Modeling_2_Notebook

February 21, 2021

Hyperparameter Tuning and Class Imbalance Notebook

1 Contents

2 Importing Packages

[128]:
```
import numpy as np
import pandas as pd
import re
from matplotlib import pyplot as plt
from matplotlib import style
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
import itertools
from collections import Counter


from sklearn import linear_model
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier,␣
↪RandomForestClassifier, BaggingClassifier, HistGradientBoostingClassifier
from sklearn.model_selection import train_test_split, KFold, cross_val_score,␣
↪cross_val_predict, GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score,␣
↪recall_score, f1_score, precision_recall_curve, roc_curve, roc_auc_score,␣
↪classification_report, plot_confusion_matrix, auc, mean_squared_error,␣
↪confusion_matrix, balanced_accuracy_score
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import make_pipeline
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer, make_column_selector as selector
from imblearn.under_sampling import CondensedNearestNeighbour
from imblearn.ensemble import BalancedBaggingClassifier,␣
↪BalancedRandomForestClassifier, EasyEnsembleClassifier, RUSBoostClassifier
from imblearn.metrics import geometric_mean_score
from imblearn.under_sampling import TomekLinks

import xgboost as xgb
from xgboost.sklearn import XGBClassifier


%reload_ext autoreload
%autoreload 2

from utils import *

plt.style.use("fivethirtyeight")
sns.set_theme(style="darkgrid", font='serif', context='poster')

import pickle

from imblearn.under_sampling import NeighbourhoodCleaningRule
from matplotlib import pyplot
from numpy import where
from imblearn.under_sampling import NearMiss
from imblearn.under_sampling import OneSidedSelection
from numpy import mean
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.tree import DecisionTreeClassifier
from imblearn.pipeline import Pipeline
```

```python
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
from sklearn.dummy import DummyClassifier
from sklearn.utils import resample
from sklearn.model_selection import cross_validate
```

3 Preparing Data for Modeling

[59]:
```python
pickle_in = open("../data/pickles/training_model.pickle","rb")
train = pickle.load(pickle_in)
pickle_in = open("../data/pickles/validate_model.pickle","rb")
validate = pickle.load(pickle_in)
```

[60]:
```python
train.head()
```

[60]:
```
     limit  behind1    paid2  delayed  latemths  age  behind2  billed1  \
0  1790.26        0   179.13        0         0   44        0  1631.93
1  5728.83       -1   173.87        0         0   46       -1   891.69
2  3580.52       -1     0.00        0         0   47       -1   238.68
3  6086.88        0    89.26        0         0   29        0  2831.87
4  5370.78       -2  1171.37        0         0   33       -2   873.40

     avg_av    avail1  default
0  0.344578  0.088440        0
1  0.957227  0.844350        0
2  0.968650  0.933339        1
3  0.650602  0.534758        0
4  0.836153  0.837379        0
```

[61]:
```python
X_train = train.drop(["default"], axis=1)
y_tr = train["default"]
X_validate = validate.drop(["default"], axis=1)
y_val = validate["default"]
```

[62]:
```python
scaler = StandardScaler()
scaler.fit(X_train)
X_tr = scaler.transform(X_train)
X_val = scaler.transform(X_validate)
```

4 Hyperparameter Tuning

4.1 Logistic Regression with GridSearchCV

[36]:
```
# 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)
# print("Best: %f using %s" % (gslog.best_score_, gslog.best_params_))
# print("")
# get_metrics(X_tr, y_tr, X_val, y_val, y_pred_gslog_tr, y_pred_gslog_val,␣
↪gslog)

# Best: 0.522622 using {'C': 1, 'penalty': 'l2', 'solver': 'newton-cg'}
```

