Credit Risk Resampling Techniques

In [30]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [31]:

```
import numpy as np
import pandas as pd
from pathlib import Path
from collections import Counter
```

Read the CSV and Perform Basic Data Cleaning

In [32]:

```
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_de
linq",
    "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", "n
um_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_bankrup
tcies",
    "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"]
```

In [33]:

```
# 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 valu
es
x = {'Current': 'low_risk'}
df = df.replace(x)

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

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

df.head()
```

Out[33]:

	loan_amnt	int_rate	installment	home_ownership	annual_inc	verification_status	issue_d	l
0	10500.0	0.1719	375.35	RENT	66000.0	Source Verified	Mar-2019	
1	25000.0	0.2000	929.09	MORTGAGE	105000.0	Verified	Mar-2019	
2	20000.0	0.2000	529.88	MORTGAGE	56000.0	Verified	Mar-2019	
3	10000.0	0.1640	353.55	RENT	92000.0	Verified	Mar-2019	
4	22000.0	0.1474	520.39	MORTGAGE	52000.0	Not Verified	Mar-2019	

5 rows × 86 columns

Split the Data into Training and Testing

In [34]:

```
# Create our features
X = pd.get_dummies(df.drop(columns='loan_status', axis ='columns'))# YOUR CODE H
ERE
```

In [35]:

```
X.describe()
```

Out[35]:

	loan_amnt	int_rate	installment	annual_inc	dti	delinq_2yrs	i
count	68817.000000	68817.000000	68817.000000	6.881700e+04	68817.000000	68817.000000	
mean	16677.594562	0.127718	480.652863	8.821371e+04	21.778153	0.217766	
std	10277.348590	0.048130	288.062432	1.155800e+05	20.199244	0.718367	
min	1000.000000	0.060000	30.890000	4.000000e+01	0.000000	0.000000	
25%	9000.000000	0.088100	265.730000	5.000000e+04	13.890000	0.000000	
50%	15000.000000	0.118000	404.560000	7.300000e+04	19.760000	0.000000	
75%	24000.000000	0.155700	648.100000	1.040000e+05	26.660000	0.000000	
max	40000.000000	0.308400	1676.230000	8.797500e+06	999.000000	18.000000	

8 rows × 95 columns

In [36]:

```
# Create our target
y = df['loan_status']# YOUR CODE HERE
```

In [37]:

```
y.value_counts()
```

Out[37]:

```
low_risk     68470
high_risk      347
Name: loan_status, dtype: int64
```

In [38]:

```
# Create X_train, X_test, y_train, y_test
# YOUR CODE HERE
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test= train_test_split(X,
                                                    y,
                                                    random_state=1
                                                   )
X_train.shape
```

Out[38]:

```
(51612, 95)
```

In [39]:

```
Counter(y_train)
```

Out[39]:

```
Counter({'low_risk': 51366, 'high_risk': 246})
```

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

Naive Random Oversampling

In [40]:

```
# Resample the training data with the RandomOversampler
# YOUR CODE HERE
from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(random_state=1)
X_resampled, y_resampled = ros.fit_resample(X_train, y_train)
Counter(y_resampled)
```

Out[40]:

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

In [44]:

```
# Logistic regression using random oversampled data
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(solver='liblinear',random_state=1)
model.fit(X_resampled, y_resampled)
```

Out[44]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_interce
pt=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=1, solver='liblinear', tol=0.0001, v
erbose=0,
                   warm_start=False)
```

In [45]:

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

In [46]:

```
from sklearn.metrics import balanced_accuracy_score

balanced_accuracy_score(y_test, y_pred)
```

Out[46]:

```
0.7163908158823367
```

In [47]:

```
# Display the confusion matrix
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, y_pred)
```

Out[47]:

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

In [48]:

```
from imblearn.metrics import classification_report_imbalanced

print(classification_report_imbalanced(y_test, y_pred))
```

```
                   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
```

SMOTE Oversampling

In [50]:

```
# Resample the training data with SMOTE
# YOUR CODE HERE
from imblearn.over_sampling import SMOTE
X_resampled, y_resampled = SMOTE(random_state=1, sampling_strategy=1.0).fit_resa
mple(
    X_train, y_train
)
from collections import Counter
Counter(y_resampled)
```

Out[50]:

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

In [74]:

```
# Train the Logistic Regression model using the resampled data
# YOUR CODE HERE
model = LogisticRegression(solver = 'liblinear',random_state=1)
model.fit(X_resampled, y_resampled)
```

Out[74]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_interce
pt=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=1, solver='liblinear', tol=0.0001, v
erbose=0,
                   warm_start=False)
```

In [75]:

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

In [76]:

```
# Calculated the balanced accuracy score
# YOUR CODE HERE
balanced_accuracy_score(y_test, y_pred)
```

Out[76]:

```
0.7107375728218286
```

In [77]:

```
# Display the confusion matrix
# YOUR CODE HERE
confusion_matrix(y_test, y_pred)
```

Out[77]:

```
array([[   69,    32],
       [ 4476, 12628]])
```

In [78]:

```
# Print the imbalanced classification report
# YOUR CODE HERE
print(classification_report_imbalanced(y_test, y_pred))
```

```
                   pre       rec       spe        f1       geo
iba       sup

  high_risk       0.02      0.68      0.74      0.03      0.71
0.50       101
   low_risk       1.00      0.74      0.68      0.85      0.71
0.51     17104

avg / total       0.99      0.74      0.68      0.84      0.71
0.51     17205
```

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

In [79]:

```
# Resample the data using the ClusterCentroids resampler
# YOUR CODE HERE
from imblearn.under_sampling import ClusterCentroids

cc = ClusterCentroids(random_state=1)
X_resampled, y_resampled = cc.fit_resample(X_train, y_train)

from collections import Counter

Counter(y_resampled)
```

Out[79]:

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

In [91]:

```
# Train the Logistic Regression model using the resampled data
# YOUR CODE HERE
model = LogisticRegression(solver='liblinear',random_state=1)
model.fit(X_resampled, y_resampled)
```

Out[91]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_interce
pt=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=1, solver='liblinear', tol=0.0001, v
erbose=0,
                   warm_start=False)
```

In [92]:

```
y_pred = model.predict(X_test)
balanced_accuracy_score(y_test, y_pred)
```

Out[92]:

```
0.6564285813520547
```

In [93]:

```
# Display the confusion matrix
# YOUR CODE HERE
confusion_matrix(y_test, y_pred)
```

Out[93]:

```
array([[  84,   17],
       [8874, 8230]])
```

In [94]:

```
# Print the imbalanced classification report
# YOUR CODE HERE
print(classification_report_imbalanced(y_test, y_pred))
```

```
                   pre       rec       spe        f1       geo
iba       sup

  high_risk       0.01      0.83      0.48      0.02      0.63
0.41       101
   low_risk       1.00      0.48      0.83      0.65      0.63
0.39     17104

avg / total       0.99      0.48      0.83      0.65      0.63
0.39     17205
```

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

In [100]:

```
# Resample the training data with SMOTEENN
# YOUR CODE HERE
from imblearn.combine import SMOTEENN

smote_enn = SMOTEENN(random_state=0)
X_resampled, y_resampled = smote_enn.fit_resample(X, y)
Counter(y_resampled)
```

Out[100]:

```
Counter({'high_risk': 68460, 'low_risk': 62011})
```

In [101]:

```
# Train the Logistic Regression model using the resampled data
# YOUR CODE HERE
model = LogisticRegression(solver='liblinear',random_state=1)
model.fit(X_resampled, y_resampled)
```

Out[101]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_interce
pt=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=1, solver='liblinear', tol=0.0001, v
erbose=0,
                   warm_start=False)
```

In [102]:

```
# Calculated the balanced accuracy score
# YOUR CODE HERE
y_pred = model.predict(X_test)
balanced_accuracy_score(y_test, y_pred)
```

Out[102]:

```
0.7317583635117487
```

In [103]:

```
# Display the confusion matrix
# YOUR CODE HERE
confusion_matrix(y_test, y_pred)
```

Out[103]:

```
array([[   75,    26],
       [ 4773, 12331]])
```

In [104]:

```
# Print the imbalanced classification report
# YOUR CODE HERE
print(classification_report_imbalanced(y_test, y_pred))
```

```
                   pre       rec       spe        f1       geo
iba       sup

  high_risk       0.02      0.74      0.72      0.03      0.73
0.54       101
   low_risk       1.00      0.72      0.74      0.84      0.73
0.53     17104

avg / total       0.99      0.72      0.74      0.83      0.73
0.53     17205
```