

HandlingImbalancedDataSet

October 11, 2024

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

# Set the random seed for reproducibility
np.random.seed(123)

# Create a dataframe with two classes
n_samples = 1000 # total number of instances
class_0_ratio = 0.9

n_class_0 = int(n_samples * class_0_ratio) # number of instances in class 0
n_class_1 = n_samples - n_class_0 # number of instances in class 1

print(n_class_0)
print(n_class_1)
```

```
900
100
```

The line `np.random.seed(123)` is used in the NumPy library (imported as `np`) to set the seed for the random number generator. Setting a seed ensures that the sequence of random numbers generated is reproducible. This is important for debugging, testing, and sharing results, as it allows others (or yourself at a later time) to obtain the same results by generating the same sequence of random numbers.

0.0.1 Explanation of Components:

np.random: This is a submodule of NumPy that provides functions to generate random numbers, including random integers, floats, samples from various probability distributions, and more.

seed(): The `seed()` function initializes the random number generator with a specific starting point. The argument passed to `seed()` (in this case, 123) is an integer that determines the starting point of the sequence of random numbers. Using the same seed value will always produce the same sequence of random numbers, making your experiments repeatable.

123: This is just an arbitrary integer used as the seed. You can choose any integer. The important thing is that using the same integer will yield the same random numbers

every time you run the code. (A seed in the context of random number generation is a starting point or initial value for a pseudorandom number generator (PRNG). When a random number generator is seeded with a specific value, it initializes the internal state of the generator in such a way that it will produce a specific sequence of random numbers)

```
[16]: class_0 = pd.DataFrame(
      {
        'feature_1' : np.random.normal(loc = 0, scale = 1, size = n_class_0),
        'feature_2' : np.random.normal(loc = 0, scale = 1, size = n_class_0),
        'target' : [0] * n_class_0
      }
    )

class_1 = pd.DataFrame(
      {
        'feature_1' : np.random.normal(loc = 1, scale = 1, size = n_class_1),
        'feature_2' : np.random.normal(loc = 1, scale = 1, size = n_class_1),
        'target' : [1] * n_class_1
      }
    )
```

The function `np.random.normal()` from the NumPy library generates random samples from a normal (Gaussian) distribution. This is commonly used in statistical analysis, simulations, and various applications in data science and machine learning.

0.0.2 Parameters:

The function can take several parameters, but the most commonly used ones are: `loc` (mean): This parameter specifies the mean (average) of the normal distribution. It determines the center of the distribution. The default value is 0. `scale` (standard deviation): This parameter specifies the standard deviation of the distribution, which measures the spread or dispersion of the data around the mean. A larger standard deviation indicates that the data points are spread out more widely. The default value is 1. `size`: This parameter specifies the number of random samples to generate. It can be an integer (for a 1D array) or a tuple (for multi-dimensional arrays). The default value is `None`, which returns a single float

```
[18]: df = pd.concat([class_0, class_1])
      df
```

```
[18]:   feature_1  feature_2  target
0   -1.774224   0.285744       0
1   -1.201377   0.333279       0
2    1.096257   0.531807       0
3    0.861037  -0.354766       0
4   -1.520367  -1.120815       0
..      ...      ...      ...
95    1.677156   0.092048       1
96    1.963404  -0.818045       1
```

```

97    0.621476    0.877267    1
98    2.429559    2.794486    1
99    2.532273    0.679490    1

```

[1000 rows x 3 columns]

```
[20]: df['target'].value_counts()
```

```

[20]: target
0      900
1      100
Name: count, dtype: int64

```

```
[ ]: # upsampling
```

Upsampling and downsampling are techniques used in data processing, particularly in the context of handling imbalanced datasets, time series data, and image processing. Here's an in-depth explanation of both concepts:

0.0.3 1. Upsampling

Definition: Upsampling is the process of increasing the number of instances in a dataset. This is typically done to balance a dataset, particularly in classification problems where one class (the minority class) has significantly fewer instances than another class (the majority class).

