credit_risk_resampling

May 30, 2021

1 Credit Risk Resampling Techniques

[89]:
```
import warnings
warnings.filterwarnings('ignore')
```

[90]:
```
import numpy as np
import pandas as pd
from pathlib import Path
from collections import Counter
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import RandomOverSampler
from collections import Counter
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import balanced_accuracy_score
from sklearn.metrics import confusion_matrix
from imblearn.metrics import classification_report_imbalanced
from imblearn.under_sampling import RandomUnderSampler
from imblearn.combine import SMOTEENN
from sklearn.metrics import recall_score, precision_score
from sklearn.metrics.cluster import fowlkes_mallows_score
```

2 Read the CSV into DataFrame

[91]:
```
# Load the data
file_path = Path('Resources/lending_data.csv')
df = pd.read_csv(file_path)
df.head()
```

[91]:
```
   loan_size  interest_rate homeowner  borrower_income  debt_to_income  \
0    10700.0          7.672       own            52800        0.431818
1     8400.0          6.692       own            43600        0.311927
2     9000.0          6.963      rent            46100        0.349241
3    10700.0          7.664       own            52700        0.430740
```

```
4    10800.0          7.698  mortgage            53000        0.433962

   num_of_accounts  derogatory_marks  total_debt loan_status
0                5                 1       22800    low_risk
1                3                 0       13600    low_risk
2                3                 0       16100    low_risk
3                5                 1       22700    low_risk
4                5                 1       23000    low_risk
```

[92]:
```
df
```

[92]:
```
       loan_size  interest_rate homeowner  borrower_income  debt_to_income  \
0        10700.0          7.672       own            52800        0.431818
1         8400.0          6.692       own            43600        0.311927
2         9000.0          6.963      rent            46100        0.349241
3        10700.0          7.664       own            52700        0.430740
4        10800.0          7.698  mortgage            53000        0.433962
…            …              …       …                …              …
77531    19100.0         11.261       own            86600        0.653580
77532    17700.0         10.662  mortgage            80900        0.629172
77533    17600.0         10.595      rent            80300        0.626401
77534    16300.0         10.068  mortgage            75300        0.601594
77535    15600.0          9.742  mortgage            72300        0.585062

       num_of_accounts  derogatory_marks  total_debt loan_status
0                    5                 1       22800    low_risk
1                    3                 0       13600    low_risk
2                    3                 0       16100    low_risk
3                    5                 1       22700    low_risk
4                    5                 1       23000    low_risk
…                  …                …       …         …
77531               12                 2       56600   high_risk
77532               11                 2       50900   high_risk
77533               11                 2       50300   high_risk
77534               10                 2       45300   high_risk
77535                9                 2       42300   high_risk

[77536 rows x 9 columns]
```

3 Split the Data into Training and Testing

[93]:
```python
# Create our features
X = df[['loan_size', 'interest_rate', 'homeowner', 'borrower_income',
       'debt_to_income', 'num_of_accounts', 'derogatory_marks', 'total_debt']]
```

```python
# Create our target
y = df["loan_status"]
```

[94]:
```python
X.describe()
```

[94]:
```
          loan_size  interest_rate  borrower_income  debt_to_income  \
count  77536.000000   77536.000000     77536.000000    77536.000000
mean    9805.562577       7.292333     49221.949804        0.377318
std     2093.223153       0.889495      8371.635077        0.081519
min     5000.000000       5.250000     30000.000000        0.000000
25%     8700.000000       6.825000     44800.000000        0.330357
50%     9500.000000       7.172000     48100.000000        0.376299
75%    10400.000000       7.528000     51400.000000        0.416342
max    23800.000000      13.235000    105200.000000        0.714829

       num_of_accounts  derogatory_marks    total_debt
count     77536.000000      77536.000000  77536.000000
mean          3.826610          0.392308  19221.949804
std           1.904426          0.582086   8371.635077
min           0.000000          0.000000      0.000000
25%           3.000000          0.000000  14800.000000
50%           4.000000          0.000000  18100.000000
75%           4.000000          1.000000  21400.000000
max          16.000000          3.000000  75200.000000
```