[37]:
```
logb = LogisticRegression(C=1, penalty='l2', solver='newton-cg').fit(X_tr, y_tr)
y_pred_logb_tr = logb.predict(X_tr)
y_pred_logb_val = logb.predict(X_val)
get_metric(X_tr, y_tr, X_val, y_val, y_pred_logb_tr, y_pred_logb_val, logb)
```

```
Training Accuracy:  0.8087142857142857
Validation Accuracy:  0.8071666666666667
Training F1 Score:  0.40320903283316006
Validation F1 Score:  0.380952380952381
Training AUC Score:  0.7476971040793051
Validation AUC Score:  0.7467126831177808
Training Recall Score:  0.2914518900343643
Validation Recall Score:  0.27113480578827115
Training Precision Score:  0.6539759036144578
Validation Precision Score:  0.6402877697841727
Training Average Precision Score:  0.5205202430336466
Validation Average Precision Score:  0.4986582661881126
```

4.2 Random Forest Classifier with GridSearchCV

[27]:
```
# 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)
# print("Best: %f using %s" % (gsrfc.best_score_, gsrfc.best_params_))
# print("")
# get_metrics(X_tr, y_tr, X_val, y_val, y_pred_gsrfc_tr, y_pred_gsrfc_val,␣
 ↪gsrfc)

# Best: 0.565196 using {'criterion': 'gini', 'max_depth': 8, 'min_samples_leaf':
 ↪ 1, 'min_samples_split': 2, 'n_estimators': 400}
```

[53]:
```
rfcb = RandomForestClassifier(criterion='gini', max_depth=8,␣
 ↪min_samples_leaf=1, min_samples_split=2, n_estimators=400).fit(X_tr, y_tr)
y_pred_rfcb_tr = rfcb.predict(X_tr)
y_pred_rfcb_val = rfcb.predict(X_val)
get_metric(X_tr, y_tr, X_val, y_val, y_pred_rfcb_tr, y_pred_rfcb_val, rfcb)
```

```
Training Accuracy:  0.8333333333333334
Validation Accuracy:  0.82
Training F1 Score:  0.5149667405764967
Validation F1 Score:  0.46375372393247266
Training AUC Score:  0.8263813590913907
Validation AUC Score:  0.7805258862036932
Training Recall Score:  0.39905498281786944
Validation Recall Score:  0.3556740289413557
Training Precision Score:  0.72578125
Validation Precision Score:  0.666191155492154
Training Average Precision Score:  0.6523890600337748
Validation Average Precision Score:  0.5445212676503224
```

4.3 AdaBoost Classifier with GridSearchCV

[28]:
```
# 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)
# y_pred_gsabc_val = gsabc.predict(X_val)
# print("Best: %f using %s" % (gsabc.best_score_, gsabc.best_params_))
```

```
# print("")
# get_metrics(X_tr, y_tr, X_val, y_val, y_pred_gsabc_tr, y_pred_gsabc_val,␣
↪gsabc)

# Best: 0.545818 using {'learning_rate': 0.1, 'n_estimators': 200}
```

[43]:
```
abcb = AdaBoostClassifier(learning_rate=0.1, n_estimators=200).fit(X_tr, y_tr)
y_pred_abcb_tr = abcb.predict(X_tr)
y_pred_abcb_val = abcb.predict(X_val)
get_metric(X_tr, y_tr, X_val, y_val, y_pred_abcb_tr, y_pred_abcb_val, abcb)
```

```
Training Accuracy:  0.8192380952380952
Validation Accuracy:  0.8208333333333333
Training F1 Score:  0.44969556393157434
Validation F1 Score:  0.4438696326952923
Training AUC Score:  0.7866775976198166
Validation AUC Score:  0.7772694027703144
Training Recall Score:  0.3331185567010309
Validation Recall Score:  0.32673267326732675
Training Precision Score:  0.691793041926851
Validation Precision Score:  0.6919354838709677
Training Average Precision Score:  0.5528304845005094
Validation Average Precision Score:  0.5244329096074963
```

4.4 Gradient Boosting with GridSearchCV

[29]:
```
# 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)
# print("Best: %f using %s" % (gsgbc.best_score_, gsgbc.best_params_))
# print("")
# get_metric(X_tr, y_tr, X_val, y_val, y_pred_gsgbc_tr, y_pred_gsgbc_tr, gsgbc)

# Best: 0.558390 using {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators':␣
↪1000}
```

