Imbalanced Data

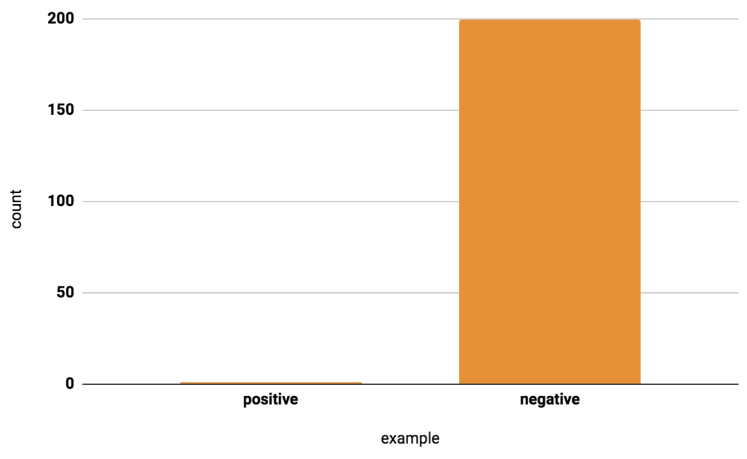
A classification data set with skewed class proportions is called **imbalanced** (/machine-learning/glossary#class_imbalanced_data_set). Classes that make up a large proportion of the data set are called **majority classes** (/machine-learning/glossary#majority_class). Those that make up a smaller proportion are **minority classes** (/machine-learning/glossary#minority_class).

What counts as imbalanced? The answer could range from mild to extreme, as the table below shows.

Degree of imbalance	Proportion of Minority Class
Mild	20-40% of the data set
Moderate	1-20% of the data set
Extreme	<1% of the data set

Why look out for imbalanced data? You may need to apply a particular sampling technique if you have a classification task with an imbalanced data set.

Consider the following example of a model that detects fraud. Instances of fraud happen once per 200 transactions in this data set, so in the true distribution, about 0.5% of the data is positive.



Why would this be problematic? With so few positives relative to negatives, the training model will spend most of its time on negative examples and not learn enough from positive ones. For example, if your batch size is 128, many batches will have no positive examples, so the gradients will be less informative.

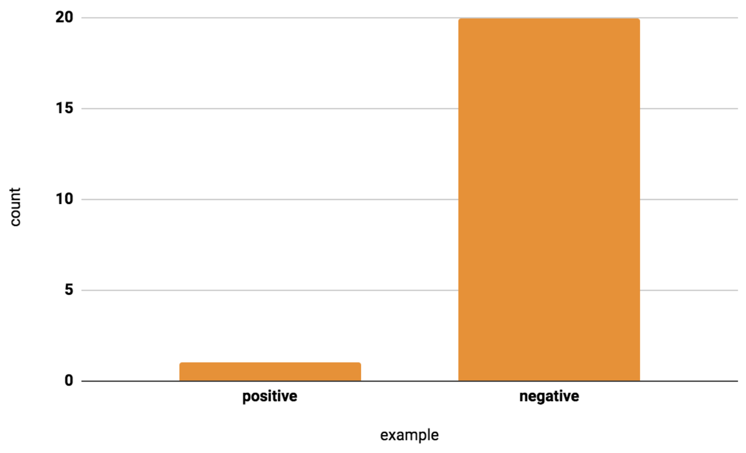
If you have an imbalanced data set, **first try training on the true distribution.** If the model works well and generalizes, you're done! If not, try the following downsampling and upweighting technique.

Downsampling and Upweighting

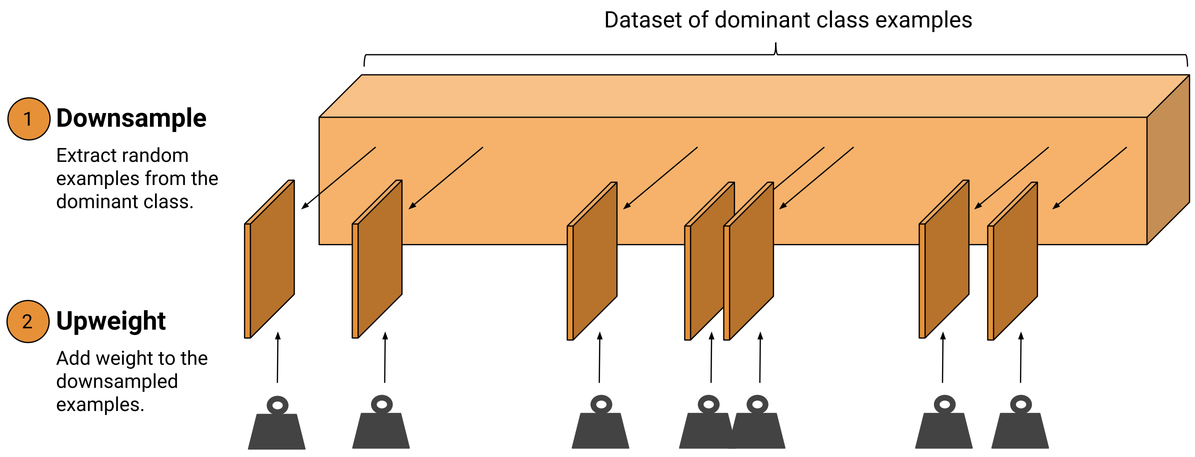
An effective way to handle imbalanced data is to downsample and upweight the majority class. Let's start by defining those two new terms:

- **Downsampling** (in this context) means training on a disproportionately low subset of the majority class examples.
- **Upweighting** means adding an example weight to the downsampled class equal to the factor by which you downsampled.

Step 1: Downsample the majority class. Consider again our example of the fraud data set, with 1 positive to 200 negatives. We can downsample by a factor of 20, taking 1/10 negatives. Now about 10% of our data is positive, which will be much better for training our model.



Step 2: Upweight the downsampled class: The last step is to add example weights to the downsampled class. Since we downsampled by a factor of 20, the example weight should be 20.


Dataset of dominant class examples
1 Downsample
Extract random examples from the dominant class.
2 Upweight
Add weight to the downsampled examples.

You may be used to hearing the term *weight* when it refers to model parameters, like connections in a neural network. Here we're talking about *example weights*, which means counting an individual example more importantly during training. An example weight of 10 means the model treats the example as 10 times as important (when computing loss) as it would an example of weight 1.

The weight should be equal to the factor you used to downsample:

$$\{\text{example weight}\} = \{\text{original example weight}\} \times \{\text{downsampling factor}\}$$

Why Downsample and Upweight?

It may seem odd to add example weights after downsampling. We were trying to make our model improve on the minority class -- why would we upweight the majority? These are the resulting changes:

- **Faster convergence**: During training, we see the minority class more often, which will help the model converge faster.
- **Disk space**: By consolidating the majority class into fewer examples with larger weights, we spend less disk space storing them. This savings allows more disk space for the minority class, so we can collect a greater number and a wider range of examples from that class.
- **Calibration**: Upweighting ensures our model is still calibrated; the outputs can still be interpreted as probabilities.

Previous
← Sampling (/machine-learning/data-prep/construct/sampling-splitting/sampling)
Next
Data Split Example (/machine-learning/data-prep/construct/sampling-splitting/example) →

Last updated 2020-02-10 UTC.