credit_risk_resampling

September 24, 2019

1 Credit Risk Resampling Techniques

[1]:
```
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
import numpy as np
import pandas as pd
from pathlib import Path
from collections import Counter
```

2 Read the CSV and Perform Basic Data Cleaning

[3]:
```
columns = [
    "loan_amnt", "int_rate", "installment", "home_ownership",
    "annual_inc", "verification_status", "issue_d", "loan_status",
    "pymnt_plan", "dti", "delinq_2yrs", "inq_last_6mths",
    "open_acc", "pub_rec", "revol_bal", "total_acc",
    "initial_list_status", "out_prncp", "out_prncp_inv", "total_pymnt",
    "total_pymnt_inv", "total_rec_prncp", "total_rec_int", "total_rec_late_fee",
    "recoveries", "collection_recovery_fee", "last_pymnt_amnt", "next_pymnt_d",
    "collections_12_mths_ex_med", "policy_code", "application_type",␣
↪"acc_now_delinq",
    "tot_coll_amt", "tot_cur_bal", "open_acc_6m", "open_act_il",
    "open_il_12m", "open_il_24m", "mths_since_rcnt_il", "total_bal_il",
    "il_util", "open_rv_12m", "open_rv_24m", "max_bal_bc",
    "all_util", "total_rev_hi_lim", "inq_fi", "total_cu_tl",
    "inq_last_12m", "acc_open_past_24mths", "avg_cur_bal", "bc_open_to_buy",
    "bc_util", "chargeoff_within_12_mths", "delinq_amnt", "mo_sin_old_il_acct",
    "mo_sin_old_rev_tl_op", "mo_sin_rcnt_rev_tl_op", "mo_sin_rcnt_tl",␣
↪"mort_acc",
    "mths_since_recent_bc", "mths_since_recent_inq", "num_accts_ever_120_pd",␣
↪"num_actv_bc_tl",
    "num_actv_rev_tl", "num_bc_sats", "num_bc_tl", "num_il_tl",
    "num_op_rev_tl", "num_rev_accts", "num_rev_tl_bal_gt_0",
    "num_sats", "num_tl_120dpd_2m", "num_tl_30dpd", "num_tl_90g_dpd_24m",
```

```
    "num_tl_op_past_12m", "pct_tl_nvr_dlq", "percent_bc_gt_75",␣
 ↪"pub_rec_bankruptcies",
    "tax_liens", "tot_hi_cred_lim", "total_bal_ex_mort", "total_bc_limit",
    "total_il_high_credit_limit", "hardship_flag", "debt_settlement_flag"
]

target = ["loan_status"]
```

[4]:
```
# Load the data
file_path = Path('../Resources/LoanStats_2019Q1.csv.zip')
df = pd.read_csv(file_path, skiprows=1)[:-2]
df = df.loc[:, columns].copy()

# Drop the null columns where all values are null
df = df.dropna(axis='columns', how='all')

# Drop the null rows
df = df.dropna()

# Remove the `Issued` loan status
issued_mask = df['loan_status'] != 'Issued'
df = df.loc[issued_mask]

# convert interest rate to numerical
df['int_rate'] = df['int_rate'].str.replace('%', '')
df['int_rate'] = df['int_rate'].astype('float') / 100


# Convert the target column values to low_risk and high_risk based on their␣
 ↪values
x = {'Current': 'low_risk'}
df = df.replace(x)

x = dict.fromkeys(['Late (31-120 days)', 'Late (16-30 days)', 'Default', 'In␣
 ↪Grace Period'], 'high_risk')
df = df.replace(x)

df.reset_index(inplace=True, drop=True)

df.head()
```

[4]:

	loan_amnt	int_rate	installment	home_ownership	annual_inc	\
0	10500.0	0.1719	375.35	RENT	66000.0	
1	25000.0	0.2000	929.09	MORTGAGE	105000.0	
2	20000.0	0.2000	529.88	MORTGAGE	56000.0	
3	10000.0	0.1640	353.55	RENT	92000.0	
4	22000.0	0.1474	520.39	MORTGAGE	52000.0	

```
  verification_status   issue_d loan_status pymnt_plan    dti  ...  \
0     Source Verified  Mar-2019    low_risk          n  27.24  ...
1            Verified  Mar-2019    low_risk          n  20.23  ...
2            Verified  Mar-2019    low_risk          n  24.26  ...
3            Verified  Mar-2019    low_risk          n  31.44  ...
4        Not Verified  Mar-2019    low_risk          n  18.76  ...

   pct_tl_nvr_dlq  percent_bc_gt_75  pub_rec_bankruptcies  tax_liens  \
0            85.7             100.0                   0.0        0.0
1            91.2              50.0                   1.0        0.0
2            66.7              50.0                   0.0        0.0
3           100.0              50.0                   1.0        0.0
4           100.0               0.0                   0.0        0.0

   tot_hi_cred_lim  total_bal_ex_mort total_bc_limit  \
0          65687.0            38199.0         2000.0
1         271427.0            60641.0        41200.0
2          60644.0            45684.0         7500.0
3          99506.0            68784.0        19700.0
4         219750.0            25919.0        27600.0

   total_il_high_credit_limit  hardship_flag  debt_settlement_flag
0                     61987.0              N                     N
1                     49197.0              N                     N
2                     43144.0              N                     N
3                     76506.0              N                     N
4                     20000.0              N                     N

[5 rows x 86 columns]
```

3 Split the Data into Training and Testing

[5]:
```
# Create our features
X = # YOUR CODE HERE

# Create our target
y = # YOUR CODE HERE
```

[6]:
```
X.describe()
```

[6]:
```
          loan_amnt      int_rate   installment    annual_inc           dti  \
count  68817.000000  68817.000000  68817.000000  6.881700e+04  68817.000000
mean   16677.594562      0.127718    480.652863  8.821371e+04     21.778153
std    10277.348590      0.048130    288.062432  1.155800e+05     20.199244
min     1000.000000      0.060000     30.890000  4.000000e+01      0.000000
25%     9000.000000      0.088100    265.730000  5.000000e+04     13.890000
50%    15000.000000      0.118000    404.560000  7.300000e+04     19.760000
```

```
75%    24000.000000      0.155700    648.100000  1.040000e+05     26.660000
max    40000.000000      0.308400   1676.230000  8.797500e+06    999.000000

        delinq_2yrs  inq_last_6mths      open_acc       pub_rec  \
count  68817.000000   68817.000000  68817.000000  68817.000000
mean       0.217766       0.497697     12.587340      0.126030
std        0.718367       0.758122      6.022869      0.336797
min        0.000000       0.000000      2.000000      0.000000
25%        0.000000       0.000000      8.000000      0.000000
50%        0.000000       0.000000     11.000000      0.000000
75%        0.000000       1.000000     16.000000      0.000000
max       18.000000       5.000000     72.000000      4.000000

           revol_bal  ...  issue_d_Mar-2019  pymnt_plan_n  \
count   68817.000000  ...      68817.000000       68817.0
mean    17604.142828  ...          0.177238           1.0
std     21835.880400  ...          0.381873           0.0
min         0.000000  ...          0.000000           1.0
25%      6293.000000  ...          0.000000           1.0
50%     12068.000000  ...          0.000000           1.0
75%     21735.000000  ...          0.000000           1.0
max    587191.000000  ...          1.000000           1.0

       initial_list_status_f  initial_list_status_w  next_pymnt_d_Apr-2019  \
count           68817.000000           68817.000000           68817.000000
mean                0.123879               0.876121               0.383161
std                 0.329446               0.329446               0.486161
min                 0.000000               0.000000               0.000000
25%                 0.000000               1.000000               0.000000
50%                 0.000000               1.000000               0.000000
75%                 0.000000               1.000000               1.000000
max                 1.000000               1.000000               1.000000

       next_pymnt_d_May-2019  application_type_Individual  \
count           68817.000000                 68817.000000
mean                0.616839                     0.860340
std                 0.486161                     0.346637
min                 0.000000                     0.000000
25%                 0.000000                     1.000000
50%                 1.000000                     1.000000
75%                 1.000000                     1.000000
max                 1.000000                     1.000000

       application_type_Joint App  hardship_flag_N  debt_settlement_flag_N
count                68817.000000          68817.0                 68817.0
mean                     0.139660              1.0                     1.0
std                      0.346637              0.0                     0.0
```

```
min                    0.000000              1.0                    1.0
25%                    0.000000              1.0                    1.0
50%                    0.000000              1.0                    1.0
75%                    0.000000              1.0                    1.0
max                    1.000000              1.0                    1.0

[8 rows x 95 columns]
```

[7]:
```
# Check the balance of our target values
y['loan_status'].value_counts()
```

