Boosting has the highest PR AUC Score as well as the highest F1 Score, so we are maximizing Recall and Precision in this model

12 Hyperparameter Tuning

[ ]:
```
logreg = LogisticRegression()
params = {'C': [0.001, 0.01, 0.1, 1, 10],
          'penalty': ['none', 'l1', 'l2', 'elasticnet'],
          'solver': ['liblinear', 'newton-cg', 'lbfgs', 'sag', 'saga']}
```

```
gslog = GridSearchCV(estimator = logreg,
                     param_grid = params,
                     scoring = 'average_precision',
                     cv = 10,
                     n_jobs = -1).fit(X_tr, y_tr)
y_pred_gslog_tr = gslog.predict(X_tr)
y_pred_gslog_val = gslog.predict(X_val)


rfc = RandomForestClassifier()
params = {'n_estimators': [100, 200, 400, 600, 1000],
          'criterion': ['entropy', 'gini'],
          'max_depth': [5, 8, 15, 25, 30],
          'min_samples_split': [2, 5, 10, 15, 100],
          'min_samples_leaf': [1, 2, 5, 10]}
gsrfc = GridSearchCV(estimator = rfc,
                     param_grid = params,
                     scoring = 'average_precision',
                     cv = 5,
                     n_jobs = -1).fit(X_tr, y_tr)
y_pred_gsrfc_tr = gsrfc.predict(X_tr)
y_pred_gsrfc_val = gsrfc.predict(X_val)


rfc = RandomForestClassifier()
params = {'n_estimators': [100, 200, 400, 600, 1000],
          'criterion': ['entropy', 'gini'],
          'max_depth': [5, 8, 15, 25, 30],
          'min_samples_split': [2, 5, 10, 15, 100],
          'min_samples_leaf': [1, 2, 5, 10]}
gsrfc = GridSearchCV(estimator = rfc,
                     param_grid = params,
                     scoring = 'average_precision',
                     cv = 5,
                     n_jobs = -1).fit(X_tr, y_tr)
y_pred_gsrfc_tr = gsrfc.predict(X_tr)
y_pred_gsrfc_val = gsrfc.predict(X_val)

abc = AdaBoostClassifier()
params = {'n_estimators': [10, 50, 100, 200],
          'learning_rate': [0.001, 0.01, 0.1, 0.2, 0.5]}
gsabc = GridSearchCV(estimator = abc,
                     param_grid = params,
                     n_jobs = -1,
                     cv = 5,
                     scoring = 'average_precision').fit(X_tr, y_tr)
y_pred_gsabc_tr = gsabc.predict(X_tr)
```

```python
y_pred_gsabc_val = gsabc.predict(X_val)

gbc = GradientBoostingClassifier()
params = {'n_estimators': [10, 100, 1000],
          'learning_rate': [0.001, 0.01, 0.1],
          'max_depth': [3, 7, 9]}
gsgbc = GridSearchCV(estimator = gbc,
                     param_grid = params,
                     n_jobs = -1,
                     cv = 5,
                     scoring = 'average_precision').fit(X_tr, y_tr)
y_pred_gsgbc_tr = gsgbc.predict(X_tr)
y_pred_gsgbc_val = gsgbc.predict(X_val)
```

** Best Hyperparameters for each model: **

- Logistic Regression: # Best: 0.522622 using {'C': 1, 'penalty': 'l2', 'solver': 'newton-cg'}
- Random Forest: # Best: 0.565196 using {'criterion': 'gini', 'max_depth': 8, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 400}
- AdaBoost: # Best: 0.545818 using {'learning_rate': 0.1, 'n_estimators': 200}
- Gradient Boosting: # Best: 0.558390 using {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 1000}
- XGBoost: # Best: 0.555500 using {'max_depth': 3, 'min_child_weight': 5, 'n_estimators': 50}

[35]:
```python
scores3 = pd.read_csv("data/charts/scores3.csv")
scores3
```

[35]:
```
                          Unnamed: 0  Accuracy  F1 Score    Recall  \
0  Logistic Regression with GridSearchCV  0.807167  0.380952  0.271135
1        Random Forest with GridSearchCV  0.820667  0.464143  0.354912
2             AdaBoost with GridSearchCV  0.820833  0.443870  0.326733
3    Gradient Boosting with GridSearchCV  0.819667  0.462227  0.354151
4              XGBoost with GridSearchCV  0.818500  0.457939  0.350343

   Precision    PR AUC
0   0.640288  0.498658
1   0.670504  0.545164
2   0.691935  0.524433
3   0.665236  0.543499
4   0.660920  0.539395
```

** Observations: **

- GridSearchCV was run with the scoring parameter set to find the highest average precision score, which is the PR AUC score.
- Random Forest after hyperparameter tuning has the best accuracy as well as the highest PR AUC score.

- We pickle out Random Forest with GridSearchCV tuned parameters as our best model for now before completing class imbalance methods.

13 Class Imbalance Methods

(Will update soon....)

- Initial findings show that there is dramatically improved PR AUC scores without sacrificing any accuracy.
- Accuracy has not improved much, but has not decreased.