[44]:
```
gbcb = GradientBoostingClassifier(learning_rate=0.01, max_depth=3,␣
↪n_estimators=1000).fit(X_tr, y_tr)
y_pred_gbcb_tr = gbcb.predict(X_tr)
```

```
y_pred_gbcb_val = gbcb.predict(X_val)
get_metric(X_tr, y_tr, X_val, y_val, y_pred_gbcb_tr, y_pred_gbcb_val, gbcb)
```

```
Training Accuracy:  0.8276190476190476
Validation Accuracy:  0.8196666666666667
Training F1 Score:  0.49694274596998333
Validation F1 Score:  0.4622266401590458
Training AUC Score:  0.805111244939135
Validation AUC Score:  0.7812120218438938
Training Recall Score:  0.38402061855670105
Validation Recall Score:  0.3541507996953541
Training Precision Score:  0.7039370078740157
Validation Precision Score:  0.6652360515021459
Training Average Precision Score:  0.6003473510491001
Validation Average Precision Score:  0.5434988921201059
```

4.5 XGBoost Classifier with GridSearchCV

[32]:
```
# xgb = XGBClassifier()
# params = {'n_estimators': [50, 100, 150, 200],
#           'max_depth': [3, 5, 7, 10],
#           'min_child_weight': [2, 3, 4, 5]}
# gsxgb = GridSearchCV(estimator = xgb,
#                      param_grid = params,
#                      scoring = 'average_precision',
#                      cv = 5,
#                      n_jobs = -1).fit(X_tr, y_tr)
# y_pred_gsxgb_tr = gsxgb.predict(X_tr)
# y_pred_gsxgb_val = gsxgb.predict(X_val)
# print("Best: %f using %s" % (gsxgb.best_score_, gsxgb.best_params_))
# print("")
# get_metrics(X_tr, y_tr, X_val, y_val, y_pred_gsxgb_tr, y_pred_gsxgb_val,␣
↪gsxgb)

# Best: 0.555500 using {'max_depth': 3, 'min_child_weight': 5, 'n_estimators':␣
↪50}
```

[46]:
```
xgbb = XGBClassifier(max_depth=3, min_child_weight=1, n_estimators=50).
↪fit(X_tr, y_tr)
y_pred_xgbb_tr = xgbb.predict(X_tr)
y_pred_xgbb_val = xgbb.predict(X_val)
get_metric(X_tr, y_tr, X_val, y_val, y_pred_xgbb_tr, y_pred_xgbb_val, xgbb)
```

```
[11:59:18] WARNING: /Users/runner/miniforge3/conda-
bld/xgboost_1607604592557/work/src/learner.cc:1061: Starting in XGBoost 1.3.0,
the default evaluation metric used with the objective 'binary:logistic' was
changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
```

```
restore the old behavior.
Training Accuracy:  0.8276190476190476
Validation Accuracy:  0.8185
Training F1 Score:  0.49833702882483377
Validation F1 Score:  0.4579392732702837
Training AUC Score:  0.810540741434586
Validation AUC Score:  0.7767761488364293
Training Recall Score:  0.3861683848797251
Validation Recall Score:  0.3503427265803503
Training Precision Score:  0.70234375
Validation Precision Score:  0.6609195402298851
Training Average Precision Score:  0.6011613543281619
Validation Average Precision Score:  0.5393954705676095
```

Best Hyperparameters for each Model:

