

10 Techniques to Solve Imbalanced Classes in Machine Learning (Updated 2024)

Master Generative AI: Your clear, step-by-step guide to



Home > Classification

> 10 Techniques to Solve Imbalanced Classes in Machine Learning

(Upda...

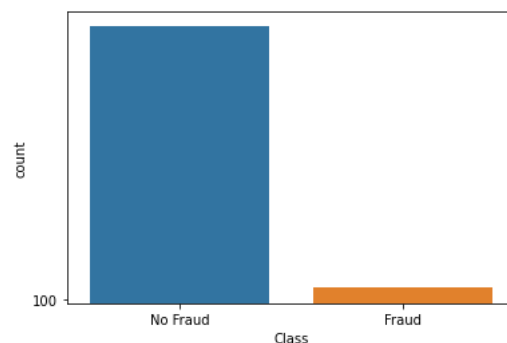


[guest blog](#)

17 Jan, 2024 • 10 min read

Introduction

While working as a data scientist, some of the most frequently occurring problem statements are related to binary classification. A common problem when solving these problem statements is that of class imbalance. When observation in one class is higher than in other classes, a class imbalance exists. Example: To detect fraudulent credit card transactions. As shown in the graph below, the fraudulent transaction is around 400 compared to the non-fraudulent transaction of around 90000.



Class Imbalance is a common problem in machine learning, especially in classification problems. Imbalance data can hamper our model accuracy big time. It appears in many domains, including fraud detection, spam filtering, disease screening, SaaS subscription churn, advertising click-throughs, etc. Let's understand how to deal with imbalanced data in machine learning.

Learning Objectives

10 Techniques to Solve Imbalanced Classes in Machine Learning (Updated 2024)

tutorials in this article.

- Understand various techniques for handling imbalanced data, such as Random under-sampling, Random over-sampling, and NearMiss.

[Table of contents](#)

The Problem With Class Imbalance in Machine Learning

Most machine learning algorithms work best when the number of samples in each class is about equal. This is because most algorithms are designed to maximize accuracy and reduce errors.

However, if the dataframe has imbalanced classes, then In such cases, you get a pretty high accuracy just by predicting the **majority class**, but you fail to capture the **minority class**, which is most often the point of creating the model in the first place. For example, if the class distribution shows that 99% of the data has the majority class, then any basic classification model like the logistic regression or decision tree will not be able to identify the minor class data points.

Credit Card Fraud Detection Example

Let's say we have a [dataset](#) of credit card companies where we have to find out whether the credit card transaction was fraudulent or not.

But here's the catch... fraud transaction is relatively rare. Only 6% of the transactions are fraudulent.

Now, before you even start, do you see how the problem might break? Imagine if you didn't bother training a model at all. Instead, what if you just wrote a single line of code that always predicts 'no fraudulent transaction'

```
def transaction(transaction_data):  
    return 'No fraudulent transaction'
```

10 Techniques to Solve Imbalanced Classes in Machine Learning (Updated 2024)

accuracy:

Unfortunately, that accuracy is misleading.

- For all those **non-fraudulent** transactions, you'd have 100% accuracy.
- For those transactions which are **fraudulent**, you'd have 0% accuracy.
- Your overall **accuracy would be high** simply because most of the transactions are not fraudulent (not because your model is any good).

This is clearly a problem because many machine learning algorithms are designed to maximize overall accuracy. In this article, we will see different techniques to handle imbalanced data.

Sample Dataset

We will use a credit card fraud detection dataset for this article. You can find the dataset [here](#).

After loading the data display the first five-row of the data set.

Python Code:

You can clearly see that there is a huge difference between the data set. 9000 non-fraudulent transactions and 492 fraudulent.

The Metric Trap

One of the major issues that new developer users fall into when dealing with unbalanced datasets relates to the evaluation metrics used to evaluate their machine learning model. Using simpler metrics like **accuracy score** can be misleading. In a dataset with highly unbalanced classes, the classifier will always "predicts" the most common class without performing any analysis of the features, and it will have a high accuracy rate, obviously not the correct one.