```

[27]: df_minority = df[df['target'] == 1]
      df_majority = df[df['target'] == 0]

```

```
[29]: df_minority
```

```

[29]:   feature_1  feature_2  target
0    2.131538    2.966215      1
1    0.678490    1.938504      1
2    1.392148    1.161370      1
3    0.458002   -0.518005      1
4   -0.513386    1.541051      1
..      ...      ...      ...
95    1.080243    0.200871      1
96    0.457629    1.371559      1
97    2.456532    0.543461      1
98    2.046854    0.860446      1
99    0.590834    0.307492      1

```

[100 rows x 3 columns]

```
[33]: df_majority
```

```
[33]:
```

	feature_1	feature_2	target
0	-0.941691	-0.460530	0
1	1.038645	-0.910269	0
2	-1.432479	-0.780990	0
3	-1.620503	-0.107118	0
4	-0.279527	-0.657885	0
..
895	0.515323	0.506249	0
896	-1.465313	1.134479	0
897	0.855500	-0.653978	0
898	1.028167	0.015896	0
899	1.814466	-0.110166	0

[900 rows x 3 columns]

```
[39]: from sklearn.utils import resample

df_minority_upsample = resample(df_minority, replace = True, n_samples =
    len(df_majority), random_state = 42)
df_minority_upsample.shape
```

```
[39]: (900, 3)
```

0.0.4 The resample() function is used in data analysis to create a sample of data from a given dataset, often with specific parameters to control the sampling process

df_minority: This is the dataset from which you want to create a sample. It typically represents the minority class in a classification problem (e.g., when dealing with imbalanced datasets). **#### replace=True:** This parameter indicates whether to sample with replacement. If replace=True, it means that after a sample is drawn, it is put back into the dataset, allowing it to be selected again. This is useful for generating additional instances of the minority class to balance it with the majority class. **#### n_samples = len(df_majority):** This parameter specifies the number of samples you want to draw. In your case, it sets the sample size to be equal to the number of instances in the majority class (df_majority). This is a common practice in handling imbalanced datasets, as it aims to create a balanced dataset by generating enough instances of the minority class to match the size of the majority class. **#### random_state=42:** This parameter sets a seed for the random number generator, ensuring that the results are reproducible. Using the same seed value will yield the same random samples each time the code is run.

```
[41]: df_minority_upsample
```

```
[41]:
```

	feature_1	feature_2	target
51	3.718427	0.641871	1
92	-0.002881	0.885766	1
14	3.212847	0.609004	1
71	0.928964	0.203220	1
60	1.524267	3.188399	1
..

```

52  -0.327599    0.888842    1
65   1.153650    1.183450    1
76   0.124318   -0.962565    1
42   1.147497   -0.817325    1
74   0.971470    0.801509    1

```

[900 rows x 3 columns]

```
[45]: df_minority_upsample['target'].value_counts()
```

```
[45]: target
1      900
Name: count, dtype: int64
```

```
[47]: df_upsampled = pd.concat([df_majority, df_minority_upsample])
df_upsampled
```

```
[47]:   feature_1  feature_2  target
0   -0.941691 -0.460530      0
1    1.038645 -0.910269      0
2   -1.432479 -0.780990      0
3   -1.620503 -0.107118      0
4   -0.279527 -0.657885      0
..      ...      ...      ...
52  -0.327599    0.888842      1
65   1.153650    1.183450      1
76   0.124318   -0.962565      1
42   1.147497   -0.817325      1
74   0.971470    0.801509      1

```

[1800 rows x 3 columns]

```
[49]: df_upsampled['target'].value_counts()
```

```
[49]: target
0      900
1      900
Name: count, dtype: int64
```

```
[51]: # Downsampling
```

0.0.5 2. Downsampling

Definition: Downsampling is the process of reducing the number of instances in a dataset. This is often done when a dataset is too large or when the majority class has many more instances than the minority class, leading to potential overfitting in machine learning models.

```
[53]: df_majority_downsample = resample(df_majority, replace = False, n_samples =  
      ↪len(df_minority), random_state = 42)  
      df_majority_downsample.shape
```

```
[53]: (100, 3)
```

```
[55]: df_downsampled = pd.concat([df_majority_downsample, df_minority])  
      df_downsampled
```

```
[55]:
```

	feature_1	feature_2	target
70	-1.243695	-0.588096	0
827	-0.817504	-0.966776	0
231	-0.739751	-0.457629	0
588	0.004720	0.843374	0
39	-1.190833	0.996139	0
..
95	1.080243	0.200871	1
96	0.457629	1.371559	1
97	2.456532	0.543461	1
98	2.046854	0.860446	1
99	0.590834	0.307492	1

```
[200 rows x 3 columns]
```

```
[ ]:
```