What To Do When Your Classification Data is Imbalanced

In this article, we will look at different ways and tools that can be used to solve a common problem that arises in machine learning, the problem of the skewed dataset.

Judy T Raj
Sep 6, 2019 · 4 min read ★

The key to building a good machine learning model is the data it is trained on. Therefore it is imperative that the training data be clean and balanced. The more time you spend on perfecting your training data, the less efforts you'll need to spend on your model. So lets look at how to go about getting ourselves a balanced dataset. In this article we'll discuss,

- What is meant by an imbalanced dataset?
- Why does it matter if the dataset is imbalanced?
- Different ways to deal with an imbalanced dataset.
- Different tools to deal with an imbalanced dataset.

What is meant by an imbalanced dataset?

Lets see what skewness means when trying to solve a classification problem. When the majority of data items in your dataset represents items belonging to one class, we say the dataset is skewed or imbalanced. For better understanding, lets consider a binary classification problem, cancer detection. Say we've five thousand instances in our dataset but only five hundred positive instances, i.e., instances were cancer was actually present. Then we've an imbalanced dataset. This happens more often with datasets in real life as the chances of finding cancer among all the checks that happen

233	10.94	18.59	70.39	370.0	0.10040	1
35	16.16	21.54	106.20	809.8	0.10080	0
132	16.07	19.65	104.10	817.7	0.09168	0
175	14.78	23.94	97.40	668.3	0.11720	0
190	15.85	23.95	103.70	782.7	0.08401	0

```
In [65]: df.diagnosis.value_counts()

Out[65]: 0    212
         1     57
         Name: diagnosis, dtype: int64
```

In this cancer dataset, there are only 57 positive instances whereas there are 212 negative instances, making it a perfect example of class imbalance.

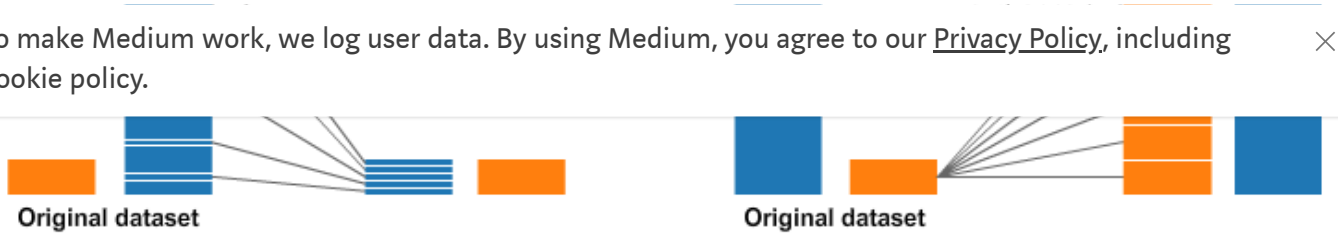
Why does it matter if the dataset is skewed?

When your dataset do not represent all classes of data equally, the model might overfit to the class that's represented more in your dataset and become oblivious to the existence of the minority class. It might even give you a good accuracy but fail miserably in real life. In our example, a model that keeps predicting that there's no cancer every single time will also have a good accuracy as the occurrence of cancer itself will be rare among the inputs. But it will fail when an actual case of cancer is subjected to classification, failing its original purpose.

Different ways to deal with an imbalanced dataset

A widely adopted technique for dealing with highly unbalanced datasets is called resampling. Resampling is done after the data is split into training, test and validation sets. Resampling is done only on the training set or the performance measures could get skewed. Resampling can be of two types: Over-sampling and Under-sampling.

Under sampling involves removing samples from the majority class and over-sampling involves adding more examples from the minority class . 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 loss of information



Under- and Over-Sampling

Another technique similar to upsampling is to create synthetic samples. Adding synthetic samples is also only done after the train-test split, into the training data.

Different tools to deal with an imbalanced dataset

The sklearn.utils.resample package from Scikit Learn lets you resample data. It takes arrays as input and resamples them in a consistent way.

First, lets try over-sampling this dataset.

```
X = df.drop('diagnosis',axis=1)
y = df['diagnosis']

from sklearn.model_selection import train_test_split
from sklearn.utils import resample

#split data into test and training sets
X_train, X_test, y_train, y_test = train_test_split( X, y,
test_size=0.33, random_state=42)

#combine them back for resampling
train_data = pd.concat([X_train, y_train], axis=1)

# separate minority and majority classes
negative = train_data[train_data.diagnosis==0]
positive = train_data[train_data.diagnosis==1]

# upsample minority
pos_upsampled = resample(positive,
 replace=True, # sample with replacement
 n_samples=len(negative), # match number in majority class
 random_state=27) # reproducible results

# combine majority and upsampled minority
upsampled = pd.concat([negative, pos_upsampled])
```

Now we've the same number of instances of both the classes in our train data.

Lets look at under sampling next.

```
# downsample majority
neg_downsampled = resample(negative,
 replace=True, # sample with replacement
 n_samples=len(positive), # match number in minority class
 random_state=27) # reproducible results

# combine minority and downsampled majority
downsampled = pd.concat([positive, neg_downsampled])

# check new class counts
downsampled.diagnosis.value_counts()

1    41
0    41
Name: diagnosis, dtype: int64
```

Both the classes have 41 instances each.

The library imblearn has a class named imblearn.over_sampling.SMOTE which performs over-sampling using SMOTE. This is an implementation of SMOTE or the Synthetic Minority Over-sampling Technique. Lets look at the implementation below.

```
from imblearn.over_sampling import SMOTE

# Separate input features and target
X = df.drop('diagnosis',axis=1)
y = df['diagnosis']

# setting up testing and training sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=27)

sm = SMOTE(random_state=27, ratio=1.0)
X_train, y_train = sm.fit_sample(X_train, y_train)
```

SMOTE has created enough synthetic data for both the classes to have 157 data items each.

We've just covered the most popular way of resampling here. Imblearn has numerous other under- and over- sampling methods defined under the classes imblearn.under_sampling and imblearn.over_sampling as well as methods that combine the two under the class, imblearn.combine.

You can read more about them at, https://imbalanced-learn.readthedocs.io/en/stable/api.html#module-imblearn.over_sampling