Let's do this experiment using the simple XGBClassifier and no feature engineering:

10 Techniques to Solve Imbalanced Classes in Machine Learning (Updated 2024)

```
xgb_model = XGBClassifier().fit(x_train, y_train)

# predict
xgb_y_predict = xgb_model.predict(x_test)

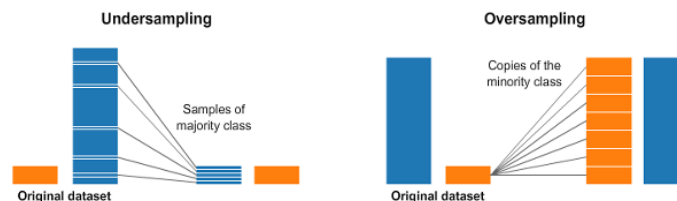
# accuracy score
xgb_score = accuracy_score(xgb_y_predict, y_test)

print('Accuracy score is:', xgb_score)
Accuracy score is: 0.992
```

We can see 99% accuracy, we are getting very high accuracy because it is predicting mostly the **majority** class that is 0 (Non-fraudulent).

Resampling Techniques to Solve Class Imbalance

One of the widely adopted class imbalance techniques 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).

The simplest implementation of **over-sampling** is to duplicate random records from the minority class, which can cause overfitting.

In **under-sampling**, the simplest technique involves removing random records from the majority class, which can cause a loss of information.

Let's implement this with the credit card fraud detection example.

We will start by separating the class that will be 0 and class 1.

```
# class count
class_count_0, class_count_1 = data['Class'].value_counts()
```

10 Techniques to Solve Imbalanced Classes in Machine Learning (Updated 2024)

```
class_1 = data[data['Class'] == 1]# print the shape of the class
print('class 0:', class_0.shape)
print('class 1:', class_1.shape)
```

```
class 0: (9000, 31)
```

```
class 1: (492, 31)
```

1. Random Under-Sampling

Undersampling can be defined as **removing some observations of the majority class**. This is done until the majority and minority class is balanced out.

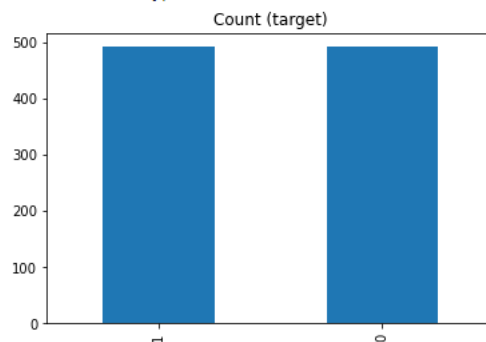
Undersampling can be a good choice when you have a ton of data -think millions of rows. But a drawback to undersampling is that we are removing information that may be valuable.

```
class_0_under = class_0.sample(class_count_1)

test_under = pd.concat([class_0_under, class_1], axis=0)

print("total class of 1 and 0:", test_under['Class'].value_counts)
test_under['Class'].value_counts().plot(kind='bar', title='count')
```

```
total class of 1 and 0:
1    492
0    492
Name: Class, dtype: int64
```



2. Random Over-Sampling

Oversampling can be defined as adding more copies to the minority class. Oversampling can be a good choice when you don't have a ton of data to work with.

A con to consider when undersampling is that it can cause overfitting and poor generalization to your test set.