[95]:
```python
# Check the balance of our target values
y.value_counts()
```

[95]:
```
low_risk     75036
high_risk     2500
Name: loan_status, dtype: int64
```

[96]:
```python
# Create X_train, X_test, y_train, y_test
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=32)
```

3.1 Data Pre-Processing

Scale the training and testing data using the `StandardScaler` from `sklearn`. Remember that when scaling the data, you only scale the features data (`X_train` and `X_testing`).

[97]:
```python
homeowner_dummies_train = pd.get_dummies(X_train["homeowner"])
homeowner_dummies_test = pd.get_dummies(X_test["homeowner"])
```

[98]:
```python
homeowner_dummies_train
```

[98]:

```
       mortgage  own  rent
11447         1    0     0
34848         1    0     0
74571         1    0     0
1662          1    0     0
7058          0    1     0
…           …  …   …
24828         0    1     0
20414         1    0     0
60284         1    0     0
75062         0    1     0
10967         1    0     0

[58152 rows x 3 columns]
```

[99]:

```
dummies_df_train = pd.concat([X_train, homeowner_dummies_train], axis="columns")
dummies_df_train.drop("homeowner", axis="columns", inplace=True)

dummies_df_test = pd.concat([X_test, homeowner_dummies_test], axis="columns")
dummies_df_test.drop("homeowner", axis="columns", inplace=True)
```

[100]:

```
dummies_df_test
```

[100]:

```
       loan_size  interest_rate  borrower_income  debt_to_income  \
3376      8300.0          6.673            43400        0.308756
26052     9500.0          7.148            47900        0.373695
39747    11200.0          7.889            54800        0.452555
58554     9200.0          7.034            46800        0.358974
66653    11100.0          7.840            54400        0.448529
…            …              …                …               …
73470    12500.0          8.428            59900        0.499165
7674      8100.0          6.572            42400        0.292453
45720     9800.0          7.288            49200        0.390244
46091     8900.0          6.908            45600        0.342105
10366     8700.0          6.804            44600        0.327354

       num_of_accounts  derogatory_marks  total_debt  mortgage  own  rent
3376                 3                 0       13400         1    0     0
26052                4                 0       17900         0    1     0
39747                5                 1       24800         0    1     0
58554                3                 0       16800         0    1     0
66653                5                 1       24400         1    0     0
…                  …                 …           …       …   …    …
73470                6                 1       29900         0    1     0
7674                 2                 0       12400         1    0     0
45720                4                 0       19200         0    1     0
46091                3                 0       15600         1    0     0
```

```
10366                   3                  0        14600           0    1     0

[19384 rows x 10 columns]
```

[101]:
```
# Create the StandardScaler instance
scaler = StandardScaler()
```

[102]:
```
# Fit the Standard Scaler with the training data
# When fitting scaling functions, only train on the training dataset
X_train_scaled = scaler.fit(dummies_df_train)
```

[103]:
```
# Scale the training and testing data
X_test_scaled = StandardScaler().fit(dummies_df_test)
```

[104]:
```
X_train = dummies_df_train.copy(deep=True)
```

[105]:
```
X_test = dummies_df_test.copy(deep=True)
```

4 Simple Logistic Regression

[106]:
```
model = LogisticRegression(solver='lbfgs', random_state=1)
model.fit(X_train, y_train)
```

[106]:
```
LogisticRegression(random_state=1)
```

[107]:
```
# Calculated the balanced accuracy score
y_pred = model.predict(X_test)
balanced_accuracy_score(y_test, y_pred)
```

