Credit Card Fraud Detection - Logistic Regression_Mar 16

March 17, 2022

In [26]:
```
import numpy as np
# working with arrays, has functionsin domain of linear algebra, fourier transform, and
import pandas as pd
# pd is used for data processing
import matplotlib.pyplot as plt
#plt provides an implicit, MATLAB-like, way of plotting
import seaborn as sns
#sns is a data visualization library based on matplotlib

data=pd.read_csv('/Users/eden_zoo/Desktop/Certificate in Data Analytics/CIND820 Capston
```

In [27]:
```
#Histograms above show that there is very few outlier compared to the data size
#Histograms above show that in majority of attributes, the fraud distribution line fitt

#Looking at the histograms, V1-V28 all seem scaled, but not amount and time. So the nex

from sklearn.preprocessing import RobustScaler

rbst_scaler=RobustScaler() # robustscaler is less prone to outliers

data['scaled_time']=rbst_scaler.fit_transform(data['Time'].values.reshape(-1,1))
data['scaled_amount']=rbst_scaler.fit_transform(data['Amount'].values.reshape(-1,1))

data.drop(['Time','Amount'],axis=1,inplace=True)

scaled_time=data['scaled_time']
scaled_amount=data['scaled_amount']

data.drop(['scaled_time','scaled_amount'],axis=1,inplace=True)
data.insert(0,'scaled_amount',scaled_amount)
data.insert(1,'scaled_time',scaled_time)
```

1 Data split and modeling

In [28]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
#from sklearn.model_selection import StratifiedShuffleSplit
```

```
from sklearn.model_selection import StratifiedKFold

from imblearn.over_sampling import SMOTE
#from imblearn.under_sampling import NearMiss
from imblearn.under_sampling import RandomUnderSampler
#from imblearn.pipeline import make_pipeline

from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import f1_score
# import warnings
# warnings.filterwarnings("ignore")
```

In [29]:
```
#axis=1 drops labels from columns.
X = data.drop('Class', axis=1)
y = data['Class']
```

In [30]:
```
#5-fold stratified split for cross-validation
sss = StratifiedKFold(n_splits=5, random_state=42, shuffle=True)
```

2 Undersampling the majority with RUS - minority:majority = 1:1

In [31]:
```
# Hyperparameters for logistic regression
Cs=[0.001, 0.01, 0.1, 1, 10, 100, 1000] #=np.logspace(-3,3,7)
penalty = ["l1","l2"]
solver = ['saga', 'liblinear'] #only two optimizers work for both l1 and l2 regularizat

rus = RandomUnderSampler(random_state=42)

# hyperparameter tuning
best_model={'C':-1,'penalty':"-1", 'solver':'unknown'}
best_result=0

for train_index, test_index in sss.split(X, y):
    original_Xtrain, original_Xtest = X.iloc[train_index], X.iloc[test_index]
    original_ytrain, original_ytest = y.iloc[train_index], y.iloc[test_index]

    X_train_sampled, y_train_sampled = rus.fit_resample(original_Xtrain, original_ytrai

    for c in Cs:
        for p in penalty:
            for s in solver:
                clf = LogisticRegression(random_state=0,C=c,penalty=p,solver=s,max_iter
                results=clf.predict(original_Xtest)
                f1=f1_score(original_ytest,results)
                if f1 > best_result:
                    best_result=f1
                    best_model['C']=c
                    best_model['penalty']=p
```

```
                        best_model['solver']=s
        # 'liblinear' and 'saga' both handle L1 penalty. 'liblinear' is good for small data
        print(best_model)

        clf = LogisticRegression(random_state=0,C=best_model['C'],penalty=best_model['penal
        results=clf.predict(original_Xtest)
        re=precision_recall_fscore_support(original_ytest, results, average='macro')
        precision=re[0]
        recall=re[1]
        fscore=re[2]
        #f1=f1_score(original_ytest,results)
        print("precision={}, recall={}, f1={}".format(precision,recall,fscore))
    #     clf = LogisticRegression(random_state=0,C=best_model['C'],penalty=best_model['pen
    #     results=clf.predict(original_Xtest)
    #     re=precision_recall_fscore_support(original_ytest, results, average='macro')
    #     precision=re[0]
    #     recall=re[1]
    #     fscore=re[2]
    #     #f1=f1_score(original_ytest,results)
    #     print("precision={}, recall={}, f1={}".format(precision,recall,fscore))
```