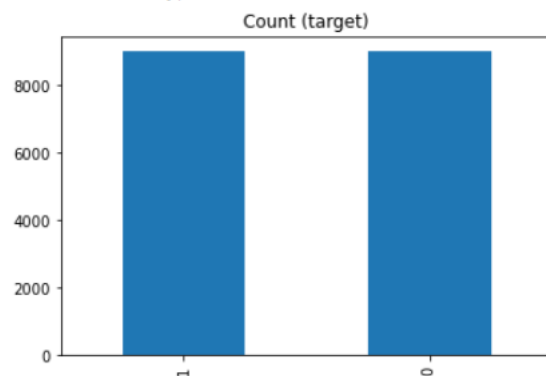
```
class_1_over = class_1.sample(class_count_0, replace=True)

test_over = pd.concat([class_1_over, class_0], axis=0)
```

10 Techniques to Solve Imbalanced Classes in Machine Learning (Updated 2024)

```
test_over = class_balance_tester().plot(name='over', class='count')
```

```
class count of 1 and 0:
1    9000
0    9000
Name: Class, dtype: int64
```



How to Balance Data With the Imbalanced-Learn Python 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.

Let's apply some of these resampling techniques using the Python library [imbalanced-learn](#). It is compatible with scikit-learn and is part of scikit-learn-contrib projects.

```
import imblearn
```



12



3



You may have heard about pandas, numpy, matplotlib, etc. while learning data science. But there is another library: imblearn, which is used to sample imbalanced datasets and improve your model performance.

RandomUnderSampler is a fast and easy way to balance the data by randomly selecting a subset of data for the targeted classes. Under-sample the majority class(es) by randomly picking samples with or without replacement.

10 Techniques to Solve Imbalanced Classes in Machine Learning (Updated 2024)

```
rus = RandomUnderSampler(random_state=42, replacement=True)# fi
x_rus, y_rus = rus.fit_resample(x, y)

print('original dataset shape:', Counter(y))
print('Resample dataset shape', Counter(y_rus))
```

```
original dataset shape: Counter({0: 9000, 1: 492})
Resample dataset shape Counter({0: 492, 1: 492})
```

4. Random Over-Sampling With imblearn

One way to fight imbalanced data is to **generate new samples** in the minority classes. The most naive strategy is to generate new samples by random sampling with the replacement of the currently available samples.

The `RandomOverSampler` offers such a scheme.

```
# import library
from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(random_state=42)

# fit predictor and target variable
x_ros, y_ros = ros.fit_resam

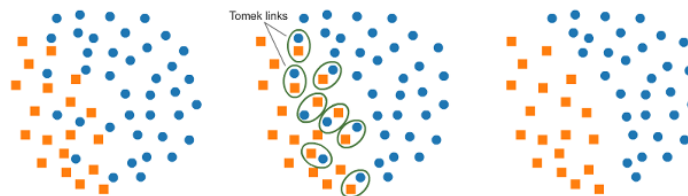
print('Original dataset shape', Counter(y))
print('Resample dataset shape', Counter(y_ros))
```

```
Original dataset shape Counter({0: 9000, 1: 492})
Resample dataset shape Counter({1: 9000, 0: 9000})
```

5. Under-Sampling: 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.

Tomek's link exists if the two samples are the nearest neighbors of each other.



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

```
# import library
from imblearn.under_sampling import TomekLinks
```

10 Techniques to Solve Imbalanced Classes in Machine Learning (Updated 2024)

```
# fit predictor and target variable
x_t1, y_t1 = ros.fit_resample(x, y)

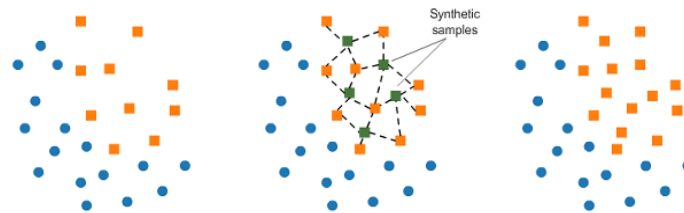
print('Original dataset shape', Counter(y))
print('Resample dataset shape', Counter(y_ros))

Original dataset shape: Counter({0: 9000, 1: 492})
Resample dataset shape: Counter({0: 8839, 1: 492})
```

6. Synthetic Minority Oversampling Technique (SMOTE)

This technique generates synthetic data for the minority class.