[107]:
```
0.9422645899808877
```

[108]:
```
# Display the confusion matrix
confusion_matrix(y_test, y_pred)
```

[108]:
```
array([[  559,    69],
       [  105, 18651]])
```

[109]:
```
# Print the imbalanced classification report
print(classification_report_imbalanced(y_test, y_pred))
```

```
                   pre       rec       spe        f1       geo       iba
sup

  high_risk       0.84      0.89      0.99      0.87      0.94      0.88
628
```

```
   low_risk       1.00      0.99      0.89      1.00      0.94      0.89
18756

avg / total       0.99      0.99      0.89      0.99      0.94      0.89
19384
```

5 Oversampling

In this section, you will compare two oversampling algorithms to determine which algorithm results in the best performance. You will oversample the data using the naive random oversampling algorithm and the SMOTE algorithm. For each algorithm, be sure to complete the folliowing steps:

1. View the count of the target classes using `Counter` from the collections library.
2. Use the resampled data to train a logistic regression model.
3. Calculate the balanced accuracy score from sklearn.metrics.
4. Print the confusion matrix from sklearn.metrics.
5. Generate a classication report using the `imbalanced_classification_report` from imbalanced-learn.

Note: Use a random state of 1 for each sampling algorithm to ensure consistency between tests

5.0.1 Naive Random Oversampling

[53]:
```
# Resample the training data with the RandomOversampler
ros = RandomOverSampler(random_state=45)
X_resampled, y_resampled = ros.fit_resample(X_train, y_train)

# View the count of target classes with Counter
Counter(y_resampled)
```

[53]:
```
Counter({'low_risk': 56280, 'high_risk': 56280})
```

[60]:
```
# Train the Logistic Regression model using the resampled data
model.fit(X_resampled, y_resampled)
y_pred = model.predict(X_resampled)
```

[62]:
```
# Calculated the balanced accuracy score
balanced_accuracy_score(y_resampled, y_pred)
```

[62]:
```
0.9945184790334044
```

[64]:
```
# Display the confusion matrix
confusion_matrix(y_resampled, y_pred)
```

[64]:
```
array([[55969,   311],
       [  306, 55974]])
```

[65]:
```
# Print the imbalanced classification report
print(classification_report_imbalanced(y_resampled, y_pred))
```

```
                   pre       rec       spe        f1       geo       iba
sup

  high_risk       0.99      0.99      0.99      0.99      0.99      0.99
56280
   low_risk       0.99      0.99      0.99      0.99      0.99      0.99
56280

avg / total       0.99      0.99      0.99      0.99      0.99      0.99
112560
```

5.0.2 SMOTE Oversampling

[68]:
```
# Resample the training data with SMOTE
X_resampled, y_resampled = SMOTE(random_state=40, sampling_strategy=1.0).
 ↪fit_resample(
    X_train, y_train)

# View the count of target classes with Counter
Counter(y_resampled)
```

[68]:
```
Counter({'low_risk': 56280, 'high_risk': 56280})
```

[70]:
```
# Train the Logistic Regression model using the resampled data
model.fit(X_resampled, y_resampled)
y_pred = model.predict(X_resampled)
```

[71]:
```
# Calculated the balanced accuracy score
balanced_accuracy_score(y_resampled, y_pred)
```

[71]:
```
0.9943407960199004
```

[72]:
```
# Display the confusion matrix
confusion_matrix(y_resampled, y_pred)
```

[72]:
```
array([[55950,   330],
       [  307, 55973]])
```

[73]:
```
# Print the imbalanced classification report
print(classification_report_imbalanced(y_resampled, y_pred))
```

```
                   pre       rec       spe        f1       geo       iba
sup

  high_risk       0.99      0.99      0.99      0.99      0.99      0.99
56280
   low_risk       0.99      0.99      0.99      0.99      0.99      0.99
56280

avg / total       0.99      0.99      0.99      0.99      0.99      0.99
112560
```

6 Undersampling

In this section, you will test an undersampling algorithm to determine which algorithm results in the best performance compared to the oversampling algorithms above. You will undersample the data using the Cluster Centroids algorithm and complete the folliowing steps:

1. View the count of the target classes using `Counter` from the collections library.
2. Use the resampled data to train a logistic regression model.
3. Calculate the balanced accuracy score from sklearn.metrics.
4. Display the confusion matrix from sklearn.metrics.
5. Generate a classication report using the `imbalanced_classification_report` from imbalanced-learn.

