Resampling - Imbalanced dataset classification

September 13, 2020

0.1 Aim:

Resample imbalanced dataset for the credit card transaction dataset(classify credict card transaction as fraud or not).

0.2 Source:

https://www.kaggle.com/janiobachmann/credit-fraud-dealing-with-imbalanced-datasets/notebook https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets

0.3 About the data:

1. Except for the transaction and amount we dont know what the other columns are (due to privacy reasons). The only thing we know, is that those columns that are unknown have been scaled already.
2. The description of the data says that all the features went through a PCA transformation (Dimensionality Reduction technique) (Except for time and amount).
3. Keep in mind that in order to implement a PCA transformation features need to be previously scaled. (In this case, all the V features have been scaled or at least that is what we are assuming the people that develop the dataset did.)

0.4 The metric trap:

One of the major issues that novice users fall into when dealing with unbalanced datasets relates to the metrics used to evaluate their model. Using simpler metrics like accuracy_score can be misleading. In a dataset with highly unbalanced classes, if the classifier always "predicts" the most common class without performing any analysis of the features, it will still have a high accuracy rate, obviously illusory.

0.5 Confusion matrix:

An interesting way to evaluate the results is by means of a confusion matrix, which shows the correct and incorrect predictions for each class.

0.6 Importing data

[1]:
```
import pandas as pd
import numpy as np
```

[2]:
```
df=pd.read_csv(r'C:\Users\aasth\Desktop\Data analytics\Data Science\Machine␣
↪Learning\ML algos - Github\creditcard.csv')
```

[3]:
```
df.head()
```

[3]:
```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  …       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  … -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  … -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  …  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  … -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  … -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
```

[4]:
```
df.describe()
```

[4]:
```
                Time            V1            V2            V3            V4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean    94813.859575  3.919560e-15  5.688174e-16 -8.769071e-15  2.782312e-15
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01

                 V5            V6            V7            V8            V9  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean  -1.552563e-15  2.010663e-15 -1.694249e-15 -1.927028e-16 -3.137024e-15
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02
```

```
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01

       …           V21           V22           V23           V24  \
count  …  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   …  1.537294e-16  7.959909e-16  5.367590e-16  4.458112e-15
std    …  7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min    … -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%    … -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%    … -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%    …  1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max    …  2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                V25           V26           V27           V28         Amount  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  284807.000000
mean   1.453003e-15  1.699104e-15 -3.660161e-16 -1.206049e-16      88.349619
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01     250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01       0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02       5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02      22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02      77.165000
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01   25691.160000

              Class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 31 columns]
```

[5]:
```
df.Class.value_counts(normalize=True)
```

[5]:
```
0    0.998273
1    0.001727
Name: Class, dtype: float64
```

99% of the rows belong to class 0 and only 1% belong to class 1. This is highly imbalanced dataset

0.7 Distribution of Amount and Time class

[6]:
```
import plotly.express as px
fig=px.histogram(df,x="Amount")
fig.update_layout(title_text="Distribution of Amount")
```

Tail heavy distribution Scaling and transformation is neeeded

[7]:
```
fig=px.histogram(df,x="Time")
fig.update_layout(title_text="Distribution of Time")
```

Scaling is required

0.8 Missing values

[8]:
```
df.isnull().sum()
```

[8]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
```

```
dtype: int64
```

0.9 Scaling

[3]:
```
#columns that are left to scale (Amount and Time)
from sklearn.preprocessing import RobustScaler

# RobustScaler is less prone to outliers.

rob_scaler = RobustScaler()

df['Amount']=rob_scaler.fit_transform(df['Amount'].values.reshape(-1,1))
df['Time']=rob_scaler.fit_transform(df['Time'].values.reshape(-1,1))
```

0.10 Train/Test Split

[4]:
```
df.columns
```

[4]:
```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

[5]:
```
from sklearn.model_selection import train_test_split

X=df.copy()
X.drop(['Class'],axis=1,inplace=True)
y=df['Class']

#stratify=data is split in a stratified fashion, using this as the class labels.

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,stratify=y)
```

[6]:
```
print("Shape of training features:",X_train.shape)
print("Shape of training labels:",y_train.shape)
print("Shape of testing features:",X_test.shape)
print("Shape of testing labels:",y_test.shape)
```

```
Shape of training features: (227845, 30)
Shape of training labels: (227845,)
Shape of testing features: (56962, 30)
Shape of testing labels: (56962,)
```

[7]:
```
len(y_train[y_train==1])
```

[7]: 394

[8]:
```
len(y_test[y_test==1])
```

[8]: 98

0.11 Resampling:

A widely adopted technique for dealing with highly unbalanced datasets is called resampling. It consists of removing samples from the majority class (under-sampling) and / or adding more examples from the minority class (over-sampling).