SMOTE (Synthetic Minority Oversampling Technique) 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.



SMOTE algorithm works in 4 simple steps:

1. Choose a minority class as the input vector.
2. Find its k nearest neighbors (**k_neighbors** is specified as an argument in the **SMOTE()** function).
3. Choose one of these neighbors and place a synthetic point anywhere on the line joining the point under consideration and its chosen neighbor.
4. Repeat the steps until the data is balanced.

```
# import library
from imblearn.over_sampling import SMOTE

smote = SMOTE()

# fit predictor and target variable
x_smote, y_smote = smote.fit_resample(x, y)

print('Original dataset shape', Counter(y))
print('Resample dataset shape', Counter(y_ros))

Original dataset shape Counter({0: 9000, 1: 492})
Resample dataset shape Counter({1: 9000, 0: 9000})
```


10 Techniques to Solve Imbalanced Classes in Machine Learning (Updated 2024)

NearMiss is an under-sampling technique. Instead of resampling the Minority class, using a distance will make the majority class equal to the minority class.

```
from imblearn.under_sampling import NearMiss

nm = NearMiss()

x_nm, y_nm = nm.fit_resample(x, y)

print('Original dataset shape:', Counter(y))
print('Resample dataset shape:', Counter(y_nm))

Original dataset shape: Counter({0: 16000, 1: 492})
Resample dataset shape: Counter({0: 492, 1: 492})
```

8. Change the Performance Metric

Accuracy is not the best metric to use when evaluating imbalanced datasets, as it can be misleading.

Metrics that can provide better insight are:

- **Confusion Matrix:** a table showing correct predictions and types of incorrect predictions.
- **Precision:** the number of true positives divided by all positive predictions. Precision is also called Positive Predictive Value. It is a measure of a classifier's exactness. Low precision indicates a high number of false positives.
- **Recall:** the number of true positives divided by the number of positive values in the test data. The recall is also called Sensitivity or the True Positive Rate. It is a measure of a classifier's completeness. Low recall indicates a high number of false negatives.
- **F1: Score:** the weighted average of precision and recall.
- **Area Under ROC Curve (AUROC):** AUROC represents the likelihood of your model distinguishing observations from two classes.

In other words, if you randomly select one observation from each class, what's the probability that your model will be able to "rank" them correctly?

9. Penalize Algorithms (Cost-Sensitive Training)

10 Techniques to Solve Imbalanced Classes in Machine Learning (Updated 2024)

that increase the cost of classification mistakes in the minority class.

A popular algorithm for this technique is Penalized-SVM.

During training, we can use the argument `class_weight='balanced'` to penalize mistakes on the minority class by an amount proportional to how under-represented it is.

We also want to include the argument `probability=True` if we want to enable probability estimates for SVM algorithms.

Let's train a model using Penalized-SVM on the original imbalanced dataset:

```
# load library
from sklearn.svm import SVC

# we can add class_weight='balanced' to add panalize mistake
svc_model = SVC(class_weight='balanced', probability=True)

svc_model.fit(x_train, y_train)

svc_predict = svc_model.predict(x_test)# check performance
print('ROCAUC score:',roc_auc_score(y_test, svc_predict))
print('Accuracy score:',accuracy_score(y_test, svc_predict))
print('F1 score:',f1_score(y_test, svc_predict))
```

```
ROCAUC score: 0.548660218804503
Accuracy score: 0.4807517429524098
F1 score: 0.07952713594841485
```

10. Change the Algorithm

While in every machine learning problem, it's a good rule of thumb to try a variety of algorithms, it can be especially beneficial with imbalanced datasets.

