CreditRiskModelling

September 28, 2023

[1]:
```
#Import libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score, precision_score, recall_score, f1_score
```

[2]:
```
# Load the dataset
df = pd.read_csv("lending_club_loan_dataset.csv")
df
```

[2]:
```
             id grade  annual_inc  short_emp  emp_length_num home_ownership  \
0      11454641     A    100000.0          1               1           RENT
1       9604874     A     83000.0          0               4            OWN
2       9684700     D     78000.0          0              11       MORTGAGE
3       9695736     D     37536.0          0               6       MORTGAGE
4       9795013     D     65000.0          0              11       MORTGAGE
…           …   …         …        …             …            …
19995   6595657     B     27000.0          0               9           RENT
19996   1576331     B     45000.0          0               2       MORTGAGE
19997   6645736     B    104000.0          0               5       MORTGAGE
19998   6625736     A     38400.0          0               2       MORTGAGE
19999   6625685     B    150000.0          0              11       MORTGAGE

         dti             purpose        term  last_delinq_none  \
0      26.27         credit_card   36 months                 1
1       5.39         credit_card   36 months                 0
2      18.45  debt_consolidation   60 months                 1
3      12.28             medical   60 months                 0
4      11.26  debt_consolidation   36 months                 0
…        …                 …         …               …
19995  18.36  debt_consolidation   36 months                 1
19996  23.22      major_purchase   36 months                 0
19997  13.27  debt_consolidation   36 months                 1
19998  12.84  debt_consolidation   36 months                 0
19999   2.20         credit_card   36 months                 0
```

```
       last_major_derog_none  revol_util  total_rec_late_fee  od_ratio  \
0                        NaN        43.2                 0.0  0.160624
1                        NaN        21.5                 0.0  0.810777
2                        NaN        46.3                 0.0  0.035147
3                        NaN        10.7                 0.0  0.534887
4                        NaN        15.2                 0.0  0.166500
…                        …          …                   …        …
19995                    NaN        46.5                 0.0  0.821782
19996                    NaN        46.2                 0.0  0.652200
19997                    NaN        78.5                 0.0  0.482555
19998                    NaN        47.4                 0.0  0.822980
19999                    NaN        40.7                 0.0  0.201388

       bad_loan
0             0
1             0
2             1
3             1
4             0
…           …
19995         1
19996         0
19997         0
19998         0
19999         0

[20000 rows x 15 columns]
```

[3]:
```python
# Display basic information about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 15 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   id                     20000 non-null  int64
 1   grade                  20000 non-null  object
 2   annual_inc             20000 non-null  float64
 3   short_emp              20000 non-null  int64
 4   emp_length_num         20000 non-null  int64
 5   home_ownership         18509 non-null  object
 6   dti                    19846 non-null  float64
 7   purpose                20000 non-null  object
 8   term                   20000 non-null  object
 9   last_delinq_none       20000 non-null  int64
```

```
 10  last_major_derog_none  574 non-null    float64
 11  revol_util             20000 non-null  float64
 12  total_rec_late_fee     20000 non-null  float64
 13  od_ratio               20000 non-null  float64
 14  bad_loan               20000 non-null  int64
dtypes: float64(6), int64(5), object(4)
memory usage: 2.3+ MB
```

[7]:
```
# Check the shape of the dataset
df.shape
```

[7]:
```
(20000, 15)
```

[8]:
```
# Check for missing values
df.isnull().sum()
```

[8]:
```
id                           0
grade                        0
annual_inc                   0
short_emp                    0
emp_length_num               0
home_ownership            1491
dti                        154
purpose                      0
term                         0
last_delinq_none             0
last_major_derog_none    19426
revol_util                   0
total_rec_late_fee           0
od_ratio                     0
bad_loan                     0
dtype: int64
```

[10]:
```
# Drop unnecessary columns
df.drop(["id", "last_major_derog_none"], axis=1, inplace=True)
df.head()
```

