Week 4 Homework

In [19]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import imblearn # imbalance library for oversamplinga and undersampling
%matplotlib inline
```

In [20]:
```
# importing data
url = "https://github.com/JWarmenhoven/ISLR-python/blob/master/Notebooks/Data/Default.xlsx?raw=true"
df = pd.read_excel(url, index_col=0)
df.head()
```

Out[20]:

	default	student	balance	income
1	No	No	729.526495	44361.625074
2	No	Yes	817.180407	12106.134700
3	No	No	1073.549164	31767.138947
4	No	No	529.250605	35704.493935
5	No	No	785.655883	38463.495879

In [9]:
```
#Convert Yes and No to 1 and 0
def convert_yes2no(x):
    if x == "Yes":
        return 1
    elif x == "No":
        return 0
    else:
        return x
df['default'] = df['default'].apply(convert_yes2no)
df['student'] = df['student'].apply(convert_yes2no)
df.head()
```

Out[9]:

	default	student	balance	income
1	0	0	729.526495	44361.625074
2	0	1	817.180407	12106.134700
3	0	0	1073.549164	31767.138947
4	0	0	529.250605	35704.493935
5	0	0	785.655883	38463.495879

1. statified sampling

In [10]:
```
# split data into test, train and validate
df_train, df_validate, df_test = np.split(df, [int(.6*len(df)),int(.8*len(df))])
print(df_train['default'].value_counts())
print(df_validate['default'].value_counts())
print(df_test['default'].value_counts())
```

```
0    5801
1     199
Name: default, dtype: int64
0    1933
1      67
Name: default, dtype: int64
0    1933
1      67
Name: default, dtype: int64
```

2. Augment data with oversampling

In [12]:
```
#Random OverSampling
from collections import Counter
from sklearn.datasets import make_classification
from imblearn.over_sampling import RandomOverSampler
```

In [13]:
```
#over sampling Strategy
oversample = RandomOverSampler(sampling_strategy = 'minority')
#df_train['student','balance','income'],df_train_over = make_classification(n_samples=6000, n_features = 3, n_informative = 1, n_classes = 2, n_clusters_per_class = 1)
X = df_train[['income','balance','student']]
y = df_train['default']

#fit and apply the transform
X_over, y_over = oversample.fit_resample(X,y)
```

In [22]:
```
#convering to a single dataframe
df_train_X_over = pd.DataFrame(data = X_over, columns = ['income','balance','student'])
df_train_y_over =  pd.DataFrame(data = y_over, columns = ['default'])
df_train_over = df_train_X_over.assign(default = df_train_y_over)
df_train_over.head()
```

Out[22]:

	income	balance	student	default
0	44361.625074	729.526495	0.0	0
1	12106.134700	817.180407	1.0	0
2	31767.138947	1073.549164	0.0	0
3	35704.493935	529.250605	0.0	0
4	38463.495879	785.655883	0.0	0

3. Augment data with undersampling

In [23]:
```
# under sampling Strategy
from imblearn.under_sampling import RandomUnderSampler
undersample = RandomUnderSampler(sampling_strategy = 'majority')
X = df_train[['income','balance','student']]
y = df_train['default']

#fit and apply the transform
X_under, y_under = undersample.fit_resample(X,y)
```

In [24]:
```
# convering to asinlge dataframe
df_train_X_under = pd.DataFrame(data = X_under, columns = ['income','balance','student'])
df_train_y_under =  pd.DataFrame(data = y_under, columns = ['default'])
df_train_under = df_train_X_under.assign(default = df_train_y_under)
df_train_under.head()
```

Out[24]:

	income	balance	student	default
0	28521.366885	4.712142	0.0	0
1	54335.224100	569.790873	0.0	0
2	30067.410647	1314.765205	0.0	0
3	37074.061367	1116.065771	0.0	0
4	19318.059334	703.838766	1.0	0

4. Training data

- df_train

In [27]:
```
# Use sklearn
from sklearn.linear_model import LogisticRegression
import warnings
warnings.filterwarnings('ignore')
clf_train = LogisticRegression(penalty='none',solver='newton-cg')
clf_train.fit(df_train[['balance', 'income', 'student']],df_train['default'])
print(clf_train)
print('classes: ',clf_train.classes_)
print('coefficients: ',clf_train.coef_)
print('intercept :', clf_train.intercept_)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='none',
                   random_state=None, solver='newton-cg', tol=0.0001, verbose=0,
                   warm_start=False)
classes:  [0 1]
coefficients:  [[ 5.73512809e-03  5.34865851e-06 -6.19140023e-01]]
intercept : [-10.91078838]
```

- df_train_over

In [29]:
```
clf_train_over = LogisticRegression(penalty='none',solver='newton-cg')
clf_train_over.fit(df_train_over[['balance', 'income', 'student']],df_train_over['default'])
print(clf_train_over)
print('classes: ',clf_train_over.classes_)
print('coefficients: ',clf_train_over.coef_)
print('intercept :', clf_train_over.intercept_)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='none',
                   random_state=None, solver='newton-cg', tol=0.0001, verbose=0,
                   warm_start=False)
classes:  [0 1]
coefficients:  [[ 5.76588813e-03 -1.13679197e-07 -7.64075707e-01]]
intercept : [-7.33354857]
```

- df_train_under

In [28]:
```
clf_train_under = LogisticRegression(penalty='none',solver='newton-cg')
clf_train_under.fit(df_train_under[['balance', 'income', 'student']],df_train_under['default'])
print(clf_train_under)
print('classes: ',clf_train_under.classes_)
print('coefficients: ',clf_train_under.coef_)
print('intercept :', clf_train_under.intercept_)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='none',
                   random_state=None, solver='newton-cg', tol=0.0001, verbose=0,
                   warm_start=False)
classes:  [0 1]
coefficients:  [[ 5.67319905e-03 -1.19201105e-05 -9.21789201e-01]]
intercept : [-6.86116909]
```

5. confusion matrix, precesion score, and recall score on validate data

In [30]:
```
# validating data
y_predict_train = clf_train.predict(df_validate[['balance', 'income', 'student']])
y_predict_over = clf_train_over.predict(df_validate[['balance', 'income', 'student']])
y_predict_under = clf_train_under.predict(df_validate[['balance', 'income', 'student']])
```

In [31]:
```
print(np.count_nonzero(y_predict_train))
print(np.count_nonzero(y_predict_over))
print(np.count_nonzero(y_under))
```