14 Pickle Out Best Model

[ ]:
```
pickle_out = open("../data/best_model.pickle","wb")
pickle.dump(rfcb, pickle_out)
pickle_out.close()
```

15 Holdout Set Prediction

[ ]:
```
test = pd.read_csv('../data/testing.csv')
tt = test.drop(["ID"], axis=1)
```

[ ]:
```
url = 'https://openexchangerates.org/api/latest.json?
 ↪app_id=c51b1508fb4145259b1c2fade72a2c04'
response = requests.get(url)
data = response.json()
rate = data['rates']['TWD']
```

[ ]:
```
data = [tt]
for d in data:
    d.rename(columns={"PAY_0": "behind1",
                        "PAY_2": "behind2",
                        "PAY_3": "behind3",
                        "PAY_4": "behind4",
                        "PAY_5": "behind5",
                        "PAY_6": "behind6",
                        "BILL_AMT1": "billed1",
                        "BILL_AMT2": "billed2",
                        "BILL_AMT3": "billed3",
                        "BILL_AMT4": "billed4",
                        "BILL_AMT5": "billed5",
                        "BILL_AMT6": "billed6",
                        "PAY_AMT1": "paid1",
                        "PAY_AMT2": "paid2",
                        "PAY_AMT3": "paid3",
                        "PAY_AMT4": "paid4",
```

```
                              "PAY_AMT5": "paid5",
                              "PAY_AMT6": "paid6",
                              "SEX": "gender",
                              "EDUCATION": "education",
                              "MARRIAGE": "marriage",
                              "AGE": "age",
                              "LIMIT_BAL": "limit"}, inplace=True)
    d[['limit']] = d[['limit']]/rate
    d[['billed1', 'billed2', 'billed3', 'billed4', 'billed5', 'billed6']] =␣
 ↪d[['billed1', 'billed2', 'billed3', 'billed4', 'billed5', 'billed6']].
 ↪divide(rate, axis=1)
    d[['paid1', 'paid2', 'paid3', 'paid4', 'paid5', 'paid6']] = d[['paid1',␣
 ↪'paid2', 'paid3', 'paid4', 'paid5', 'paid6']].divide(rate, axis=1)
    d['limit'] = d['limit'].apply(lambda x: round(x, 2))
    d[['billed1', 'billed2', 'billed3', 'billed4', 'billed5', 'billed6']] =␣
 ↪d[['billed1', 'billed2', 'billed3', 'billed4', 'billed5', 'billed6']].
 ↪apply(lambda x: round(x, 2))
    d[['paid1', 'paid2', 'paid3', 'paid4', 'paid5', 'paid6']] = d[['paid1',␣
 ↪'paid2', 'paid3', 'paid4', 'paid5', 'paid6']].apply(lambda x: round(x, 2))
    d.replace({'marriage': {0:3}}, inplace=True)
    d.replace({'education': {5:4, 0:4, 6:4}}, inplace=True)

tt = tt.drop(["Unnamed: 0"], axis=1)
```

[ ]:
```
for d in data:
    d['avail6'] = (d.limit - d.billed6) / d.limit
    d['avail5'] = (d.limit - d.billed5) / d.limit
    d['avail4'] = (d.limit - d.billed4) / d.limit
    d['avail3'] = (d.limit - d.billed3) / d.limit
    d['avail2'] = (d.limit - d.billed2) / d.limit
    d['avail1'] = (d.limit - d.billed1) / d.limit
    d['avg_av'] = (d.avail1 + d.avail2 + d.avail3 + d.avail4 + d.avail5 + d.
 ↪avail6) / 6

def delayed_payment(d):
    if (d.behind1 > 0) or (d.behind2 > 0) or (d.behind3 > 0) or (d.behind4 > 0)␣
 ↪or (d.behind5 > 0) or (d.behind6 > 0):
        return 1
    else:
        return 0
for d in data:
    d['delayed'] = d.apply(delayed_payment, axis=1)

def total_months_with_delayed_payments(d):
    count = 0
    if (d.behind1 > 0):
```

```
        count += 1
    if (d.behind2 > 0):
        count += 1
    if (d.behind3 > 0):
        count += 1
    if (d.behind4 > 0):
        count += 1
    if (d.behind5 > 0):
        count += 1
    if (d.behind6 > 0):
        count += 1
    return count
for d in data:
    d['latemths'] = d.apply(total_months_with_delayed_payments, axis=1)
```

[ ]:
```
X_tt = tt[['limit', 'behind1', 'paid2', 'delayed', 'latemths', 'age',␣
 ↪'behind2', 'billed1', 'avg_av', 'avail1']]
```

[ ]:
```
pickle_in = open("../data/best_model.pickle","rb")
model = pickle.load(pickle_in)
```

[ ]:
```
y_pred_tt = model.predict(X_tt)
```

[ ]:
```
pickle_out = open("../data/final_prediction.pickle","wb")
pickle.dump(y_pred_tt, pickle_out)
pickle_out.close()
```

Analysis:

There was not a significant difference in the vanilla model, model with all the engineered features, and model after using feature selection methods. The initial models were selected for the highest accuracy and PR AUC score.

Some of the engineered features created seemed to have a stronger correlation than the original variables. I have to check for collinearity as some of the variables would overlap in context. I am surprised that the demographic features does not have a greater correlation with default. It would seem useful for companies to be able to identify certain demographic groups that are more prone to defaulting.

The metric I used was the PR AUC score, but with an eye to increasing accuracy and PR AUC score, which is the scoring parameters I used in GridSearchCV for hyperparameter tuning. Hyperparameter tuning improved accuracy to 82% from a baseline of 77%, and the highest PR AUC score at around 54%. My initial analysis of implementing class imbalance methods is that it substantially increases the PR AUC score to almost 90%, but accuracy tops out at 82% on the validation set.