Credit Card Fraud Detection - Random Forest_Mar 16

March 17, 2022

In [ ]:
```
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 Capstone
```

In [ ]:
```
#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 fitte

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

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 [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

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

```
In [ ]: #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 [ ]: http://localhost:8888/notebooks/Desktop/Certificate%20in%20Data%20Analytics/CIND820%20Ca
        from sklearn.model_selection import RepeatedStratifiedKFold
        from sklearn.model_selection import GridSearchCV
        from sklearn.ensemble import RandomForestClassifier

        # # summarize results
        # print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
        # means = grid_result.cv_results_['mean_test_score']
        # stds = grid_result.cv_results_['std_test_score']
        # params = grid_result.cv_results_['params']
        # for mean, stdev, param in zip(means, stds, params):
        # print("%f (%f) with: %r" % (mean, stdev, param))

        rus = RandomUnderSampler(random_state=42)

        # define hyperparameters
        n_estimators = [10, 50, 100]
        max_features = ['sqrt', 'log2']
        class_weight =[None, 'balanced', 'balanced_subsample']

        # hyperparameter tuning
        best_model={'n_estimators':-1,'max_features':"-1", 'class_weight': "-1"}
        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

    counter = Counter(y_train_sampled)
    print(counter)

    for n in n_estimators:
        for m_f in max_features:
            for c_w in class_weight:
                clf = RandomForestClassifier(random_state=0,n_estimators=n,max_features=
                results=clf.predict(original_Xtest)
                f1=f1_score(original_ytest,results)
                if f1 > best_result:
                    best_result=f1
                    best_model['n_estimators']=n
                    best_model['max_features']=m_f
                    best_model['class_weight']=c_w
    # 'liblinear' and 'saga' both handle L1 penalty. 'liblinear' is good for small datae
    print(best_model)

    clf = RandomForestClassifier(random_state=0,n_estimators=best_model['n_estimators'],
    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['pena
#     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))
```

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

```
In [ ]: from collections import Counter

        rus = RandomUnderSampler(sampling_strategy=0.11,random_state=42)

        # define hyperparameters
```

```
n_estimators = [10, 50, 100]
max_features = ['sqrt', 'log2']
class_weight =[None, 'balanced', 'balanced_subsample']

# hyperparameter tuning
best_model={'n_estimators':-1,'max_features':"-1", 'class_weight': "-1"}
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

    counter = Counter(y_train_sampled)
    print(counter)

    for n in n_estimators:
        for m_f in max_features:
            for c_w in class_weight:
                clf = RandomForestClassifier(random_state=0,n_estimators=n,max_features=
                results=clf.predict(original_Xtest)
                f1=f1_score(original_ytest,results)
                if f1 > best_result:
                    best_result=f1
                    best_model['n_estimators']=n
                    best_model['max_features']=m_f
                    best_model['class_weight']=c_w
    # 'liblinear' and 'saga' both handle L1 penalty. 'liblinear' is good for small datae
    print(best_model)

    clf = RandomForestClassifier(random_state=0,n_estimators=best_model['n_estimators'],
    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))
```

In [16]:
```
#     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))
```

4 Oversampling the minority with SMOTE - minority:majority = 1:1

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

        oversampler_smote = SMOTE(random_state=42)

        # define hyperparameters
        n_estimators = [10, 50, 100]
        max_features = ['sqrt', 'log2']
        class_weight =[None, 'balanced', 'balanced_subsample']

        # hyperparameter tuning
        best_model={'n_estimators':-1,'max_features':"-1", 'class_weight': "-1"}
        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 = oversampler_smote.fit_resample(original_Xtrain, o

            counter = Counter(y_train_sampled)
            print(counter)

            for n in n_estimators:
                for m_f in max_features:
                    for c_w in class_weight:
                        clf = RandomForestClassifier(random_state=0,n_estimators=n,max_features=
                        results=clf.predict(original_Xtest)
                        f1=f1_score(original_ytest,results)
                        if f1 > best_result:
                            best_result=f1
                            best_model['n_estimators']=n
                            best_model['max_features']=m_f
                            best_model['class_weight']=c_w
            # 'liblinear' and 'saga' both handle L1 penalty. 'liblinear' is good for small datae
            print(best_model)

            clf = RandomForestClassifier(random_state=0,n_estimators=best_model['n_estimators'],
            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))
```

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

```
In [ ]: from sklearn.ensemble import RandomForestClassifier
        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)

        # define hyperparameters
        n_estimators = [10, 50, 100]
        max_features = ['sqrt', 'log2']
        class_weight =[None, 'balanced', 'balanced_subsample']

        # hyperparameter tuning
        best_model={'n_estimators':-1,'max_features':"-1", 'class_weight': "-1"}
        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 = pipeline.fit_resample(original_Xtrain, original_y

            counter = Counter(y_train_sampled)
            print(counter)

            for n in n_estimators:
                for m_f in max_features:
                    for c_w in class_weight:
                        clf = RandomForestClassifier(random_state=0,n_estimators=n,max_features=
                        results=clf.predict(original_Xtest)
                        f1=f1_score(original_ytest,results)
                        if f1 > best_result:
                            best_result=f1
                            best_model['n_estimators']=n
                            best_model['max_features']=m_f
                            best_model['class_weight']=c_w
            # 'liblinear' and 'saga' both handle L1 penalty. 'liblinear' is good for small datae
            print(best_model)
```

```python
clf = RandomForestClassifier(random_state=0,n_estimators=best_model['n_estimators'],
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))
```