[10]:
```
  grade  annual_inc  short_emp  emp_length_num home_ownership    dti  \
0     A    100000.0          1               1           RENT  26.27
1     A     83000.0          0               4            OWN   5.39
2     D     78000.0          0              11       MORTGAGE  18.45
3     D     37536.0          0               6       MORTGAGE  12.28
4     D     65000.0          0              11       MORTGAGE  11.26

              purpose        term  last_delinq_none  revol_util  \
0         credit_card   36 months                 1        43.2
1         credit_card   36 months                 0        21.5
```

```
2  debt_consolidation   60 months                1        46.3
3             medical   60 months                0        10.7
4  debt_consolidation   36 months                0        15.2

   total_rec_late_fee  od_ratio  bad_loan
0                 0.0  0.160624         0
1                 0.0  0.810777         0
2                 0.0  0.035147         1
3                 0.0  0.534887         1
4                 0.0  0.166500         0
```

[12]:
```
# Display summary statistics of the dataset
df.describe()
```

[12]:
```
          annual_inc     short_emp  emp_length_num           dti  \
count   20000.000000  20000.000000     20000.00000  19846.000000
mean    73349.578350      0.112500         6.82140     16.587841
std     45198.567255      0.315989         3.77423      7.585812
min      8412.000000      0.000000         0.00000      0.000000
25%     47000.000000      0.000000         3.00000     10.852500
50%     65000.000000      0.000000         7.00000     16.190000
75%     88000.000000      0.000000        11.00000     22.060000
max   1000000.000000      1.000000        11.00000     34.990000

       last_delinq_none    revol_util  total_rec_late_fee      od_ratio  \
count      20000.000000  20000.000000        20000.000000  20000.000000
mean           0.546600     55.958148            0.290622      0.504430
std            0.497836     42.117456            3.108654      0.287720
min            0.000000      0.000000            0.000000      0.000077
25%            0.000000     38.800000            0.000000      0.257356
50%            1.000000     57.100000            0.000000      0.506681
75%            1.000000     73.900000            0.000000      0.753771
max            1.000000   5010.000000           96.466600      0.999894

          bad_loan
count  20000.00000
mean       0.20000
std        0.40001
min        0.00000
25%        0.00000
50%        0.00000
75%        0.00000
max        1.00000
```

[14]:
```
# Handle missing values in the 'dti' column by filling with the mean
mean_dti = df["dti"].mean()
df["dti"].fillna(mean_dti, inplace=True)
```

[16]:
```
# Check the distribution of the 'home_ownership' column
df['home_ownership'].value_counts()

# Fill missing values in the 'home_ownership' column with the mode
mode_ho = df["home_ownership"].mode().iloc[0]
df["home_ownership"].fillna(mode_ho, inplace=True)
df["home_ownership"].isnull().sum()
```

[16]: 0

[18]:
```
# Check for duplicate rows in the dataset
df.duplicated().sum()
```

[18]: 0

[21]:
```
# Remove "months" from the "term" column and convert it to an integer
df['term'] = df['term'].str.strip().str.split(" ").str[0].astype(int)
```

[23]:
```
# Use one-hot encoding to convert categorical columns to numerical
df = pd.get_dummies(df, columns=['home_ownership', 'purpose', 'grade'],␣
↪prefix=['home_ownership', 'purpose', 'grade'], drop_first=True)
df.head()
```

[23]:
```
   annual_inc  short_emp  emp_length_num    dti  term  last_delinq_none  \
0    100000.0          1               1  26.27    36                 1
1     83000.0          0               4   5.39    36                 0
2     78000.0          0              11  18.45    60                 1
3     37536.0          0               6  12.28    60                 0
4     65000.0          0              11  11.26    36                 0

   revol_util  total_rec_late_fee  od_ratio  bad_loan  …  purpose_other  \
0        43.2                 0.0  0.160624         0  …              0
1        21.5                 0.0  0.810777         0  …              0
2        46.3                 0.0  0.035147         1  …              0
3        10.7                 0.0  0.534887         1  …              0
4        15.2                 0.0  0.166500         0  …              0

   purpose_small_business  purpose_vacation  purpose_wedding  grade_B  \
0                       0                 0                0        0
1                       0                 0                0        0
2                       0                 0                0        0
3                       0                 0                0        0
4                       0                 0                0        0

   grade_C  grade_D  grade_E  grade_F  grade_G
0        0        0        0        0        0
1        0        0        0        0        0
```