Note: Use a random state of 1 for each sampling algorithm to ensure consistency between tests

[75]:
```
# Resample the data using the ClusterCentroids resampler
ros = RandomUnderSampler(random_state=32)
X_resampled, y_resampled = ros.fit_resample(X_train, y_train)

# View the count of target classes with Counter
Counter(y_resampled)
```

[75]:
```
Counter({'high_risk': 1872, 'low_risk': 1872})
```

[76]:
```
# Train the Logistic Regression model using the resampled data
model.fit(X_resampled, y_resampled)
y_pred = model.predict(X_resampled)
```

[77]:
```
# Calculate the balanced accuracy score
balanced_accuracy_score(y_resampled, y_pred)
```

[77]:
```
0.9943910256410257
```

[78]:
```
# Display the confusion matrix
confusion_matrix(y_resampled, y_pred)
```

[78]:
```
array([[1861,   11],
       [  10, 1862]])
```

[79]:
```
# Print the imbalanced classification report
print(classification_report_imbalanced(y_resampled, y_pred))
```

```
                   pre       rec       spe        f1       geo       iba
sup

  high_risk       0.99      0.99      0.99      0.99      0.99      0.99
1872
   low_risk       0.99      0.99      0.99      0.99      0.99      0.99
1872

avg / total       0.99      0.99      0.99      0.99      0.99      0.99
3744
```

7 Combination (Over and Under) Sampling

In this section, you will test a combination over- and under-sampling algorithm to determine if the algorithm results in the best performance compared to the other sampling algorithms above. You will resample the data using the SMOTEENN algorithm and complete the folliowing steps:

1. View the count of the target classes using `Counter` from the collections library.
2. Use the resampled data to train a logistic regression model.
3. Calculate the balanced accuracy score from sklearn.metrics.
4. Display the confusion matrix from sklearn.metrics.
5. Generate a classication report using the `imbalanced_classification_report` from imbalanced-learn.

Note: Use a random state of 1 for each sampling algorithm to ensure consistency between tests

[81]:
```
# Resample the training data with SMOTEENN
sm = SMOTEENN(random_state=29)
X_resampled, y_resampled = sm.fit_resample(X_train, y_train)

# View the count of target classes with Counter
Counter(y_resampled)
```

[81]:
```
Counter({'high_risk': 55555, 'low_risk': 55906})
```

[82]:
```
# Train the Logistic Regression model using the resampled data
model.fit(X_resampled, y_resampled)
```

```
y_pred = model.predict(X_resampled)
```

[83]:
```
# Calculate the balanced accuracy score
balanced_accuracy_score(y_resampled, y_pred)
```

[83]:
```
0.9988480449860128
```

[84]:
```
# Display the confusion matrix
confusion_matrix(y_resampled, y_pred)
```

[84]:
```
array([[55428,   127],
       [    1, 55905]])
```

[85]:
```
# Print the imbalanced classification report
print(classification_report_imbalanced(y_resampled, y_pred))
```

```
                   pre       rec       spe        f1       geo       iba
sup

  high_risk       1.00      1.00      1.00      1.00      1.00      1.00
55555
   low_risk       1.00      1.00      1.00      1.00      1.00      1.00
55906

avg / total       1.00      1.00      1.00      1.00      1.00      1.00
111461
```

8 Final Questions

1. Which model had the best balanced accuracy score?

   SMOTTEN with a balanced accuracy score of 0.9988480449860128

2. Which model had the best recall score?

   SMOTTEN with a recall of 1

3. Which model had the best geometric mean score?

   SMOTTEN with a geometric mean score

[ ]:
```

```