```
{'C': 0.001, 'penalty': 'l2', 'solver': 'saga'}
precision=0.6940073740973967, recall=0.9029324779724863, f1=0.7616569844659149
{'C': 0.001, 'penalty': 'l2', 'solver': 'saga'}
precision=0.8305767004777609, recall=0.9238643224890872, f1=0.871426354274645
{'C': 0.001, 'penalty': 'l2', 'solver': 'saga'}
precision=0.8027877215446131, recall=0.9229945622974232, f1=0.8528880245856063
{'C': 0.001, 'penalty': 'l2', 'solver': 'saga'}
precision=0.6264348647610047, recall=0.9010124405705514, f1=0.6915729120290709
{'C': 0.001, 'penalty': 'l2', 'solver': 'saga'}
precision=0.7038391551754967, recall=0.9173209723190756, f1=0.7736534217761801
```

3 Undersampling the majority with RUS - minority:majority = 1:9

In [ ]:
```
Cs=[0.001, 0.01, 0.1, 1, 10, 100, 1000] #=np.logspace(-3,3,7)
penalty = ["l1","l2"]
solver = ['saga', 'liblinear'] #only two optimizers work for both l1 and l2 regularizati

rus = RandomUnderSampler(sampling_strategy=0.1,random_state=42)

# hyperparameter tuning
best_model={'C':-1,'penalty':"-1", 'solver':'unknown'}
best_result=0

for train_index, test_index in sss.split(X, y):
    original_Xtrain, original_Xtest = X.iloc[train_index], X.iloc[test_index]
```

```
        original_ytrain, original_ytest = y.iloc[train_index], y.iloc[test_index]

        X_train_sampled, y_train_sampled = rus.fit_resample(original_Xtrain, original_ytrain

        for c in Cs:
            for p in penalty:
                for s in solver:
                    clf = LogisticRegression(random_state=0,C=c,penalty=p,solver=s,max_iter=
                    results=clf.predict(original_Xtest)
                    f1=f1_score(original_ytest,results)
                    if f1 > best_result:
                        best_result=f1
                        best_model['C']=c
                        best_model['penalty']=p
                        best_model['solver']=s
        # 'liblinear' and 'saga' both handle L1 penalty. 'liblinear' is good for small datae
        print(best_model)

        clf = LogisticRegression(random_state=0,C=best_model['C'],penalty=best_model['penalt
        results=clf.predict(original_Xtest)
        re=precision_recall_fscore_support(original_ytest, results, average='macro')
        precision=re[0]
        recall=re[1]
        fscore=re[2]
        #f1=f1_score(original_ytest,results)
        print("precision={}, recall={}, f1={}".format(precision,recall,fscore))
```

```
{'C': 0.001, 'penalty': 'l2', 'solver': 'saga'}
precision=0.8772575227769617, recall=0.8886690622881116, f1=0.8828779346586362
{'C': 0.001, 'penalty': 'l2', 'solver': 'saga'}
precision=0.8892647878520579, recall=0.9088886686181941, f1=0.8988345125870449
```

4 Combine undersampling (RUS) with oversampling (SMOTE) - minority:majority = 1:1

In [ ]:
```
from imblearn.pipeline import Pipeline
from collections import Counter

over = SMOTE(sampling_strategy=0.05,random_state=42)
under = RandomUnderSampler(sampling_strategy=1,random_state=42)
steps = [('o', over), ('u', under)]
pipeline = Pipeline(steps=steps)

for train_index, test_index in sss.split(X, y):
    original_Xtrain, original_Xtest = X.iloc[train_index], X.iloc[test_index]
    original_ytrain, original_ytest = y.iloc[train_index], y.iloc[test_index]
```

```
# transform the dataset
    X_train_sampled, y_train_sampled = pipeline.fit_resample(original_Xtrain, original_y

    counter = Counter(y_train_sampled)
    print(counter)

    clf = LogisticRegression(random_state=0,C=best_model['C'],penalty=best_model['penalt
    results=clf.predict(original_Xtest)
    re=precision_recall_fscore_support(original_ytest, results, average='macro')
    precision=re[0]
    recall=re[1]
    fscore=re[2]
    print("precision={}, recall={}, f1={}".format(precision,recall,fscore))
```