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

[5 rows x 29 columns]
```

[25]:
```
# Check the distribution of the target variable 'bad_loan'
df["bad_loan"].value_counts()
```

[25]:
```
0    16000
1     4000
Name: bad_loan, dtype: int64
```

[27]:
```
# Separate the data into normal and fraud classes for oversampling
normal = df[df.bad_loan == 0]
fraud = df[df.bad_loan == 1]

# Calculate the difference in sample sizes between normal and fraud classes
sample_size_difference = len(normal) - len(fraud)

# Create additional samples from the fraud class
fraud_samples = fraud.sample(n=sample_size_difference, replace=True)

# Concatenate the original normal samples with the additional fraud samples
oversampled_df = pd.concat([normal, fraud_samples], axis=0)

# Shuffle the oversampled data to ensure randomness
oversampled_df = oversampled_df.sample(frac=1, random_state=42)
oversampled_df.head()
```

[27]:
```
       annual_inc  short_emp  emp_length_num    dti  term  last_delinq_none  \
9473      85000.0          0              11  15.45    60                 1
6261      80000.0          0              11  21.36    60                 0
3510      23000.0          0               4  26.40    36                 0
4925      45600.0          0               5  20.11    60                 1
11615    100360.0          0               6  17.90    60                 1

       revol_util  total_rec_late_fee  od_ratio  bad_loan  …  purpose_other  \
9473         85.4                 0.0  0.705981         1  …              0
6261         84.5                 0.0  0.361839         0  …              0
3510         48.8                 0.0  0.005252         1  …              0
4925         88.7                 0.0  0.772774         1  …              0
11615        37.5                 0.0  0.231019         1  …              0

       purpose_small_business  purpose_vacation  purpose_wedding  grade_B  \
9473                        0                 0                0        0
6261                        0                 0                0        0
```

```
3510                           0                  0                 0         0
4925                           0                  0                 0         0
11615                          0                  0                 0         0

       grade_C  grade_D  grade_E  grade_F  grade_G
9473         0        0        0        0        1
6261         0        0        1        0        0
3510         1        0        0        0        0
4925         1        0        0        0        0
11615        1        0        0        0        0

[5 rows x 29 columns]
```

[29]:
```python
# Check the distribution of the target variable after oversampling
oversampled_df["bad_loan"].value_counts()
```

[29]:
```
0    16000
1    12000
Name: bad_loan, dtype: int64
```

[31]:
```python
# Split the data into features (X) and target (y)
X = oversampled_df.drop('bad_loan', axis=1)
y = oversampled_df['bad_loan']

# Split the resampled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42, stratify=y)
```

[33]:
```python
# Check the shapes of the training and testing sets
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[33]:
```
((19600, 28), (8400, 28), (19600,), (8400,))
```

[35]:
```python
# Standardize numerical features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize and train a RandomForestClassifier model on the resampled data
rfc = RandomForestClassifier(random_state=52)
rfc.fit(X_train, y_train)

# Make predictions on the test set
y_pred = rfc.predict(X_test)
```

[36]:
```python
# Evaluate the model using appropriate metrics
accuracy = accuracy_score(y_test, y_pred)
```

```python
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print the evaluation metrics
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1-Score: {f1:.2f}')

# Print classification report and confusion matrix
print('Classification Report:\n', classification_report(y_test, y_pred))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.91
Precision: 0.89
Recall: 0.90
F1-Score: 0.89
Classification Report:
               precision    recall  f1-score   support

           0       0.92      0.91      0.92      4800
           1       0.89      0.90      0.89      3600

    accuracy                           0.91      8400
   macro avg       0.91      0.91      0.91      8400
weighted avg       0.91      0.91      0.91      8400

Confusion Matrix:
 [[4391  409]
 [ 369 3231]]
```