[7]:
```
low_risk     68470
high_risk      347
Name: loan_status, dtype: int64
```

[8]:
```
# Create X_train, X_test, y_train, y_test
# YOUR CODE HERE
```

4 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

4.0.1 Naive Random Oversampling

[9]:
```
# Resample the training data with the RandomOversampler
# YOUR CODE HERE
```

[9]:
```
Counter({'low_risk': 51366, 'high_risk': 51366})
```

[10]:
```
# Train the Logistic Regression model using the resampled data
# YOUR CODE HERE
```

[10]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=1, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

[11]:
```
# Calculated the balanced accuracy score
# YOUR CODE HERE
```

[11]:
```
0.7163908158823367
```

[12]:
```
# Display the confusion matrix
# YOUR CODE HERE
```

[12]:
```
array([[   73,    28],
       [ 4960, 12144]])
```

[13]:
```
# Print the imbalanced classification report
# YOUR CODE HERE
```

```
                   pre       rec       spe        f1       geo       iba
sup

  high_risk       0.01      0.72      0.71      0.03      0.72      0.51
101
   low_risk       1.00      0.71      0.72      0.83      0.72      0.51
17104

avg / total       0.99      0.71      0.72      0.82      0.72      0.51
17205
```

4.0.2 SMOTE Oversampling

[ ]:
```
# Resample the training data with SMOTE
# YOUR CODE HERE
```

[ ]:
```
Counter({'low_risk': 51366, 'high_risk': 51366})
```

[ ]:
```
# Train the Logistic Regression model using the resampled data
# YOUR CODE HERE
```

[ ]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=1, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

[ ]:
```
# Calculated the balanced accuracy score
# YOUR CODE HERE
```

[ ]:
```
0.7001170474858525
```

[ ]:
```
# Display the confusion matrix
# YOUR CODE HERE
```

[ ]:
```
array([[   71,    30],
       [ 5178, 11926]])
```

[ ]:
```
# Print the imbalanced classification report
# YOUR CODE HERE
```

```
                   pre       rec       spe        f1       geo       iba
sup

  high_risk       0.01      0.70      0.70      0.03      0.70      0.49
101
   low_risk       1.00      0.70      0.70      0.82      0.70      0.49
17104

avg / total       0.99      0.70      0.70      0.82      0.70      0.49
17205
```

5 Undersampling

In this section, you will test an undersampling algorithms 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. 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

[ ]:
```
# Resample the data using the ClusterCentroids resampler
# YOUR CODE HERE
```

[ ]:
```
Counter({'high_risk': 246, 'low_risk': 246})
```

[ ]:
```
# Train the Logistic Regression model using the resampled data
# YOUR CODE HERE
```

[ ]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=1, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

[ ]:
```
# Calculated the balanced accuracy score
# YOUR CODE HERE
```

[ ]:
```
0.6433412021043078
```

[ ]:
```
# Display the confusion matrix
# YOUR CODE HERE
```

[ ]:
```
array([[  82,   19],
       [8983, 8121]])
```

[ ]:
```
# Print the imbalanced classification report
# YOUR CODE HERE
```

```
                   pre       rec       spe        f1       geo       iba
sup

  high_risk       0.01      0.81      0.47      0.02      0.62      0.40
101
   low_risk       1.00      0.47      0.81      0.64      0.62      0.37
17104

avg / total       0.99      0.48      0.81      0.64      0.62      0.37
17205
```

6 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. 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

[ ]:
```
# Resample the training data with SMOTEENN
# YOUR CODE HERE
```

[ ]:
```
Counter({'high_risk': 51361, 'low_risk': 46653})
```

[ ]:
```
# Train the Logistic Regression model using the resampled data
# YOUR CODE HERE
```

[ ]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=1, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

[ ]:
```
# Calculated the balanced accuracy score
# YOUR CODE HERE
```

[ ]:
```
0.69732081662329
```

[ ]:
```
# Display the confusion matrix
# YOUR CODE HERE
```

[ ]:
```
array([[   72,    29],
       [ 5443, 11661]])
```

[ ]:
```
# Print the imbalanced classification report
# YOUR CODE HERE
```

```
                   pre       rec       spe        f1       geo       iba
sup

  high_risk       0.01      0.71      0.68      0.03      0.70      0.49
101
   low_risk       1.00      0.68      0.71      0.81      0.70      0.48
17104

avg / total       0.99      0.68      0.71      0.81      0.70      0.48
17205
```

[ ]:
```

```