-

4.6 Evaluation Metrics

[47]:
```
data = {'Accuracy': [accuracy(y_val, y_pred_logb_val),
                     accuracy(y_val, y_pred_rfcb_val),
                     accuracy(y_val, y_pred_abcb_val),
                     accuracy(y_val, y_pred_gbcb_val),
                     accuracy(y_val, y_pred_xgbb_val)],
        'F1 Score': [f1(y_val, y_pred_logb_val),
                     f1(y_val, y_pred_rfcb_val),
                     f1(y_val, y_pred_abcb_val),
                     f1(y_val, y_pred_gbcb_val),
                     f1(y_val, y_pred_xgbb_val)],
        'Recall': [recall(y_val, y_pred_logb_val),
                   recall(y_val, y_pred_rfcb_val),
                   recall(y_val, y_pred_abcb_val),
                   recall(y_val, y_pred_gbcb_val),
                   recall(y_val, y_pred_xgbb_val)],
        'Precision': [precision(y_val, y_pred_logb_val),
                      precision(y_val, y_pred_rfcb_val),
                      precision(y_val, y_pred_abcb_val),
                      precision(y_val, y_pred_gbcb_val),
                      precision(y_val, y_pred_xgbb_val)],
        'PR AUC': [aps(X_val, y_val, logb),
                   aps(X_val, y_val, rfcb),
                   aps(X_val, y_val, abcb),
                   aps(X_val, y_val, gbcb),
                   aps(X_val, y_val, xgbb)]}
scores3 = pd.DataFrame(data=data, index = ['Logistic Regression with␣
↪GridSearchCV',
                                          'Random Forest with GridSearchCV',
```

```
                                 'AdaBoost with GridSearchCV',
                                 'Gradient Boosting with GridSearchCV',
                                 'XGBoost with GridSearchCV'])
```

[51]:
```
scores3
```

[51]:
```
                                       Accuracy  F1 Score    Recall  \
Logistic Regression with GridSearchCV  0.807167  0.380952  0.271135
Random Forest with GridSearchCV        0.820667  0.464143  0.354912
AdaBoost with GridSearchCV             0.820833  0.443870  0.326733
Gradient Boosting with GridSearchCV    0.819667  0.462227  0.354151
XGBoost with GridSearchCV              0.818500  0.457939  0.350343

                                       Precision    PR AUC
Logistic Regression with GridSearchCV   0.640288  0.498658
Random Forest with GridSearchCV         0.670504  0.545164
AdaBoost with GridSearchCV              0.691935  0.524433
Gradient Boosting with GridSearchCV     0.665236  0.543499
XGBoost with GridSearchCV               0.660920  0.539395
```

[49]:
```
scores3.to_csv("../data/charts/scores3.csv")
```

4.7 Pickle out best model

[54]:
```
rfcb
```

[54]:
```
RandomForestClassifier(max_depth=8, n_estimators=400)
```

[55]:
```
pickle_out = open("../data/best_model.pickle","wb")
pickle.dump(rfcb, pickle_out)
pickle_out.close()
```

5 Class Imbalance

[64]:
```
X_train.head()
```

[64]:
```
     limit  behind1    paid2  delayed  latemths  age  behind2  billed1  \
0  1790.26        0   179.13        0         0   44        0  1631.93
1  5728.83       -1   173.87        0         0   46       -1   891.69
2  3580.52       -1     0.00        0         0   47       -1   238.68
3  6086.88        0    89.26        0         0   29        0  2831.87
4  5370.78       -2  1171.37        0         0   33       -2   873.40

     avg_av    avail1
0  0.344578  0.088440
1  0.957227  0.844350
```

```
2  0.968650  0.933339
3  0.650602  0.534758
4  0.836153  0.837379
```

[65]:
```
X_validate.head()
```

[65]:
```
     limit  behind1   paid2  delayed  latemths  age  behind2  billed1  \
0  1074.16        0   71.61        0         0   25        0   317.38
1  5370.78        0  151.64        0         0   26        0  4895.86
2  2506.36        0  111.43        0         0   32        0  2510.73
3  4654.68        0   64.74        0         0   49        0   740.38
4  1790.26        0   53.71        1         1   36        0  3373.85

     avg_av    avail1
0  0.602052  0.704532
1  0.293715  0.088427
2  0.005217 -0.001744
3  0.883482  0.840939
4  0.188227 -0.884559
```

