**Imbalanced Classification in Python: SMOTE-Tomek Links Method**

Combining SMOTE with Tomek Links for imbalanced classification in Python

**Motivation**

In a real-world application, classification modeling often encountered with an imbalanced dataset problem, where the number of majority class is much bigger than the minority class, thus make the model unable to learn from minority class well. This becomes a serious problem when the information in the dataset from the minority class is more important, for example, like disease detection dataset, churn dataset, and fraud detection dataset.

One of the popular approaches to solve this imbalance dataset problem is either to oversample the minority class or undersample the majority class. These approaches, however, have their own weakness. In the vanilla oversampling method, the idea is to duplicate some random examples from the minority class — thus this technique does not add any new information from the data. On the contrary, the undersampling method is conducted by removing some random examples from the majority class, at cost of some information in the original data are removed as well.

One of the solutions to overcome that weakness is to generate new examples that are synthesized from the existing minority class. This method is well known as **Synthetic Minority Oversampling Technique** or **SMOTE**. There are many variations of SMOTE but in this article, I will explain the SMOTE-Tomek Links method and its implementation using Python, where this method combines oversampling method from SMOTE and the undersampling method from Tomek Links.

**The Concept: SMOTE**

SMOTE is one of the most popular oversampling techniques that is developed by Chawla *et al*. (2002). Unlike random oversampling that only duplicates some random examples from the minority class, SMOTE generates examples based on the distance of each data (usually using Euclidean distance) and the minority class nearest neighbors, so the generated examples are different from the original minority class.

In short, the process to generate the synthetic samples are as follows.

1. Choose random data from the minority class.
2. Calculate the Euclidean distance between the random data and its k nearest neighbors.
3. Multiply the difference with a random number between 0 and 1, then add the result to the minority class as a synthetic sample.
4. Repeat the procedure until the desired proportion of minority class is met.

This method is effective because the synthetic data that are generated are relatively close with the feature space on the minority class, thus adding new “information” on the data, unlike the original oversampling method.

**The Concept: Tomek Links**

Tomek Links is one of a modification from Condensed Nearest Neighbors (CNN, not to be confused with Convolutional Neural Network) undersampling technique that is developed by Tomek (1976). Unlike the CNN method that are only randomly select the samples with its k nearest neighbors from the majority class that wants to be removed, the Tomek Links method uses the rule to selects the pair of observation (say, **a**and **b**) that are fulfilled these properties:

1. The observation **a**’s nearest neighbor is **b**.
2. The observation **b**’s nearest neighbor is **a**.
3. Observation **a** and **b** belong to a different class. That is, **a** and **b** belong to the minority and majority class (or *vice versa*), respectively.

Mathematically, it can be expressed as follows.

*Let d(x\_i, x\_j) denotes the Euclidean distance between x\_i and x\_j, where x\_i denotes sample that belongs to the minority class and x\_j denotes sample that belongs to the majority class. If there is no sample x\_k satisfies the following condition:*

*1. d(x\_i, x\_k) < d(x\_i, x\_j), or  
2. d(x\_j, x\_k) < d(x\_i, x\_j)*

*then the pair of (x\_i, x\_j) is a****Tomek Link****.*

This method can be used to find desired samples of data from the majority class that is having the lowest Euclidean distance with the minority class data (i.e. the data from the majority class that is closest with the minority class data, thus make it ambiguous to distinct), and then remove it.

**SMOTE-Tomek Links**

Introduced first by Batista *et al*. (2003), this method combines the SMOTE ability to generate synthetic data for minority class and Tomek Links ability to remove the data that are identified as Tomek links from the majority class (that is, samples of data from the majority class that is closest with the minority class data). The process of SMOTE-Tomek Links is as follows.

1. (**Start of SMOTE**) Choose random data from the minority class.
2. Calculate the distance between the random data and its k nearest neighbors.
3. Multiply the difference with a random number between 0 and 1, then add the result to the minority class as a synthetic sample.
4. Repeat step number 2–3 until the desired proportion of minority class is met. (**End of SMOTE**)
5. (**Start of Tomek Links**) Choose random data from the majority class.
6. If the random data’s nearest neighbor is the data from the minority class (i.e. create the Tomek Link), then remove the Tomek Link.

To understand more about this method in practice, here I will give some example of how to implement SMOTE-Tomek Links in Python using imbalanced-learn library (or imblearn , in short). The model that we will use is Random Forest by using RandomForestClassifier . **For the evaluation procedure, here I will use the Repeated Stratified K-fold Cross Validation method** to ensure that we preserve the percentages of samples for each class in each fold (i.e. each fold must have some samples in each class) with different randomization in each repetition.

**Implementation: Synthetic Dataset**

For the first example, I will use a synthetic dataset that is generated using make\_classification from sklearn.datasets library. First of all, we need to import the libraries (these libraries will be used in the second example as well).

import pandas as pd  
import numpy as np  
from imblearn.pipeline import Pipeline  
import matplotlib.pyplot as plt  
from sklearn.datasets import make\_classification  
from sklearn.model\_selection import cross\_validate  
from sklearn.model\_selection import RepeatedStratifiedKFold  
from sklearn.ensemble import RandomForestClassifier  
from imblearn.combine import SMOTETomek  
from imblearn.under\_sampling import TomekLinks

Next, we generate the synthetic data that we want to use by writing these lines of code.

#Dummy dataset study case  
X, Y = make\_classification(n\_samples=10000, n\_features=4, n\_redundant=0,  
 n\_clusters\_per\_class=1, weights=[0.99], flip\_y=0, random\_state=1)

We can see from the weights parameter that the dataset will consist of 99% data that belong to the majority class, while the rest belong to the minority class.

Here I create two models — **the first one is without using any imbalance data handling, while the other is using the SMOTE-Tomek Links method** — to give you some performance comparison without and with the SMOTE-Tomek Links imbalance handling method.

## No Imbalance Handling  
# Define model  
model\_ori=RandomForestClassifier(criterion='entropy')  
# Define evaluation procedure (here we use Repeated Stratified K-Fold CV)  
cv\_ori=RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)  
# Evaluate model  
scoring=['accuracy','precision\_macro','recall\_macro']  
scores\_ori = cross\_validate(model\_ori, X, Y, scoring=scoring, cv=cv\_ori, n\_jobs=-1)# summarize performance  
print('Mean Accuracy: %.4f' % np.mean(scores\_ori['test\_accuracy']))  
print('Mean Precision: %.4f' % np.mean(scores\_ori['test\_precision\_macro']))  
print('Mean Recall: %.4f' % np.mean(scores\_ori['test\_recall\_macro']))

Without SMOTE-Tomek Links, the model performance that is produced is as follows.

Mean Accuracy: 0.9943  
Mean Precision: 0.9416  
Mean Recall: 0.7480

As we can expect from the imbalanced dataset, **the accuracy metric score is very high, but the recall metric score is pretty low** (around 0.748). This means that the model failed to “learn” the minority class well, thus failed to correctly predict the minority class label.

Let’s see if we can improve the model’s performance by using SMOTE-Tomek Links to handle the imbalanced data.

## With SMOTE-Tomek Links method  
# Define model  
model=RandomForestClassifier(criterion='entropy')  
# Define SMOTE-Tomek Links  
resample=SMOTETomek(tomek=TomekLinks(sampling\_strategy='majority'))  
# Define pipeline  
pipeline=Pipeline(steps=[('r', resample), ('m', model)])  
# Define evaluation procedure (here we use Repeated Stratified K-Fold CV)  
cv=RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)  
# Evaluate model  
scoring=['accuracy','precision\_macro','recall\_macro']  
scores = cross\_validate(pipeline, X, Y, scoring=scoring, cv=cv, n\_jobs=-1)# summarize performance  
print('Mean Accuracy: %.4f' % np.mean(scores['test\_accuracy']))  
print('Mean Precision: %.4f' % np.mean(scores['test\_precision\_macro']))  
print('Mean Recall: %.4f' % np.mean(scores['test\_recall\_macro']))

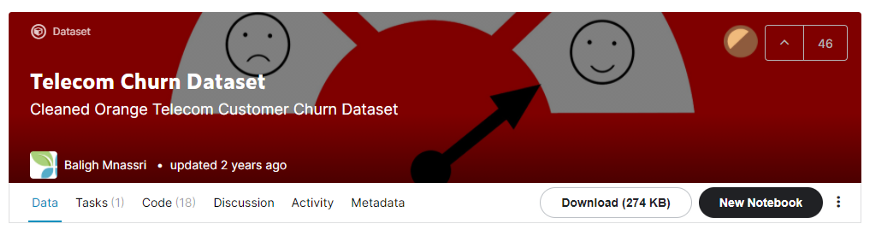
The result is as follows.

Mean Accuracy: 0.9805  
Mean Precision: 0.6499  
Mean Recall: 0.8433

**The accuracy and precision metrics might decrease, but we can see that the recall metric are higher**, it means that the model performs better to correctly predict the minority class label by using SMOTE-Tomek Links to handle the imbalanced data.

**Implementation: Telecom Churn Dataset**

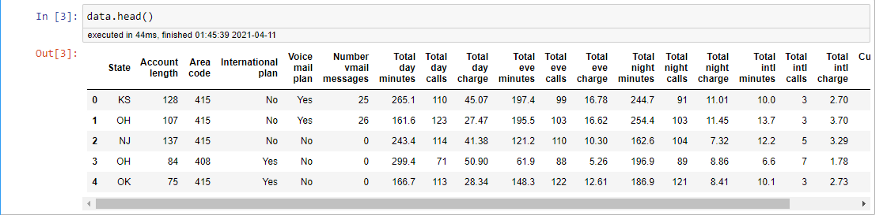
For the second example, here I use the [Telecom Churn Dataset](https://www.kaggle.com/mnassrib/telecom-churn-datasets?select=churn-bigml-80.csv) from Kaggle. There are two data file in this dataset, but in this article, I will use churn-bigml-80.csv data file.



Telecom Churn Dataset (Image taken from [Kaggle](https://www.kaggle.com/mnassrib/telecom-churn-datasets" \t "_blank))

First, we import the library (just like the first example) and the data as follows.

data=pd.read\_csv("churn-bigml-80.csv")  
data.head()



Let’s see the data description to find out the type of each variable.

> data.info()<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 2666 entries, 0 to 2665  
Data columns (total 20 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 State 2666 non-null object   
 1 Account length 2666 non-null int64   
 2 Area code 2666 non-null int64   
 3 International plan 2666 non-null object   
 4 Voice mail plan 2666 non-null object   
 5 Number vmail messages 2666 non-null int64   
 6 Total day minutes 2666 non-null float64  
 7 Total day calls 2666 non-null int64   
 8 Total day charge 2666 non-null float64  
 9 Total eve minutes 2666 non-null float64  
 10 Total eve calls 2666 non-null int64   
 11 Total eve charge 2666 non-null float64  
 12 Total night minutes 2666 non-null float64  
 13 Total night calls 2666 non-null int64   
 14 Total night charge 2666 non-null float64  
 15 Total intl minutes 2666 non-null float64  
 16 Total intl calls 2666 non-null int64   
 17 Total intl charge 2666 non-null float64  
 18 Customer service calls 2666 non-null int64   
 19 Churn 2666 non-null bool   
dtypes: bool(1), float64(8), int64(8), object(3)  
memory usage: 398.5+ KB

Then, we check whether there are exist missing values in the data as follows.

> data.isnull().sum()State 0  
Account length 0  
Area code 0  
International plan 0  
Voice mail plan 0  
Number vmail messages 0  
Total day minutes 0  
Total day calls 0  
Total day charge 0  
Total eve minutes 0  
Total eve calls 0  
Total eve charge 0  
Total night minutes 0  
Total night calls 0  
Total night charge 0  
Total intl minutes 0  
Total intl calls 0  
Total intl charge 0  
Customer service calls 0  
Churn 0  
dtype: int64

No missing values! Next, we calculate the number of data that belong to each class in Churnvariable by writing the line of code as follows.

> data['Churn'].value\_counts()False 2278  
True 388

The data are pretty imbalanced, where the majority class belongs to False label (we will label it as 0) and the minority class belongs to True label (we will label it as 1).

For the next preprocessing step, we drop the State variable (since it contains too many categories), then we recode the Churn variable (False=0, True=1), and create the dummy variables by writing these lines of code.

data=data.drop('State',axis=1)  
data['Churn'].replace(to\_replace=True, value=1, inplace=True)  
data['Churn'].replace(to\_replace=False, value=0, inplace=True)  
df\_dummies=pd.get\_dummies(data)  
df\_dummies.head()#Churn dataset study case  
Y\_churn=df\_dummies['Churn'].values  
X\_churn=df\_dummies.drop('Churn',axis=1)

The data preprocessing is complete. Now, we jump to the modeling with the same approach as the first example.