Despite the advantage of balancing classes, these techniques also have their weaknesses (there is no free lunch which basically says that there is no one algorithm that suits all types of data). The simplest implementation of over-sampling (random over sampling) is to duplicate random records from the minority class, which can cause overfitting. In under-sampling, the simplest technique (random undersampling) involves removing random records from the majority class, which can cause loss of information.

0.12 Python imbalanced-learn module:

A number of more sophisticated resampling techniques have been proposed in the scientific literature.

For example, we can cluster the records of the majority class, and do the under-sampling by removing records from each cluster, thus seeking to preserve information. In over-sampling, instead of creating exact copies of the minority class records, we can introduce small variations into those copies, creating more diverse synthetic samples.

You can also perform the basic implementation of random undersampling and random oversampling using the imbalanced learn module.

0.13 PCA for plotting

[15]:
```
from sklearn.decomposition import PCA
import plotly.express as px

pca = PCA(n_components=2)
X_train_PCA = pca.fit_transform(X_train)

px.scatter(X_train_PCA.T[0], X_train_PCA.T[1],color=y_train)
```

0.14 Undersampling

0.14.1 NearMiss:

NearMiss is an under-sampling technique. It aims to balance class distribution by randomly eliminating majority class examples. When instances of two different classes are very close to each other, we remove the instances of the majority class to increase the spaces between the two classes. This helps in the classification process.

[61]:
```
from imblearn.under_sampling import NearMiss
ns=NearMiss()
X_train_ns,y_train_ns=ns.fit_sample(X_train,y_train)


pca1 = PCA(n_components=2)
X_train_ns_PCA = pca1.fit_transform(X_train_ns)

px.scatter(X_train_ns_PCA.T[0], X_train_ns_PCA.T[1],color=y_train_ns)
```

[66]:
```
print("Shape before Resampling:",X_train.shape,y_train.shape)
print("Class 1:{} , Class 0:{}".
 ↪format(len(y_train[y_train==1]),len(y_train[y_train==0])))
print("Shape after NearMiss:",X_train_ns.shape,y_train_ns.shape)
print("Class 1:{} , Class 0:{}".
 ↪format(len(y_train_ns[y_train_ns==1]),len(y_train_ns[y_train_ns==0])))
```

```
Shape before Resampling: (227845, 30) (227845,)
Class 1:394 , Class 0:227451
Shape after NearMiss: (788, 30) (788,)
Class 1:394 , Class 0:394
```

0.14.2 Tomek links:

Tomek links are pairs of very close instances, but of opposite classes. Removing the instances of the majority class of each pair increases the space between the two classes, facilitating the classification process.

In the code below, we'll use ratio='majority' to resample the majority class.

[70]:
```
from imblearn.under_sampling import TomekLinks

tl = TomekLinks()
X_tl, y_tl = tl.fit_sample(X_train, y_train)
```

[71]:
```
pca2 = PCA(n_components=2)
X_train_tl_PCA = pca2.fit_transform(X_tl)

px.scatter(X_train_tl_PCA.T[0], X_train_tl_PCA.T[1],color=y_tl)
```

[72]:
```
print("Shape before Resampling:",X_train.shape,y_train.shape)
print("Class 1:{} , Class 0:{}".
 ↪format(len(y_train[y_train==1]),len(y_train[y_train==0])))
print("Shape after Tomek Links:",X_tl.shape,y_tl.shape)
print("Class 1:{} , Class 0:{}".format(len(y_tl[y_tl==1]),len(y_tl[y_tl==0])))
```

```
Shape before Resampling: (227845, 30) (227845,)
Class 1:394 , Class 0:227451
```

```
Shape after Tomek Links: (227820, 30) (227820,)
Class 1:394 , Class 0:227426
```

Source: https://machinelearningmastery.com/undersampling-algorithms-for-imbalanced-classification/

Because the procedure only removes so-named "Tomek Links", we would not expect the resulting transformed dataset to be balanced, only less ambiguous along the class boundary.

We can see that only 25 examples(22785-227820) from the majority class were removed.The scatter plot of the transformed dataset does not make the minor editing to the majority class obvious.

This highlights that although finding the ambiguous examples on the class boundary is useful, alone, it is not a great undersampling technique. In practice, the Tomek Links procedure is often combined with other methods.

There are many other techniques to undersampling and oversampling but we are looking only at a few of them. ## Oversampling

0.14.3 SMOTE

SMOTE (Synthetic Minority Oversampling TEchnique) consists of synthesizing elements for the minority class, based on those that already exist. It works by randomly picking a point from the minority class and computing the k-nearest neighbors for this point. The synthetic points are added between the chosen point and its neighbors.

At a high level, SMOTE creates synthetic observations of the minority class by:

1. Finding the k-nearest-neighbors for minority class observations (finding similar observations)
2. Randomly choosing one of the k-nearest-neighbors and using it to create a similar, but randomly tweaked, new observation.