5.1 Dummy Classifier as Baseline

[87]:
```
dc = DummyClassifier(strategy='most_frequent').fit(X_tr, y_tr)
y_pred_dc_tr = dc.predict(X_tr)
y_pred_dc_val = dc.predict(X_val)
get_metric(X_tr, y_tr, X_val, y_val, y_pred_dc_tr, y_pred_dc_val, dc)
```

```
Training Accuracy:  0.7782857142857142
Validation Accuracy:  0.7811666666666667
Training F1 Score:  0.0
Validation F1 Score:  0.0
Training AUC Score:  0.5
Validation AUC Score:  0.5
Training Recall Score:  0.0
Validation Recall Score:  0.0
Training Precision Score:  0.0
Validation Precision Score:  0.0
Training Average Precision Score:  0.22171428571428572
Validation Average Precision Score:  0.21883333333333332
```

5.2 Ensemble Methods

[66]:
```
bc = BaggingClassifier(n_estimators=50, random_state=42).fit(X_tr, y_tr)
y_pred_bc_tr = bc.predict(X_tr)
y_pred_bc_val = bc.predict(X_val)
get_metric(X_tr, y_tr, X_val, y_val, y_pred_bc_tr, y_pred_bc_val, bc)
print("")
```

```python
print('Bagging Classifier Performance:')
print('Balanced training accuracy: {:.2f} - Geometric mean {:.2f}'.
 ↪format(balanced_accuracy_score(y_tr, y_pred_bc_tr),␣
 ↪geometric_mean_score(y_tr, y_pred_bc_tr)))
print('Balanced validation accuracy: {:.2f} - Geometric mean {:.2f}'.
 ↪format(balanced_accuracy_score(y_val, y_pred_bc_val),␣
 ↪geometric_mean_score(y_val, y_pred_bc_val)))
```

```
Training Accuracy:  0.996904761904762
Validation Accuracy:  0.807
Training F1 Score:  0.9929964443486693
Validation F1 Score:  0.44326923076923075
Training AUC Score:  0.9998954304300327
Validation AUC Score:  0.740359286457933
Training Recall Score:  0.9896907216494846
Validation Recall Score:  0.3511043412033511
Training Precision Score:  0.9963243243243243
Validation Precision Score:  0.6010430247718384
Training Average Precision Score:  0.9996190235446103
Validation Average Precision Score:  0.48114820511254575

Bagging Classifier Performance:
Balanced training accuracy: 0.99 - Geometric mean 0.99
Balanced validation accuracy: 0.64 - Geometric mean 0.57
```

[68]:
```python
bbc = BalancedBaggingClassifier(n_estimators=50, random_state=42).fit(X_tr,␣
 ↪y_tr)
y_pred_bbc_tr = bbc.predict(X_tr)
y_pred_bbc_val = bbc.predict(X_val)
get_metric(X_tr, y_tr, X_val, y_val, y_pred_bbc_tr, y_pred_bbc_val, bbc)
print("")
print('Balanced Bagging Classifier Performance:')
print('Balanced training accuracy: {:.2f} - Geometric mean {:.2f}'.
 ↪format(balanced_accuracy_score(y_tr, y_pred_bbc_tr),␣
 ↪geometric_mean_score(y_val, y_pred_bbc_tr)))
print('Balanced validation accuracy: {:.2f} - Geometric mean {:.2f}'.
 ↪format(balanced_accuracy_score(y_val, y_pred_bbc_val),␣
 ↪geometric_mean_score(y_val, y_pred_bbc_val)))
```

```
Training Accuracy:  0.9383333333333334
Validation Accuracy:  0.7631666666666667
Training F1 Score:  0.8778186621379375
Validation F1 Score:  0.5111799105607154
Training AUC Score:  0.9955504481714446
Validation AUC Score:  0.7528188434539899
Training Recall Score:  0.9991408934707904
Validation Recall Score:  0.5658796648895659
```

```
Training Precision Score:  0.7827696449604576
Validation Precision Score:  0.46612296110414053
Training Average Precision Score:  0.9819759873453777
Validation Average Precision Score:  0.499848233823849

Balanced Bagging Classifier Performance:
Balanced training accuracy: 0.96 - Geometric mean 0.00
Balanced validation accuracy: 0.69 - Geometric mean 0.68
```