```
26
342
199
```

- confusion matrix, precesion score, and recall score for df_train

In [33]:
```
#
y_validate = df_validate['default'].values
TP = 0
FP = 0
TN = 0
FN = 0

for i in range(len(y_predict_train)):
    if y_validate[i]==y_predict_train[i]==1:
          TP += 1
    if y_predict_train[i]==1 and y_validate[i]!=y_predict_train[i]:
          FP += 1
    if y_validate[i]==y_predict_train[i]==0:
          TN += 1
    if y_predict_train[i]==0 and y_validate[i]!=y_predict_train[i]:
          FN += 1

print('Confusion matrix elements are', TP,FP,TN,FN)
#precision score
PPV = TP/(TP + FP )
print('precesion score is', PPV)
#recall score
TPR = TP/(TP + FN)
print('recall score is', TPR)
```

```
Confusion matrix elements are 18 8 1925 49
precesion score is 0.6923076923076923
recall score is 0.26865671641791045
```

- confusion matrix, precesion score, and recall score for df_train_over

In [38]:
```
#
TP = 0
FP = 0
TN = 0
FN = 0

for i in range(len(y_predict_train)):
    if y_validate[i]==y_predict_train[i]==1:
          TP += 1
    if y_predict_over[i]==1 and y_validate[i]!=y_predict_over[i]:
          FP += 1
    if y_validate[i]==y_predict_over[i]==0:
          TN += 1
    if y_predict_over[i]==0 and y_validate[i]!=y_predict_over[i]:
          FN += 1

print('confusion matrix elements are ',TP,FP,TN,FN)
#precision score
PPV = TP/(TP + FP )
print('precesion score is', PPV)
#recall score
TPR = TP/(TP + FN)
print('recall score is', TPR)
```

```
confusion matrix elements are  18 281 1652 6
precesion score is 0.06020066889632107
recall score is 0.75
```

- confusion matrix, precesion score, and recall score for df_train_under

In [39]:
```
TP = 0
FP = 0
TN = 0
FN = 0

for i in range(len(y_predict_train)):
    if y_validate[i]==y_predict_train[i]==1:
          TP += 1
    if y_predict_under[i]==1 and y_validate[i]!=y_predict_under[i]:
          FP += 1
    if y_validate[i]==y_predict_under[i]==0:
          TN += 1
    if y_predict_under[i]==0 and y_validate[i]!=y_predict_under[i]:
          FN += 1

print('confusion matrix elements are',TP,FP,TN,FN)
#precision score
PPV = TP/(TP + FP )
print('precesion score is', PPV)
#recall score
TPR = TP/(TP + FN)
print('recall score is', TPR)
```

```
confusion matrix elements are 18 261 1672 6
precesion score is 0.06451612903225806
recall score is 0.75
```

Chosing Best Model:

```
In machinc learning, Confusion matric basically give visulization of error in supervised learning. Here, we see that precesion score is very low and recall score is very high for both 'df_train_under' and 'df_train_over'. While precison score camparatively high and recall score is low for 'df_train'. Oversmapling and undersampling involve an introduction of a bias to select data from one class over another, to compensate the imbalance present in the data. The precesion score and recall score show that 'logistic regression' donot have any benefits from balancing data, whi
ch might be good for other machine learning techniques like 'neural network' training.
So the best model here is sratified random sampling.
```

6. Applying best model to calculate confusion matrix, precesion score, and recall score on test data

In [37]:
```
#validating data
y_predict_test = clf_train.predict(df_test[['balance', 'income', 'student']])
y_test = df_test['default'].values
```

In [71]:
```
TP = 0
FP = 0
TN = 0
FN = 0

for i in range(len(y_predict_test)):
    if y_test[i]==y_predict_test[i]==1:
          TP += 1
    if y_predict_test[i]==1 and y_test[i]!=y_predict_test[i]:
          FP += 1
    if y_test[i]==y_predict_test[i]==0:
          TN += 1
    if y_predict_test[i]==0 and y_test[i]!=y_predict_test[i]:
          FN += 1

print(confudiTP,FP,TN,FN)
#precision score
PPV = TP/(TP + FP )
print('precesion score is', PPV)
#recall score
TPR = TP/(TP + FN)
print('recall score is', TPR)
```

```
24 10 1923 43
precesion score is 0.7058823529411765
recall score is 0.3582089552238806
```

```
The precesion score and recall scores for 'df_test' are close to the 'df_validate' which make sense becasue these data sets are obtained by randomly stratified sampdling.
```

In [ ]: