**Imbalanced data classification: Oversampling and Undersampling**



Photo by [Håkon Grimstad](https://unsplash.com/@grimstad?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com?utm_source=medium&utm_medium=referral)

In an **imbalanced** dataset, the classes are not evenly distributed but rather highly skewed. The [*skewness*](https://en.wikipedia.org/wiki/Skewness)is not particularly the issue here but since one of the classes (referring binary classification) is overly frequent, it will affect the learning mechanism of the Machine Learning algorithm while the business is more interested in the occurrence of the rare class.

I understand that my introductory lines should be explained with a little more depth. The risk in an imbalanced dataset is that, the Machine Learning algorithm may actually ignore the rare class cases and incorrectly classify as frequent class. Let me take an example (fig-1) which is a popular scenario i.e., identifying fraudulent transactions.



(fig-1) image source: self created

As you may understand, it is unlikely to have significant fraudulent transactions in any financial structure. However, it is of utmost interest for the business to effectively identify the fraudulent transaction whenever they occur. You really don’t want your Machine Learning algorithm to incorrectly detect any fraudulent transaction as a valid else it will be a catastrophe for the business. That’s where the challenge of an imbalanced dataset lies.

If I am to explain it more from a technical standpoint, the decision function of a Machine Learning algorithm is hugely impacted by the training samples. Greater the imbalance ratio, the decision function favors the class that is more abundant i.e., the majority class.

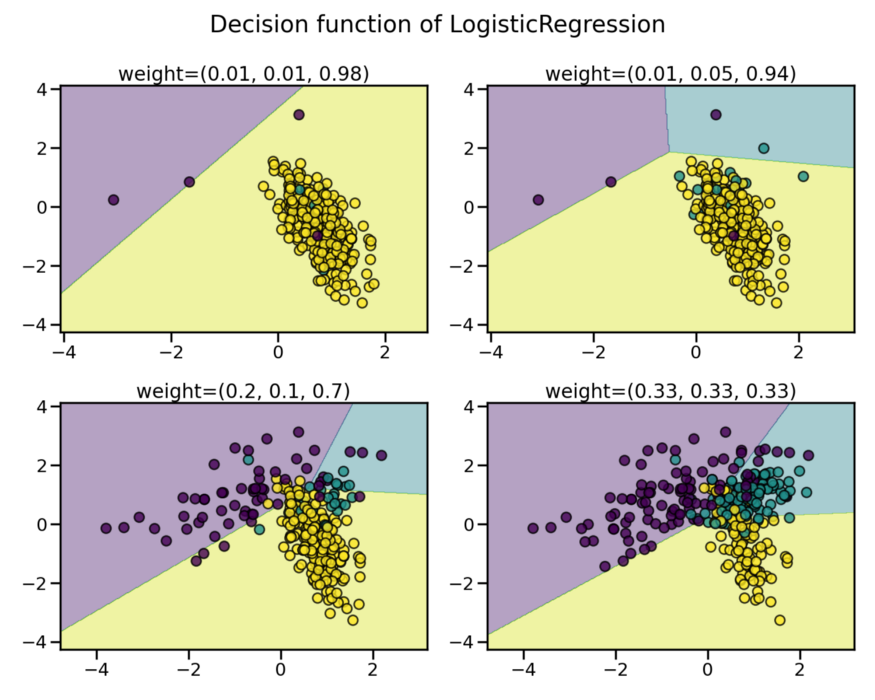


image source: <https://imbalanced-learn.org/>

**Approach**

The idea to combat the challenge of imbalanced data is *random sampling*. Now random resampling can be done in couple of ways (i) Oversampling and (ii) Undersampling

**Oversampling —** Generate new samples for the class which is under-represented.  
**Undersampling** — Remove samples from the class which is over-represented.

Both oversampling & undersampling are ways to infuse bias where you take more samples from one class than the other to neutralize the effect of the imbalance that is either already present in the data or likely to be developed if the samples were taken purely at random.

Random sampling is a naive way to perform sampling as it completely ignores the inner artifacts of the data.

**Setting the stage**

We have already spoke a descent amount about the imbalance data problem, now lets focus on the solution. Here in this discussion, we will use imbalanced-learn API which was started back in 2014, fully compatible with scikit-learn as well. Visit [imbalanced-learn](https://imbalanced-learn.org/stable/index.html) official website for the installation guidelines and entire API documentation.

#Imports  
from collections import Counter  
from sklearn.datasets import make\_classification  
from imblearn.under\_sampling import RandomUnderSampler  
from imblearn.over\_sampling import RandomOverSampler  
  
#Creating a dataset with 2 classes  
X, y = make\_classification(n\_samples=15000, n\_classes=2,   
 weights=[0.99, 0.01], flip\_y = 0)  
  
#class distribution  
print(Counter(y))  
#Output: Counter({0: 14850, 1: 150})  
#class 0 has 14850 rows wheres class 1 has only 150 instances

I have created a binary classification dataset with a high imbalance ratio of 99% for majority class. Before we dig deep into various sampling techniques, lets first try to grab the idea of resampling.

#creating the instance for random over sampler  
randomOverSampler = RandomOverSampler()  
  
#performing resampling  
X\_over, y\_over = randomOverSampler.fit\_resample(X, y)  
Counter(y\_over)  
#Output: Counter({0: 14850, 1: 14850})

The over sampler here created more samples from within the under represented class — fairly simple.

As the over sampler creates copies of the minority class, as a result over sampling technique in a way increases the probability for over-fitting.

The under sampler does just the opposite to the over sampler. It removes the instances from the majority class while resampling from the original dataset.

#creating the instance for random over sampler  
randomUnderSampler = RandomUnderSampler()  
  
#performing resampling  
X\_under, y\_under = randomUnderSampler.fit\_resample(X, y))  
Counter(y\_under)  
#Output: Counter({0: 150, 1: 150})

The under sampler removes huge amount of rows from the majority class and hence it poses serious threat for under-fitting.

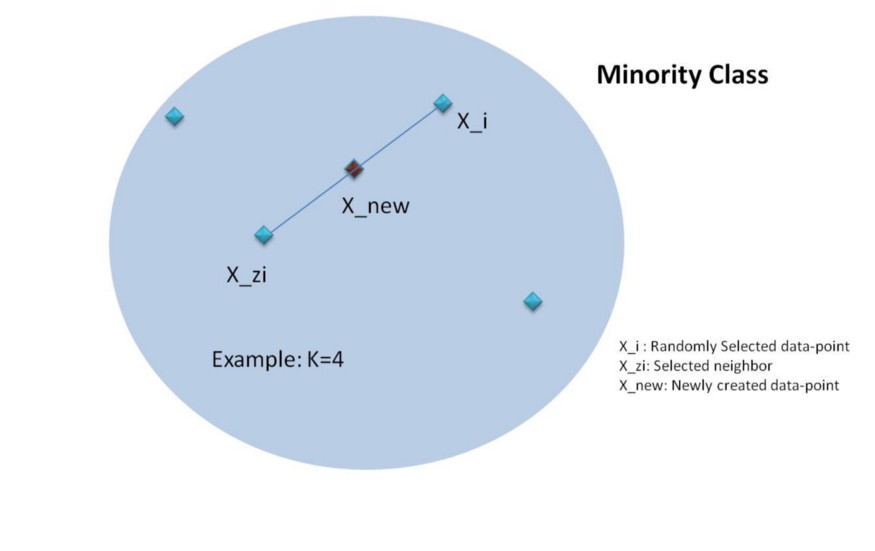
**Different Sampling Techniques**

For majority of the cases, oversampling is preferred more than undersampling. Removing data points is not ideal as it may carry significant piece of information. We will be focusing some of the popular sampling techniques pertaining to imbalanced data classification.

**1. Synthetic Minority Oversampling Technique (SMOTE)**

Instead of creating copies of existing instances of minority class like RandomOverSampler, SMOTE generates new illustrations through interpolation.

The way SMOTE creates synthetic data point is actually a sequence of few steps.  
(i) Select a random instance within the minority class.  
(ii) Identify k nearest neighbors (k=5 by default for KNN) for the randomly selected data point.  
(iii) Select one of those neighbors for the synthetic data point to be created.  
(iv) Calculate the distance vector between the data point and its neighbor.  
(v) Multiply a random number between [0,1] to get the synthetic data point.



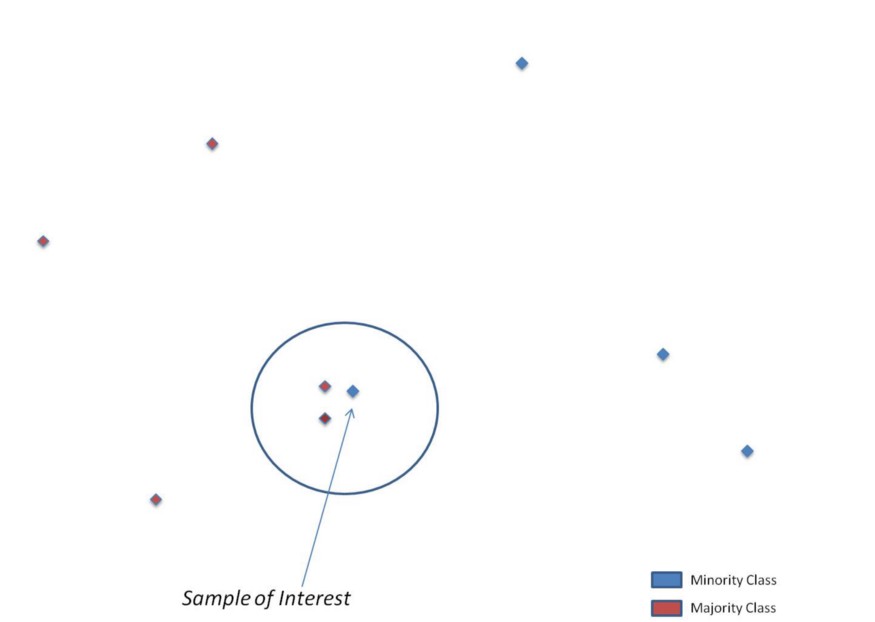
SMOTE Mechanism: self created

from imblearn.over\_sampling import SMOTE  
  
#creating the instance for SMOTE  
smote = SMOTE()  
  
#Resampling with SMOTE  
X\_smote, y\_smote = smote.fit\_resample(X, y)  
Counter(y\_smote)  
#Output: Counter({0: 14850, 1: 14850})

**2. Adaptive Synthetic (ADASYN)**

ADASYN works similar to SMOTE i.e., this algorithm as well looks to create synthetic instances of the minority class. There is one difference though. The synthetic instances are generated for the samples which are not in homogeneous group or difficult to learn.

Here are the sequence of steps that ADASYN follows:  
(i) Calculate the number of synthetic instances to generate (N\_syn). Let N\_maj & N\_min denoting majority and minority classes respectively then  
N\_syn = (N\_maj — N\_min)\*b where b is the class ratio. if b=1 then we will be seeing a perfectly balanced dataset.  
(ii) Find the k nearest neighbors and calculate the d value where  
d\_i = (#majority/k). d\_i value indicates the affluence of the majority class. Greater the value of d\_i, more it will be difficult to learn.



Choosing the minority class instance with most majority class neighbors

(iii) Now, follow the similar steps as SMOTE to calculate the distance vector, then take a random vector between [0,1] and accordingly create the synthetic instance.

from imblearn.over\_sampling import ADASYN  
  
#creating the instance for ADASYN  
adasyn = ADASYN()  
  
#Resampling with SMOTE  
X\_adasyn, y\_adasyn = adasyn.fit\_resample(X, y)  
Counter(y\_adasyn)  
#Output: Counter({0: 14850, 1: 14829})

There are also many more sampling techniques where both oversampling and undersampling techniques are combined — :  
i) SMOTE & Tomek Links (SMOTETomek)  
ii) SMOTE & Edited Nearest Neighbors (SMOTEENN)  
Let me show the python code snippet for both of these.

#SMOTE & Tomek Links  
from imblearn.combine import SMOTETomek  
from imblearn.under\_sampling import TomekLinks  
  
smote\_tomek = SMOTETomek(tomek=TomekLinks(sampling\_strategy='majority'))  
  
X\_smotomek, y\_smotomek = smote\_tomek.fit\_resample(X, y)

#SMOTE & ENN  
from imblearn.combine import SMOTEENN  
  
smoteenn = SMOTEENN()  
  
X\_smotenn, y\_smotenn = smoteenn.fit\_resample(X, y)

**Conclusion**

There are innumerable number of scenarios where you will encounter imbalanced dataset problem. Some of the most common are — detect fraudulent transaction, churn prediction, anomaly detection etc. I have discussed the sampling techniques in this article that are helpful for better detection of minority class in a classification problem.

Accuracy is not a good measure any classification problem which is imbalanced. Hence, you must check recall, precision & f1 score as model metrics. However, it all depends on your business case whether to maximize recall or anything else.