5.3 Undersampling/Downsampling Methods for Majority Class

[69]:
```python
# separate minority and majority classes
majority = train[train.default==0]
minority = train[train.default==1]

#baseline counts
counter = Counter(y_tr)
print("Baseline: ", counter)
```

```
Baseline:  Counter({0: 16344, 1: 4656})
```

[70]:
```python
downsampled = resample(majority, replace = False, n_samples = len(minority),␣
 ↪random_state=42)
dns = pd.concat([downsampled, minority])
print(dns.default.value_counts())
```

```
1    4656
0    4656
Name: default, dtype: int64
```

[71]:
```python
ns = NearMiss(version=1, n_neighbors=3)
X_tr_nm, y_tr_nm = ns.fit_resample(X_tr, y_tr)
counter_nm = Counter(y_tr_nm)
print("Near Miss: ", counter_nm)
```

```
Near Miss:  Counter({0: 4656, 1: 4656})
```

[72]:
```python
ncr = NeighbourhoodCleaningRule(n_neighbors=3, threshold_cleaning=0.5)
X_tr_ncr, y_tr_ncr = ncr.fit_resample(X_tr, y_tr)
counter_ncr = Counter(y_tr_ncr)
print("Neighborhood Cleaning Rule: ", counter_ncr)
```

```
Neighborhood Cleaning Rule:  Counter({0: 10215, 1: 4656})
```

[73]:
```python
oss = OneSidedSelection(n_neighbors=1, n_seeds_S=200)
X_tr_oss, y_tr_oss = oss.fit_resample(X_tr, y_tr)
counter_oss = Counter(y_tr_oss)
print("One Sided Selection: ", counter_oss)
```

```
One Sided Selection:  Counter({0: 13578, 1: 4656})
```

5.3.1 TomekLinks

[131]:
```python
tl = TomekLinks()
sampling(X_tr, y_tr, X_val, y_val, tl, rfcb)
```

```
Training Count:  Counter({0: 14844, 1: 4656})
Validation Count:  Counter({0: 4271, 1: 1313})
Training Accuracy:  0.839948717948718
Validation Accuracy:  0.8236031518624641
Training F1 Score:  0.5706424542578071
Validation F1 Score:  0.5135802469135803
Training AUC Score:  0.8401417120643466
Validation AUC Score:  0.7965941507069677
Training Recall Score:  0.44544673539518903
Validation Recall Score:  0.39603960396039606
Training Precision Score:  0.7937236892460773
Validation Precision Score:  0.7303370786516854
Training Average Precision Score:  0.7042583598528724
Validation Average Precision Score:  0.6045978251476949
```

5.3.2 Edited Nearest Neighbor

[130]:
```python
from imblearn.under_sampling import EditedNearestNeighbours
enn = EditedNearestNeighbours()
sampling(X_tr, y_tr, X_val, y_val, enn, rfcb)
```

```
Training Count:  Counter({0: 9921, 1: 4656})
Validation Count:  Counter({0: 2811, 1: 1313})
Training Accuracy:  0.8489401111339782
Validation Accuracy:  0.8215324927255092
Training F1 Score:  0.7129822732012513
Validation F1 Score:  0.6654545454545455
Training AUC Score:  0.892582339052397
Validation AUC Score:  0.8495370840753724
Training Recall Score:  0.5874140893470791
Validation Recall Score:  0.5575019040365575
Training Precision Score:  0.9068302387267905
Validation Precision Score:  0.8252536640360767
Training Average Precision Score:  0.8537450767788628
Validation Average Precision Score:  0.791534203883941
```