[74]:
```
#class ratio of 1 will give us a balanced dataset
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=12, ratio = 1.0)
X_train_sm, y_train_sm = sm.fit_sample(X_train, y_train)
```

[75]:
```
pca3 = PCA(n_components=2)
X_train_sm_PCA = pca3.fit_transform(X_train_sm)

px.scatter(X_train_sm_PCA.T[0], X_train_sm_PCA.T[1],color=y_train_sm)
```

[76]:
```
print("Shape before Resampling:",X_train.shape,y_train.shape)
print("Class 1:{} , Class 0:{}".
 ↪format(len(y_train[y_train==1]),len(y_train[y_train==0])))
print("Shape after Tomek Links:",X_train_sm.shape,y_train_sm.shape)
print("Class 1:{} , Class 0:{}".
 ↪format(len(y_train_sm[y_train_sm==1]),len(y_train_sm[y_train_sm==0])))
```

```
Shape before Resampling: (227845, 30) (227845,)
Class 1:394 , Class 0:227451
```

```
Shape after Tomek Links: (454902, 30) (454902,)
Class 1:227451 , Class 0:227451
```

0.15 Undersampling and Oversampling combined

The combination of SMOTE and under-sampling performs better than plain under-sampling.There are combinations of oversampling and undersampling methods that have proven effective and together may be considered resampling techniques.

Two examples are the combination of SMOTE with Tomek Links undersampling and SMOTE with Edited Nearest Neighbors undersampling.

0.15.1 SMOTE with Tomek Links

Specifically, first the SMOTE method is applied to oversample the minority class to a balanced distribution, then examples in Tomek Links from the majority classes are identified and removed.

[77]:
```
from imblearn.combine import SMOTETomek

smtl = SMOTETomek(random_state=12, ratio = 1.0)
X_train_smtl, y_train_smtl = smtl.fit_sample(X_train, y_train)
```

[78]:
```
pca4 = PCA(n_components=2)
X_train_smtl_PCA = pca4.fit_transform(X_train_smtl)

px.scatter(X_train_smtl_PCA.T[0], X_train_smtl_PCA.T[1],color=y_train_smtl)
```

[79]:
```
print("Shape before Resampling:",X_train.shape,y_train.shape)
print("Class 1:{} , Class 0:{}".
 ↪format(len(y_train[y_train==1]),len(y_train[y_train==0])))
print("Shape after Tomek Links:",X_train_smtl.shape,y_train_smtl.shape)
print("Class 1:{} , Class 0:{}".
 ↪format(len(y_train_smtl[y_train_smtl==1]),len(y_train_smtl[y_train_smtl==0])))
```

```
Shape before Resampling: (227845, 30) (227845,)
Class 1:394 , Class 0:227451
Shape after Tomek Links: (454902, 30) (454902,)
Class 1:227451 , Class 0:227451
```

0.15.2 SMOTE with ENN

SMOTE may be the most popular oversampling technique and can be combined with many different undersampling techniques.

Another very popular undersampling method is the Edited Nearest Neighbors, or ENN, rule. This rule involves using k=3 nearest neighbors to locate those examples in a dataset that are misclassified and that are then removed. It can be applied to all classes or just those examples in the majority class.ENN is more aggressive at downsampling the majority class than Tomek Links, providing more in-depth cleaning.

[7]:
```
from imblearn.combine import SMOTEENN

smen = SMOTEENN(random_state=12, ratio = 1.0)
X_train_smen, y_train_smen = smen.fit_sample(X_train, y_train)
```

[17]:
```
pca5 = PCA(n_components=2)
X_train_smen_PCA = pca5.fit_transform(X_train_smen)

px.scatter(X_train_smen_PCA.T[0], X_train_smen_PCA.T[1],color=y_train_smen)
```

[9]:
```
print("Shape before Resampling:",X_train.shape,y_train.shape)
print("Class 1:{} , Class 0:{}".
 ↪format(len(y_train[y_train==1]),len(y_train[y_train==0])))
print("Shape after Tomek Links:",X_train_smen.shape,y_train_smen.shape)
print("Class 1:{} , Class 0:{}".
 ↪format(len(y_train_smen[y_train_smen==1]),len(y_train_smen[y_train_smen==0])))
```

```
Shape before Resampling: (227845, 30) (227845,)
Class 1:394 , Class 0:227451
Shape after Tomek Links: (454517, 30) (454517,)
Class 1:227451 , Class 0:227066
```

Some of the class 1 labels (yellow circles) near the blue ones have been removed when we used SMOTE with ENN. This gives a much cleaner result than SMOTE with Tomek Links. We will go with SMOTE with ENN and use the transformed training set to train our models.