## No Imbalance Handling  
# Define model  
model2\_ori=RandomForestClassifier(criterion='entropy')  
# Define evaluation procedure (here we use Repeated Stratified K-Fold CV)  
cv2\_ori=RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)  
# Evaluate model  
scoring=['accuracy','precision\_macro','recall\_macro']  
scores2\_ori = cross\_validate(model2\_ori, X\_churn, Y\_churn, scoring=scoring, cv=cv2\_ori, n\_jobs=-1)# summarize performance  
print('Mean Accuracy: %.4f' % np.mean(scores2\_ori['test\_accuracy']))  
print('Mean Precision: %.4f' % np.mean(scores2\_ori['test\_precision\_macro']))  
print('Mean Recall: %.4f' % np.mean(scores2\_ori['test\_recall\_macro']))

Without imbalanced data handling, the result is as follows.

Mean Accuracy: 0.9534  
Mean Precision: 0.9503  
Mean Recall: 0.8572

**Remember that the data that we use are imbalanced, so we cannot simply say that the model performance is good just by observing the accuracy metric**. Although that the accuracy metric score is pretty high, the recall metric score still not high enough, which means that the model is struggling to correctly predict the minority class label (that is, the True label that is recoded to 1).

Now let’s conduct the SMOTE-Tomek Links method for the data to see the performance improvements.

## With SMOTE-Tomek Links method  
# Define model  
model2=RandomForestClassifier(criterion='entropy')  
# Define SMOTE-Tomek Links  
resample2=SMOTETomek(tomek=TomekLinks(sampling\_strategy='majority'))  
# Define pipeline  
pipeline2=Pipeline(steps=[('r', resample2), ('m', model2)])  
# Define evaluation procedure (here we use Repeated Stratified K-Fold CV)  
cv2=RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)  
# Evaluate model  
scoring=['accuracy','precision\_macro','recall\_macro']  
scores2 = cross\_validate(pipeline2, X\_churn, Y\_churn, scoring=scoring, cv=cv2, n\_jobs=-1)# summarize performance  
print('Mean Accuracy: %.4f' % np.mean(scores2['test\_accuracy']))  
print('Mean Precision: %.4f' % np.mean(scores2['test\_precision\_macro']))  
print('Mean Recall: %.4f' % np.mean(scores2['test\_recall\_macro']))

The result is as follows.

Mean Accuracy: 0.9449  
Mean Precision: 0.8981  
Mean Recall: 0.8768

The accuracy and precision score might slightly decrease, but the recall score is increased! That means that the model performs better to correctly predict the minority class label in this Churn dataset.

**Conclusion**

And that’s it! Now you learn how to use the SMOTE-Tomek Links method in Python to increase your classification model performance in the imbalanced dataset. As usual, feel free to ask and/or discuss if you have any questions!

See you in my next article! Stay safe and stay healthy!

**Author’s Contact**

LinkedIn: [Raden Aurelius Andhika Viadinugroho](https://www.linkedin.com/in/raden-aurelius-andhika-viadinugroho-b84b19163/" \t "_blank)

Medium: <https://medium.com/@radenaurelius>

**References**

[1] Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). [SMOTE: Synthetic Minority Over-sampling Technique](https://arxiv.org/abs/1106.1813). *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357.

[2] <https://www.kaggle.com/mnassrib/telecom-churn-datasets>

[3] Tomek, I. (1976). [Two Modifications of CNN](https://ieeexplore.ieee.org/document/4309452). *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 6, no. 11, pp. 769–772.

[4] He, H. and Ma, Y. (2013). [*Imbalanced Learning: Foundations, Algorithms, and Applications*](https://www.wiley.com/en-us/Imbalanced+Learning%3A+Foundations%2C+Algorithms%2C+and+Applications-p-9781118074626). 1st ed. Wiley.

[5] Zeng, M., Zou, B., Wei, F., Liu, X., and Wang, L. (2016). [Effective prediction of three common diseases by combining SMOTE with Tomek links technique for imbalanced medical data](https://ieeexplore.ieee.org/abstract/document/7563084?casa_token=_0SmM6ilaTQAAAAA:sLVb6VptUVi5N8dGd2nMCQuumxpFv0iL66P8gT5Im-xJm58f-OUFMexDf3cEDrSTdPa25RIQuENi). *2016 IEEE International Conference of Online Analysis and Computing Science (ICOACS)*, pp. 225–228.

[6] Batista, G. E. A. P. A., Bazzan, A. L. C., and Monard, M. A. (2003). [Balancing Training Data for Automated Annotation of Keywords: Case Study](http://www.inf.ufrgs.br/maslab/pergamus/pubs/balancing-training-data-for.pdf). *Proceedings of the Second Brazilian Workshop on Bioinformatics*, pp. 35–43.

[7] <https://scikit-learn.org/stable/modules/cross_validation.html>