5.4 Upsampling/Oversampling Methods for Minority Class

[86]:
```
# Random Upsampling
upsampled = resample(minority, replace=True, n_samples=len(majority),␣
 ↪random_state=42)
ups = pd.concat([majority, upsampled])
y = ups['default']
counter_upsample = Counter(y)
print(counter_upsample)
```

```
Counter({0: 16344, 1: 16344})
```

5.4.1 SMOTE

[132]:
```
sm = SMOTE(sampling_strategy='minority', random_state=42)
sampling(X_tr, y_tr, X_val, y_val, sm, rfcb)
```

```
Training Count:  Counter({0: 16344, 1: 16344})
Validation Count:  Counter({0: 4687, 1: 4687})
Training Accuracy:  0.7619309838472834
Validation Accuracy:  0.7251973543844676
Training F1 Score:  0.7467786021085513
Validation F1 Score:  0.70205875549387
Training AUC Score:  0.8465005150222252
Validation AUC Score:  0.8035167474972311
Training Recall Score:  0.7020925110132159
Validation Recall Score:  0.6475357371452956
Training Precision Score:  0.7975396163469558
Validation Precision Score:  0.7666077292245517
Training Average Precision Score:  0.8525415939064722
Validation Average Precision Score:  0.8114534881318238
```

5.4.2 ADASYN

[133]:
```
from imblearn.over_sampling import ADASYN
adsn = ADASYN()
sampling(X_tr, y_tr, X_val, y_val, adsn, rfcb)
```

```
Training Count:  Counter({1: 16573, 0: 16344})
Validation Count:  Counter({0: 4687, 1: 4536})
Training Accuracy:  0.7418355257161953
Validation Accuracy:  0.6912067656944595
Training F1 Score:  0.7368551433702856
Validation F1 Score:  0.6715109573241062
Training AUC Score:  0.8174122452175352
Validation AUC Score:  0.7659273661736146
Training Recall Score:  0.717914680504435
Validation Recall Score:  0.6417548500881834
```

```
Training Precision Score:  0.7568220851090898
Validation Precision Score:  0.7041606192549589
Training Average Precision Score:  0.8197428192789693
Validation Average Precision Score:  0.7676839081058273
```

5.5 Hybridized Methods

5.5.1 SMOTETomek

[134]:
```
from imblearn.combine import SMOTETomek
smtk = SMOTETomek()
sampling(X_tr, y_tr, X_val, y_val, smtk, rfcb)
```

```
Training Count:  Counter({0: 15662, 1: 15662})
Validation Count:  Counter({0: 4447, 1: 4447})
Training Accuracy:  0.7696335078534031
Validation Accuracy:  0.7366764110636385
Training F1 Score:  0.7545077226644893
Validation F1 Score:  0.7157076960427288
Training AUC Score:  0.8546939965049241
Validation AUC Score:  0.8153883616088726
Training Recall Score:  0.7080194100370323
Validation Recall Score:  0.6629188216775355
Training Precision Score:  0.8075298572676959
Validation Precision Score:  0.7776312318649433
Training Average Precision Score:  0.8599408404902521
Validation Average Precision Score:  0.8214997703621973
```

5.5.2 SMOTEENN

[136]:
```
from imblearn.combine import SMOTEENN
smenn = SMOTEENN(sampling_strategy="minority", n_jobs= -1)
sampling(X_tr, y_tr, X_val, y_val, smenn, rfcb)
```

```
Training Count:  Counter({1: 11341, 0: 8553})
Validation Count:  Counter({1: 3204, 0: 2392})
Training Accuracy:  0.8652357494722027
Validation Accuracy:  0.8186204431736955
Training F1 Score:  0.8755396685390651
Validation F1 Score:  0.831142904674763
Training AUC Score:  0.9462685882132698
Validation AUC Score:  0.9106238439408932
Training Recall Score:  0.8314963407106957
Validation Recall Score:  0.7796504369538078
Training Precision Score:  0.9245098039215687
Validation Precision Score:  0.8899180619878875
Training Average Precision Score:  0.9638863089945894
Validation Average Precision Score:  0.939762144075218
```