Decision trees frequently perform well on imbalanced data. In modern machine learning, tree ensembles (Random Forests, Gradient Boosted Trees, etc.) almost always outperform singular decision trees, so we'll jump right into those:

Tree base algorithm work by learning a hierarchy of if/else questions. This can force both classes to be addressed.

```
# load library
from sklearn.ensemble import RandomForestClassifier
```

10 Techniques to Solve Imbalanced Classes in Machine Learning (Updated 2024)

```
# fit the predictor and target
rfc.fit(x_train, y_train)

# predict
rfc_predict = rfc.predict(x_test) # check performance
print('ROCAUC score:', roc_auc_score(y_test, rfc_predict))
print('Accuracy score:', accuracy_score(y_test, rfc_predict))
print('F1 score:', f1_score(y_test, rfc_predict))
```

```
ROCAUC score: 0.9074057925056814
Accuracy score: 0.9930281903607153
F1 score: 0.8940092165898618
```

Advantages and Disadvantages of Under-Sampling

Advantage:

- It can help improve run time and storage problems by reducing the number of training data samples when the training data set is huge.

Disadvantages:

- It can discard potentially useful information which could be important for building rule classifiers.
- The sample chosen by random under-sampling may be a biased sample. And it will not be an accurate representation of the population. Thereby resulting in inaccurate results with the actual test data set.

Advantages and Disadvantages of Over-Sampling

Advantages:

- Unlike under-sampling, this method leads to no information loss.
- Outperforms under sampling

Disadvantages:

- It increases the likelihood of overfitting since it replicates the minority class events.

Conclusion

To summarize, in this article, we have seen various techniques to handle the class imbalance in a dataset. There are actually many methods to try when dealing with

10 Techniques to Solve Imbalanced Classes in Machine Learning (Updated 2024)

these codes in my GitHub repository [here](#).

Key Takeaways

- In this article, we learned about the different techniques that we can perform to handle class imbalance.
- Some of the most widely used techniques are SMOTE, imblearn oversampling, and undersampling.
- There is no “best” method for handling imbalance, it depends on your use case.

Frequently Asked Questions

class imbalance


imbalanced dataset

imblearn

NearMiss

random undersampling

SMOTE

 [guest blog](#)
17 Jan 2024

Classification

Intermediate

Machine Learning

Python

Structured Data

Frequently Asked Questions

Q1. How do you sample imbalanced data in Python?

A. There are multiple ways to sample imbalanced data, you could apply oversampling methods to the majority class, or you could apply undersampling methods for solving imbalanced classification problems.

Q2. What is the ratio of the classes in the data? How can you find out if your data is balanced or not?

Q3. How do you classify imbalanced data?

Responses From Readers

10 Techniques to Solve Imbalanced Classes in Machine Learning (Updated 2024)

[Submit reply](#)**Shashi**

05 Dec, 2021

What is x and y in the following code: # import library
from imblearn.over_sampling import
RandomOverSampler ros =
RandomOverSampler(random_state=42) # fit predictor
and target variablex_ros, y_ros = ros.fit_resample(x, y)
print('Original dataset shape', Counter(y))
print('Resample dataset shape', Counter(y_ros))

**Akanksha**

11 Jun, 2022

ig x are the feature values for records and y is the
overall labeled data for records.

**Cynthia**

Write for us →

Write, captivate, and earn accolades and rewards for your work

- ✓ Reach a Global Audience
- ✓ Get Expert Feedback
- ✓ Build Your Brand & Audience
- ✓ Cash In on Your Knowledge
- ✓ Join a Thriving Community
- ✓ Level Up Your Data Science Game

**Sion Chakrabarti**

16

**CHIRAG GOY**

87

10 Techniques to Solve Imbalanced Classes in Machine Learning (Updated 2024)

Company	Discover
About Us	Blogs
Contact Us	Expert session
Careers	Podcasts
	Comprehensive Guides
Learn	Engage
Free courses	Community
Learning path	Hackathons
BlackBelt program	Events
Gen AI	Daily challenges
Contribute	Enterprise
Contribute & win	Our offerings
Become a speaker	Case studies
Become a mentor	Industry report
Become an instructor	quexto.ai

Download App

 GET IT ON
Google Play

 Download on the
App Store