Here's what I've learnt about Sklearn.resample

Explaining upsampling, downsampling and some mistakes to avoid

Samson Afolabi Jun 1, 2021 · 4 min read

Photo by Andreas Brunn on Unsplash

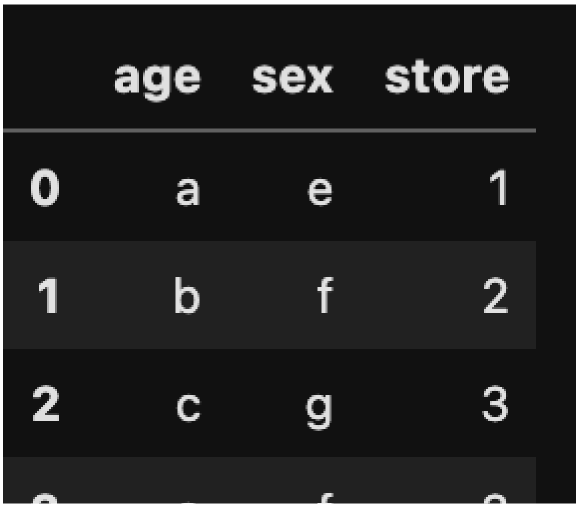
Working with imbalanced dataset can be a tough nut to crack for data scientist. One of class.

> Sklearn.resample is Scikit learn’s function for upsampling/downsampling.

From sklearn documentation, the function sklearn.resample, r*esamples arrays or sparse matrices in a consistent way and the default strategy implements one step of the bootstrapping procedure*. In simple terms, *sklearn.resample* doesn’t just generate extra data points to the datasets by magic, it basically creates a *random resampling(with/without replacement) of your dataset*. This equalization procedure prevents the Machine Learning model from inclining towards the majority class in the dataset.

Next, I show upsampling in an example. In the example below we create a dataframe with 3 columns: **age, sex and store.**

```
#import libraries
import pandas as pd
from sklearn.utils import resample,shuffle

#create a dataframe
df = {'age':['a','b','c','a','b'],'sex':
['e','f','g','f','e'],'store':[1,2,3,3,2]}

df = pd.DataFrame(df)

df.head()
```

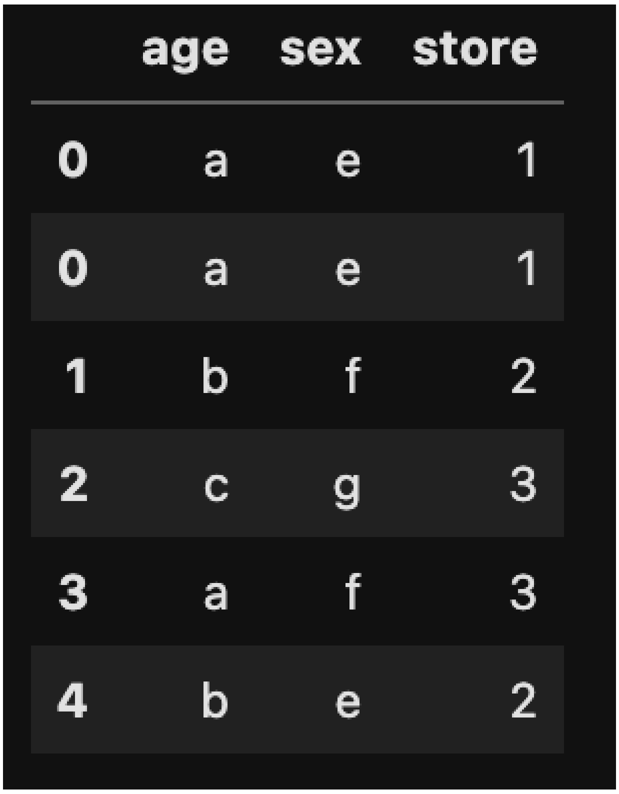


	age	sex	store
0	a	e	1
1	b	f	2
2	c	g	3

df.head()

We first, separate the minority class and then the upsample the minority class. The size of the minority class is upsampled to the size of the other classes.

```
#set the minority class to a seperate dataframe

df_1 = df[df['store'] == 1]

#set other classes to another dataframe

other_df = df[df['store'] != 1]

#upsample the minority class
df_1_upsampled =
resample(df_1,random_state=42,n_samples=2,replace=True)

#concatenate the upsampled dataframe
df_upsampled = pd.concat([df_1_upsampled,other_df])
df_upsampled
```



	age	sex	store
0	a	e	1
0	a	e	1
1	b	f	2
2	c	g	3
3	a	f	3
4	b	e	2

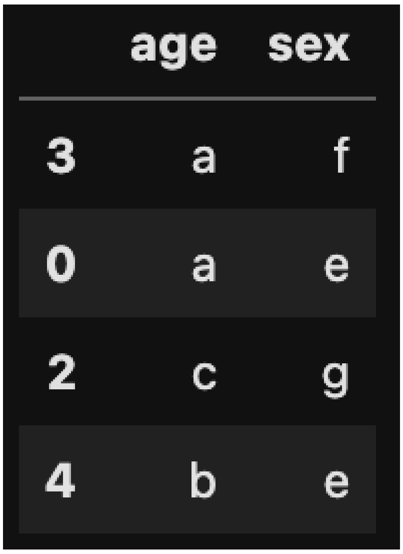
df_upsampled

we train the model.

However, when you upsample or downsample, avoid these mistakes!

1. In a Machine Learning problem, make sure to upsample/downsample **ONLY AFTER** you split into train, test (and validate if you wish). If you do upsample your dataset before you split into train and test, there is a high possibility that your model is exposed to data leakage. See an Example below.

```
from sklearn.model_selection import train_test_split

X = df_upsampled.drop('store',axis=1)
y = df_upsampled.store

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1,shuffle=True)

X_train.head()
```

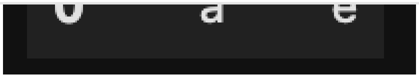


	age	sex
3	a	f
0	a	e
2	c	g
4	b	e

X_train

```
X_test.head()
```

	age	sex



X_test

Notice the data leakage! We have exactly the same data point in **X_train** as well as **X_test**. Doing this might give us a wrong sense of what our ***Machine Learning*** model is really performing.

Photo by Claudio Schwarz | @purzlbaum on Unsplash

2. After your Machine Learning Model is built, it is advisable to test your metric on your **NOT-UPSAMPLED** train dataset. Testing your metric on the **NOT-UPSAMPLED** data set gives you a more realistic estimate of your model than testing it on the **UPSAMPLED** dataset. Personally, I always like to keep a version of the train dataset that wasn’t upsampled.

Conclusion:

Upsampling/downsampling are very good approaches in handling unbalanced data. However it is important to understand how they work, so as to be able to use them

You can also read about the ***SMOTE*** operator of the imbean library. It works based on the ***KNearestNeighbours*** algorithm, synthetically generating data points that fall in the proximity of the already existing outnumbered group. Read more about it here.

I hope this was helpful for you. Looking forward to your comments here, meanwhile you can also follow me on twitter and Linkedin.

Gracias😊

Data Science Machine Learning Unbalanced Data Artificial Intelligence Modelling