Model presented in the previous project

Below is the logistic regression model presented previously in `project.pdf/project.ipynb` . As discussed before, when predicting credit risk, it might be desirable to have a high recall rate for the 'bad' case (y=1). However, the prototype logistic regression presented earlier only have a recall = 0.44, as shown in classification report below.

In [18]:
```
import pandas as pd
```

In [63]:
```
df = pd.read_csv("Tenzing Assesment Data Set.csv")
```

In [64]:
```
y = df['class'].replace('good',0).replace('bad',1).values
X = df.drop(['class'], axis = 1)
X = pd.get_dummies(X, drop_first=True).values
```

In [65]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s

# Fit a logistic regression model to our data
prototype_model = LogisticRegression(solver = 'lbfgs', max_iter = 500)
prototype_model.fit(X_train, y_train)

# Obtain model predictions
predicted = prototype_model.predict(X_test)

# Print the classifcation report and confusion matrix
print('Classification report:\n', classification_report(y_test, predicted))
conf_mat = confusion_matrix(y_true=y_test, y_pred=predicted)
print('Confusion matrix:\n', conf_mat)
```

```
Classification report:
               precision    recall  f1-score   support

           0       0.79      0.90      0.84       210
           1       0.65      0.44      0.53        90

   micro avg       0.76      0.76      0.76       300
   macro avg       0.72      0.67      0.68       300
weighted avg       0.75      0.76      0.75       300

Confusion matrix:
 [[188  22]
 [ 50  40]]
```

Improve model performance by oversampling

minority

- As we discussed in previous project, the main reason that causes the low recall rate might be the imbalance of the data set. So we will first try to re-sample the data to achieve the data balance.
- Note we should re-sample only after we do the test and train split.

In [88]:
```
from sklearn.utils import resample

y = df['class'].replace('good',0).replace('bad',1)
X = df.drop(['class'], axis = 1)
X = pd.get_dummies(X, drop_first=True)

# setting up testing and training sets. Note same parameters are set as compared
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s

# concatenate our training data back together
X = pd.concat([X_train, y_train], axis=1)
```

In [89]:
```
# separate minority and majority classes
good = X[X['class']==0]
bad = X[X['class']==1]

# upsample minority
credit_upsampled = resample(bad,
                          replace=True, # sample with replacement
                          n_samples=len(good), # match number in majority class
                          random_state=27) # reproducible results

# combine majority and upsampled minority
upsampled = pd.concat([good, credit_upsampled])

# check new class counts
upsampled['class'].value_counts()
```

Out[89]:
```
1    490
0    490
Name: class, dtype: int64
```

In [90]:
```
# trying logistic regression again with the balanced dataset
y_train = upsampled['class']
X_train = upsampled.drop('class', axis=1)

upsampled = LogisticRegression(solver='liblinear').fit(X_train, y_train)

upsampled_pred = upsampled.predict(X_test)
```

In [93]:
```python
# confusion matrix
pd.DataFrame(confusion_matrix(y_test, upsampled_pred))
```

Out[93]:

	0	1
0	150	60
1	28	62

In [95]:
```python
recall_score(y_test, upsampled_pred)
```

Out[95]:
```
0.6888888888888889
```

Summary

- We see that up-sampling of the minority class really helps increase the recall rate a lot for 'bad' class, **from 0.44 to 0.69.**
- We may also try down-sampling of the majority class and see how the recall rate changes. Also, we may try SMOTE, as mentioned in the previous project.
- Note the increase of recall rate for 'bad' class must be at the cost of reducing other metrics such as f1 score, recall rate for 'good' case. We may choose to have a better recall/precision for either 'good' or 'bad', depending on specific business requirement.
- Furthermore, we may also combine re-sampling with nonlinear algorithms such as random forest, neural network, support vector machine, etc. In summary, there should be a big room to improve.